

CASE-BASED REASONING

Cho-k-NN: A Method for Combining Interacting Pieces of Evidence in Case-Based Learning

Eyke Hüllermeier

Otto-von-Guericke-Universität Magdeburg, Fakultät für Informatik
Universitätsplatz 2, 39106 Magdeburg, Germany
eyke.huellermeier@iti.cs.uni-magdeburg.de

Abstract

The case-based learning paradigm relies upon memorizing cases in the form of successful problem solving experience, such as e.g. a pattern along with its classification in pattern recognition or a problem along with a solution in case-based reasoning. When it comes to solving a new problem, each of these cases serves as an individual piece of evidence that gives an indication of the solution to that problem. In this paper, we elaborate on issues concerning the proper combination (aggregation) of such pieces of evidence. Particularly, we argue that cases retrieved from a case library must not be considered as independent information sources, as implicitly done by most case-based learning methods. Focusing on the problem of prediction as a performance task, we propose a new inference principle that combines potentially interacting pieces of evidence by means of the so-called (discrete) Choquet-integral. Our method, called Cho-k-NN, takes interdependencies between the stored cases into account and can be seen as a generalization of weighted nearest neighbor estimation.

1 Introduction

Case-based or instance-based learning algorithms have been applied successfully in fields such as, e.g., machine learning and pattern recognition during the recent years [Aha *et al.*, 1991; Dasarathy, 1991]. The case-based learning (CBL) paradigm is also of central importance in case-based reasoning (CBR), a problem solving methodology which goes beyond the standard prediction problems of classification and regression [Riesbeck and Schank, 1989; Kolodner, 1993].

As the term suggests, in CBL special importance is attached to the concept of a *case*. A case or an instance can be thought of as a single experience, such as a pattern (along with its classification) in pattern recognition or a problem (along with a solution) in CBR.

Rather than inducing a global model (theory) from the data and using this model for further reasoning, as inductive, model-based machine learning methods typically do, CBL systems simply store the data itself. The processing of the data is deferred until a prediction (or some other type of

query) is actually requested, a property which qualifies CBL as a *lazy* learning method [Aha, 1997]. Predictions are then derived by combining the information provided by the stored cases, primarily by those which are *similar* to the new query.

In fact, the concept of similarity plays a central role in CBL. The major assumption underlying CBL has already been expressed, amongst others, by the philosopher DAVID HUME ([Hume, 1999], page 116): “In reality, all arguments from experience are founded on the similarity ... among natural objects. ... From causes, which appear *similar*, we expect similar effects.” This commonsense principle, that we shall occasionally call the “similarity hypothesis” [Rendell, 1986], serves as a basic inference paradigm in various domains of application. For example, in a classification context, it translates into the assertion that “similar objects have similar class labels”, and in CBR it suggests that “similar problems have similar solutions”.

The similarity hypothesis, which is apparently of a heuristic nature, has been put into practice by means of different inference schemes that combine similarity and frequency information in one way or the other. For example, the well-known nearest neighbor (NN) classifier first selects a neighborhood around the query, consisting of the k most similar cases, and then counts the occurrence of the different class labels. Despite of their simplicity, inference methods of such kind have proved to be quite successful in practical applications.

In this paper, we suggest a further improvement based on the idea of taking interdependencies between neighbored cases into account. In fact, in standard NN methods, these cases are implicitly considered as *independent* information sources. We argue that a corresponding assumption of independence is not always justified and propose a new inference principle that combines *interacting* pieces of evidence in a more thorough way. This principle, which makes use of non-additive measures for modeling interaction between cases and employs the so-called (discrete) Choquet-integral as an aggregation operator, can be seen as a generalization of weighted NN estimation.

By way of background, section 2 gives a concise review of the NN principle, which constitutes the core of the family of CBL algorithms. In section 3, we discuss the problem of interaction between cases in CBL. A new approach to CBL, which takes such interactions into account, is then introduced in section 4 and evaluated empirically in section 5.

2 Nearest Neighbor Estimation

The well-known nearest neighbor (NN) estimation principle is applicable to both classification problems (prediction of class labels) and regression (prediction of numeric values).

Consider a setting in which an instance space $\mathcal{X}$ is endowed with a similarity measure $\text{sim} : \mathcal{X} \times \mathcal{X} \rightarrow [0, 1]$. An instance corresponds to the description x of an object (usually in attribute–value form). In the standard classification framework, each instance x is assumed to have a (unique) label $y \in \mathcal{L}$. Here, $\mathcal{L}$ is a finite (typically small) set comprised of m class labels $\{\ell_1 \dots \ell_m\}$, and $\langle x, y \rangle \in \mathcal{X} \times \mathcal{L}$ is a labeled instance (case).

The NN principle originated in the field of pattern recognition [Dasarathy, 1991]. Given a sample S consisting of n labeled instances $\langle x_i, y_i \rangle$, $1 \leq i \leq n$, and a novel instance $x_0 \in \mathcal{X}$ (a query), this principle prescribes to estimate the label y_0 of the yet unclassified query x_0 by the label of the nearest (most similar) sample instance. The k -nearest neighbor (k -NN) approach is a slight generalization, which takes the $k \geq 1$ nearest neighbors of x_0 into account. That is, an estimation y_0^{est} of y_0 is derived from the set $\mathcal{N}_k(x_0)$ of the k nearest neighbors of x_0 , usually by means of a *majority vote*. Besides, further conceptual extensions of the (k -)NN principle have been devised, such as distance weighting [Dudani, 1976]:

$$y_0^{\text{est}} = \arg \max_{\ell \in \mathcal{L}} \sum_{\langle x, y \rangle \in \mathcal{N}_k(x_0)} \omega_x \cdot \mathbb{I}(y = \ell) \quad (1)$$

where ω_x is the weight of the instance x and $\mathbb{I}(\cdot)$ the standard $\{\text{true}, \text{false}\} \rightarrow \{0, 1\}$ mapping. (Throughout the paper, we assume the weights to be given by $\omega_x = \text{sim}(x, x_0)$.)

The NN principle can also be used for regression problems, i.e., for realizing a (locally weighted) approximation of real-valued target functions $x \mapsto y = f(x)$ (in this case, $\mathcal{L} = \mathbb{R}$). To this end, one reasonably computes the (weighted) mean of the k nearest neighbors of a new query point:

$$y_0^{\text{est}} = \frac{\sum_{\langle x, y \rangle \in \mathcal{N}_k(x_0)} \omega_x \cdot y}{\sum_{\langle x, y \rangle \in \mathcal{N}_k(x_0)} \omega_x} \quad (2)$$

3 Interaction Between Cases in CBL

Some case-based approaches completely rely on the (supposedly) most relevant piece of evidence, namely the observation which is most similar to the query. In CBR, for example, it is common practice to retrieve just a single case from the case library, and to adapt the corresponding solution to the problem under consideration. On the one hand, ignoring all but the most similar observation is conceptually simple and computationally efficient. On the other hand, this approach does of course come along with a loss of information, because only a very small part of the past experience is utilized.

If several instead of only a single case are retrieved, as e.g. in k -NN, an important question arises: How should the pieces of evidence coming from the different cases be combined? In k -NN classification, the evidences in favor of a certain class label are simply added up (see (1)). Likewise, in regression the estimation is a simple linear combination of the observed

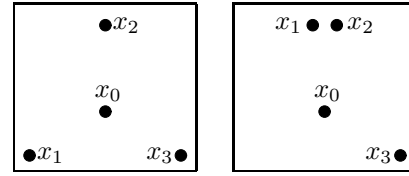


Figure 1: Different configurations of locations in two-dimensional space.

outcomes (see (2)). Thus, the neighbored cases are basically considered as *independent* information sources.

This assumption of independence between case-based evidence can thoroughly be called into question [Hüllermeier, 2002]. Indeed, it is not even in agreement with the similarity hypothesis itself! Namely, if this hypothesis is true, then two neighbored cases that are not only similar to the query case *but also similar among each other* will probably provide similar information regarding the query. In other words, when taking the similarity hypothesis for granted, the information coming from the neighbored cases is at least not independent.

In particular, from a problem solving perspective, one should realize that a set of cases can be *complementary* in the sense that the experiences represented by the individual cases complement or reinforce each other. On the other hand, cases can also be *redundant* in the sense that much of the information is already represented by a smaller subset among them. And indeed, as we said before, the similarity hypothesis suggests that similar cases are likely to be redundant.

To illustrate this point by a simple example, consider the problem of predicting student Peter's grade in computer science, knowing that he has an A in French. The latter information (i.e. the case $\langle \text{French}, A \rangle$) clearly suggests that Peter is an excellent student and, hence, supports predicting an A or maybe a B . Yet, one cannot be sure that this prediction is correct. In this situation, the additional information that Peter has a B in mathematics is probably more valuable than the information that he has an A in Spanish as well. In fact, the cases $\langle \text{French}, A \rangle$ and $\langle \text{Spanish}, A \rangle$ are partly redundant, since the two subjects are quite similar by themselves. As opposed to this, the case $\langle \text{mathematics}, B \rangle$ is complementary in a sense, as it suggests that Peter is not only good in languages.

Now, suppose that you know all three grades (mathematics, Spanish, French). Is A or B the more likely grade? Of course, mathematics is more similar to computer science than Spanish and French. However, depending on the concrete specification of similarity degrees between the various subjects, it is quite possible that the weighted k -NN rule favors grade A since the two moderately similar A 's compensate for the more similar B . Of course, this result might be judged critically, and one might wonder whether the grade A should really count twice. In fact, one reasonable alternative is to consider the two A 's as only a single piece of evidence, telling something about Peter's achievements in languages, instead of two pieces of distinct information. In any case, the close connection between the two A 's should not be ignored when combining the three observations.

As a second example, consider the problem of predicting the yearly rainfall at a certain location (city). For instance,

given the rainfall y_i at location x_i ($i = 1, 2, 3$), what about the rainfall at location x_0 in the two scenarios shown in Fig. 1? The important point to notice is that even though the individual distances between x_0 and the x_i are the same in both scenarios, the y_i should not be combined in the same way. In this example, this is due to the different *arrangements* of the neighbors [Zhang *et al.*, 1997]: Simply predicting the arithmetic mean $(y_1 + y_2 + y_3)/3$ seems to be reasonable in the left scenario, while the same prediction appears questionable in the scenario shown in the right picture. In fact, since x_1 and x_2 are closely neighbored in the latter case, information about the rainfall at these locations will be partly redundant. Consequently, the weight of the (joint) evidence that comes from the observations $\langle x_1, y_1 \rangle$ and $\langle x_2, y_2 \rangle$ should not be twice as high as the weight of the evidence that comes from $\langle x_3, y_3 \rangle$.

The above examples show the need for taking interdependencies between observed cases into account and, hence, provide a motivation for the method that will be proposed in the following section. Before proceeding, let us make two further remarks: First, the above type of interaction between cases seems to be less important if the sample size is large and even becomes negligible in asymptotic analyses of NN principles. In fact, strong results on the performance of NN estimation can be derived [Dasarathy, 1991], but these are valid only under idealized statistical assumptions and arbitrarily large sample sizes. Roughly speaking, if the sample size n tends to infinity, the distance between the query and its nearest neighbors becomes arbitrarily small (with high probability). This holds true even if the size k of the neighborhood is increased too, as a function $k(n)$ of n , provided that $k(n)/n \rightarrow 0$ for $n \rightarrow \infty$. Moreover, if the individual observations are independent and identically distributed in a statistical sense, the neighborhood becomes “well distributed”. Under these assumptions, it is intuitively clear that interdependencies between observations will hardly play any role. On the other hand, it is also clear that statistical assumptions of such kind will almost never be satisfied in practice.

The second remark concerns related work. In fact, there are a few methods that fit into the CBL framework and that allow for taking certain types of interaction between observations into account. Particularly, these are methods that make assumptions on the statistical correlation between observations, depending on their distance [Lindenbaum *et al.*, 1999]. For example, in our rainfall example one could employ a method called *kriging*, which is well-known in geostatistics [Oliver and Webster, 1990]. Usually, however, such methods are specialized on a particular type of application and, moreover, make rather restrictive assumptions on the mathematical (metric) structure of the instance space. Our approach, to be detailed in the next section, is much more general in the sense that it only requires a similarity measure $\text{sim}(\cdot)$ to be given. We do not make any particular assumptions on this measure (such as symmetry or any kind of transitivity), apart from the fact that it should be normalized to the range $[0, 1]$. From an application point of view, this seems to be an important point. In CBR, for example, cases are typically complex objects that cannot easily be embedded into a metric space.

4 The Cho-k-NN Method

4.1 Non-Additive Measures

Let X be a finite set and $\nu(\cdot)$ a measure $2^X \rightarrow [0, 1]$. For any $A \subseteq X$, we interpret $\nu(A)$ as the *weight* or, say, the *degree of relevance* of the set of elements A .

A standard assumption on the measure $\nu(\cdot)$, which is made e.g. in probability theory, is additivity: $\nu(A \cup B) = \nu(A) + \nu(B)$ for all $A, B \subseteq X$ s.t. $A \cap B = \emptyset$. Unfortunately, additive measures cannot model any kind of interaction between elements: Extending a set of elements A by a set of elements B always increases the weight $\nu(A)$ by the weight $\nu(B)$, regardless of A and B .

Suppose, for example, that the elements of two sets A and B are *complementary* in a certain sense. To illustrate, one might think of X as a set of workers in a factory, and of a weight $\nu(A)$ as the productivity of a team of workers A . In that case, complementarity means that the output produced by two teams A and B is higher if they cooperate. Formally, this can be expressed as a positive interaction: $\nu(A \cup B) > \nu(A) + \nu(B)$. Likewise, elements can interact in a negative way. For example, if two sets A and B are partly *redundant* or *competitive*, then $\nu(A \cup B) < \nu(A) + \nu(B)$.

The above considerations motivate the use of non-additive measures, also called fuzzy measures, which are simply normalized and monotone [Sugeno, 1974]:

- $\nu(\emptyset) = 0, \nu(X) = 1$
- $\nu(A) \leq \nu(B)$ for $A \subseteq B$

In order to quantify the interaction between subsets of a set X , as induced by a fuzzy measure $\nu(\cdot)$, several indices have been proposed in the literature. For two individual elements $x_i, x_j \in X$, the *interaction index* is defined by

$$I_\nu(x_i, x_j) \stackrel{\text{df}}{=} \sum_{A \subseteq X \setminus \{x_i, x_j\}} \frac{(|X| - |A| - 1)! |A|!}{(|X| - 1)!} (\Delta_{ij}\nu)(A),$$

where $|A|$ denotes the cardinality of A and

$$(\Delta_{ij}\nu)(A) \stackrel{\text{df}}{=} \nu(A \cup \{x_i, x_j\}) - \nu(A \cup \{x_i\}) - \nu(A \cup \{x_j\}) + \nu(A).$$

The latter can be seen as the *marginal interaction* between x_i and x_j in the context A [Murofushi and Soneda, 1993]. The above index can be extended from pairs x_i, x_j to any set $U \subseteq X$ of elements [Grabisch, 1997]:

$$I_\nu(U) \stackrel{\text{df}}{=} \sum_{A \subseteq X \setminus U} \frac{(|X| - |A| - |U|)! |A|!}{(|X| - |U| + 1)!} (\Delta_U\nu)(A),$$

where

$$(\Delta_U\nu)(A) \stackrel{\text{df}}{=} \sum_{V \subseteq U} (-1)^{|U| - |V|} \nu(V \cup A).$$

4.2 Modeling Interaction in Case-Based Learning

Now, recall our actual problem of combining evidence in case-based learning. In this context, the set X of elements corresponds to the neighbors of the query case x_0 :

$$X = \mathcal{N}_k(x_0) = \{x_1, x_2 \dots x_k\} \quad (3)$$

Our comments so far have shown that fuzzy measures can principally be used for modeling interaction between cases. The basic question that we have to address in this connection is the following: What is the *evidence weight* or simply the *weight*, $\nu(A)$, of a subset A of the neighborhood (3)?

First, the boundary conditions $\nu(\emptyset) = 0$ and $\nu(X) = 1$ should of course be satisfied, expressing that the full evidence is provided by the complete neighborhood X . Moreover, according to our comments on the similarity hypothesis (section 3), the evidence coming from a set of cases $A \subseteq X$ should be discounted if these cases are similar among themselves. Likewise, the weight of A should be increased if the cases are “diverse” (hence complementary) in a certain sense.¹ To express this idea in a more rigorous way, we define the *diversity* of a set of cases A by the average pairwise dissimilarity:

$$\text{div}(A) \stackrel{\text{df}}{=} \frac{2}{|A|^2 - |A|} \sum_{x_i, x_j \in A} 1 - \text{sim}(x_i, x_j)$$

(By definition, the diversity is 0 for singletons and the empty set.) We furthermore define the *relative diversity* by

$$\text{rdiv}(A) \stackrel{\text{df}}{=} \frac{2 \text{div}(A)}{1 - \min_{i,j} \text{sim}(x_i, x_j)} - 1 \in [-1, 1]$$

Now, the idea is to modify the basic (additive) measure

$$\mu(A) \stackrel{\text{df}}{=} c^{-1} \sum_{x_i \in A} \text{sim}(x_0, x_i), \quad (4)$$

where $c = \sum_{x_i \in X} \text{sim}(x_0, x_i)$, by taking the diversity of A into account. Of course, this can be done in different ways. Here, we used the following approach:

$$\bar{\nu}(A) \stackrel{\text{df}}{=} \mu(A) \cdot (1 + \alpha \text{rdiv}(A)) \quad (5)$$

As can be seen, the original measure $\mu(A)$ of a set of cases A is increased if the diversity of A is relatively high, otherwise it is decreased. The parameter $\alpha \geq 0$ controls the extent to which interactions between cases are taken into consideration: $\mu(A)$ can be modified by at most $(100\alpha)\%$. For $\alpha = 0$, interactions are completely ignored and the original measure $\mu(\cdot)$ is recovered.

For the measure $\bar{\nu}(\cdot)$ as defined in (5), the monotonicity condition does not necessarily hold. To remedy this problem, we simply enforce this property by setting

$$\nu(A) \stackrel{\text{df}}{=} \max_{B \subseteq A} \bar{\nu}(B). \quad (6)$$

Finally, the boundary conditions are guaranteed by dividing the measure thus obtained by $\nu(X)$.

To illustrate, consider again the rainfall example in the right picture of Fig. 1 and suppose that $\text{sim}(x_i, x_0) = .5$ ($i = 1, 2, 3$), $\text{sim}(x_1, x_2) = .9$, $\text{sim}(x_1, x_3) = \text{sim}(x_2, x_3) = 0$. With $\alpha = 1/2$ in (5), we obtain the following weights:

A	x_1	x_2	x_3	x_1, x_2	x_1, x_3	x_2, x_3
$\nu(A)$	.27	.27	.27	.33	.83	.83

¹The idea of “diversity” of a set of cases plays also a role in case-based retrieval [McSherry, 2002]. Here, however, the problem is more to *find* diverse cases, rather than to *combine* them.

As can be seen, the joint weight of $\{x_1, x_2\}$ is relatively low, reflecting the partial redundance of the two cases.

Before going on, let us comment on the derivation of the measure $\nu(\cdot)$ from the similarity function $\text{sim}(\cdot)$. Firstly, even though the measure (6) captures our intuitive idea of decreasing (increasing) the evidence weight of cases that are (dis-)similar by themselves, we admit that it remains ad hoc to some extent, and by no means we exclude the existence of better alternatives. For example, an interesting idea is to derive the measure in an indirect way: First, the interaction indices $I_\nu(\cdot)$ from section 4.1 are defined, again based on the similarity between cases. These indices can be seen as constraints on the measure $\nu(\cdot)$, and the idea is to find a measure that is maximally consistent with these constraints.

Secondly, even though the assumption that similar cases provide redundant information is supported by the similarity hypothesis, one might of course argue that the similarity between the predictive parts of two cases, x_i and x_j , is not sufficient to call them redundant. Rather, the associated output values y_i and y_j should be similar as well. Indeed, if y_1 differs drastically from y_2 , the first two measurements in our rainfall example might better be considered as non-redundant. (In that case, the two measurements in conjunction suggest that there is something amiss ...) This conception of redundancy can easily be represented by deriving $\nu(\cdot)$ from an extended similarity measure $\text{sim}'(\cdot)$ which is defined over $\mathcal{X} \times \mathcal{L}$.²

Anyway, the important point of this section is not so much the specification of a *particular* evidence measure, but rather the insight that non-additive measures can *in principle* be used for modeling the interaction between cases in CBL.

4.3 Aggregation of Interacting Pieces of Evidence

So far, we have a tool for modeling the interaction between different pieces of evidence in case-based learning. The next question that we have to address is how to *combine* these pieces of evidence, i.e., how to aggregate them in agreement with the evidence measure $\nu(\cdot)$.

For the time being we focus on the problem of regression. Recall that in the standard approach to NN estimation, an aggregation of the output values $f(x_i) = y_i$ is realized by means of a simple weighted average:

$$y_0^{\text{est}} = \sum_{x_i \in X} \mu(\{x_i\}) \cdot f(x_i), \quad (7)$$

where $\mu(\{x_i\}) = \text{sim}(x_0, x_i) (\sum_{x_i \in X} \text{sim}(x_0, x_i))^{-1}$. Interestingly, (7) is nothing else than the standard Lebesgue integral of the function $f : X \rightarrow \mathfrak{R}$ with respect to the additive measure (4):

$$y_0^{\text{est}} = \int_X f d\mu$$

In order to generalize this estimation, an integral with respect to the non-additive measure $\nu(\cdot)$ is needed: the Choquet integral, a concept that originated in capacity theory [Choquet, 1954].

²We didn’t explore this alternative in detail so far.

Let $\nu(\cdot)$ be a fuzzy measure and $f(\cdot)$ a non-negative function.³ The Choquet integral of $f(\cdot)$ with respect to $\nu(\cdot)$ is then defined by

$$\int^{ch} f d\nu \stackrel{\text{df}}{=} \int_0^\infty \eta([f > t]) dt$$

where $[f > t] = \{x \mid f(x) > t\}$. The integral on the right-hand side is the standard Lebesgue integral (with respect to the Borel measure on $[0, \infty)$). In our case, where X is a finite set, we can refer to the *discrete* Choquet integral which can be expressed in a rather simple form:

$$y_0^{est} = \sum_{i=1}^k f(x_{\pi(i)}) \cdot (\nu(A_i) - \nu(A_{i-1})), \quad (8)$$

where $\pi(\cdot)$ is a permutation of $\{1 \dots k\}$ such that $0 \leq f(x_{\pi(1)}) \leq \dots \leq f(x_{\pi(k)})$, and $A_i = \{x_{\pi(1)} \dots x_{\pi(i)}\}$.

The discrete Choquet integral (8) can be seen as a special type of aggregation operator, namely a generalized arithmetic mean. Indeed, (8) coincides with (7) if $\nu(\cdot)$ is an additive measure (i.e. if $\nu(\cdot) = \mu(\cdot)$). Otherwise, it is a proper generalization of the standard (weighted) NN estimation.

Coming back to our running example, suppose that we have measured the following rainfalls: $y_1 = 100$, $y_2 = 120$, $y_3 = 200$. According to (8), we then obtain the estimation $y_0^{est} \approx 157$. As the joint weight of the two locations with less rainfall, x_1 and x_2 , has been decreased, this estimation is higher (closer to y_3) than the standard weighted NN estimation given by $y_0^{est} = 140$.

So far, we have focused on the problem of regression. In the case of classification, the Choquet integral cannot be applied immediately, since an averaging of class labels y_i does not make sense. Instead, the Choquet integral can be derived for each of the indicator functions $f_\ell : y \mapsto \mathbb{I}(y = \ell)$, $\ell = \ell_1 \dots \ell_m$. As in (1), the evidence in favor of each class label is thus accumulated separately. Now, however, the interaction between cases is taken into account. As usual, the estimation is then given by the label with the highest degree of accumulated evidence.

5 Empirical Validation

In order to validate the extension of NN estimation as proposed in the previous section, we have performed several experimental studies using benchmark data sets from the UCI repository⁴ and the StatLib archive.⁵

Experiments were performed in the following way: A data set is randomly split into a training and a test set of the same size. For each example in the test set, a prediction is derived using the training set in combination with weighted k -NN resp. Cho- k -NN. In the case of regression, an estimation y_0^{est} is evaluated by the relative estimation error $|y_0^{est} - y_0| \cdot |y_0|^{-1}$, and the overall performance of a method by the mean of this error over all test examples. In the case of classification, we

³The Choquet integral can be extended to any real-valued function through decomposition into a positive and negative part.

⁴<http://www.ics.uci.edu/~mllearn>

⁵<http://stat.cmu.edu/>

data set	k	weighted k -NN	Cho- k -NN
auto-mpg	5	12.21 (0.05)	11.56 (0.05)
	7	12.18 (0.06)	11.53 (0.05)
bolts	5	47.07 (1.19)	38.77 (0.71)
	7	51.36 (1.25)	39.94 (0.81)
housing	5	14.83 (0.08)	14.48 (0.08)
	7	14.99 (0.09)	14.62 (0.08)
detroit	5	16.02 (0.55)	14.90 (0.50)
	7	15.93 (0.55)	14.71 (0.54)
echomonths	5	97.77 (7.17)	72.87 (3.87)
	7	99.03 (8.98)	74.80 (7.55)
pollution	5	4.12 (0.05)	4.05 (0.05)
	7	4.22 (0.05)	4.18 (0.05)

Table 1: Estimation of expected relative estimation error and its standard deviation.

simply took the misclassification rate as a performance index. Moreover, we derived statistical estimations of the *expected* performance of a method by repeating each experiment 100 times.

For the purpose of similarity computation, all numeric attributes have first been normalized to the unit interval by linear scaling. The similarity was then defined by 1–distance for numeric variables and by the standard 0/1-measure in the case of categorical attributes. The overall similarity $\text{sim}(\cdot)$ was finally obtained by the average over all attributes. As the purpose of our study was to *compare* –under equal conditions– weighted k -NN with Cho- k -NN in order to verify whether or not taking interactions into account is useful, we refrained from tuning both methods, e.g. by including feature selection or feature weighting (even though it is well-known that such techniques can greatly improve performance [Wettschereck *et al.*, 1997]). Results have been derived for neighborhood sizes of $k = 5$ and $k = 7$; the parameter α in (5) has always been set to $1/2$.

The application of Cho- k -NN for regression has shown that it consistently improves weighted k -NN, sometimes only slightly but often even considerably. Some results are shown in table 1. In particular, it seems that the smaller the size of the data set, the higher the gain in performance. This finding is intuitively plausible, since for large data sets the neighborhoods of a query tend to be more “balanced”; as already said, the neglect of interaction is likely to be less harmful under such circumstances.

For classification problems, it is also true that Cho- k -NN consistently outperforms weighted k -NN; see table 2. Usually, however, the gain in classification accuracy is only small, in many cases not even statistically significant. Again, this is especially true for large data sets, and all the more if the classification error is already low for standard k -NN. Nevertheless, one should bear in mind that, in the case of classification, the final prediction is largely insensitive toward modifications of the estimated evidences in favor of the potential labels. In fact, in this study we only checked whether the final prediction is correct or not and, hence, used a rather crude quality measure. More subtle improvements of an estimation such as, e.g., the enlargement of an example’s margin [Schapire *et*

data set	k	weighted k -NN	Cho- k -NN
glass	5	33.58 (0.34)	33.23 (0.35)
	7	34.54 (0.37)	33.77 (0.39)
wine	5	3.52 (0.20)	3.48 (0.20)
	7	3.27 (0.20)	3.25 (0.20)
zoo	5	10.43 (0.60)	10.08 (0.60)
	7	11.65 (0.51)	11.35 (0.52)
ecoli	5	16.30 (0.25)	16.29 (0.24)
	7	16.17 (0.24)	16.11 (0.24)
balance	5	15.62 (0.14)	15.23 (0.16)
	7	13.28 (0.16)	13.15 (0.16)
derma	5	3.75 (0.13)	3.68 (0.13)
	7	3.57 (0.12)	3.46 (0.11)

Table 2: Estimation of expected classification error and its standard deviation.

al., 1998], are not honored by this measure. For the future, we therefore plan to complement our results by more sophisticated experimental comparisons.

6 Concluding Remarks

This paper has motivated the consideration of mutual dependencies between cases that represent past experience in case-based learning. The basic idea is that a combination of two or more cases can provide *complementary* but also *redundant* evidence. In order to model this type of interaction in a formal way, we have proposed the use of non-additive measures. The aggregation of different pieces of evidence can then be accomplished by means of the Choquet integral. The inference scheme thus obtained, referred to as CHO- k -NN, is a direct extension of the standard weighted NN estimation (which is recovered in the case of an additive measure).

Our experimental results have shown that CHO- k -NN consistently outperforms standard (weighted) k -NN on publicly available benchmark data. Sometimes, only marginal improvements can be achieved, particularly in the case of classification and all the more for large, “well-distributed” data sets, but for many problems CHO- k -NN is considerably better than k -NN. All things considered, it can be said that taking interaction between cases in CBL into account is worthwhile by any means (mostly it helps, at worst it remains ineffective).

As already said, the derivation of a non-additive measure $\nu(\cdot)$ from the similarity function $\text{sim}(\cdot)$ as outlined in section 4 is only a first attempt which leaves scope for development. In particular, the degree to which a set of cases is complementary resp. redundant might not only depend on their mutual similarity but also on other aspects.

We have explained how to use the Choquet integral as an aggregation operator in both regression and classification problems. An interesting question concerns the extension of our approach to more general problem solving tasks as they are typically found in case-based reasoning. Even though it is obvious that the approach cannot be transferred immediately, the basic ideas and concepts might still be useful. In any case, the concrete solution will strongly depend on the particular type of application.

References

- [Aha *et al.*, 1991] D.W. Aha, D. Kibler, and M.K. Albert. Instance-based learning algorithms. *Machine Learning*, 6(1):37–66, 1991.
- [Aha, 1997] D.W. Aha, editor. *Lazy Learning*. Kluwer Academic Publ., 1997.
- [Choquet, 1954] G. Choquet. Theory of capacities. *Annales de l’Institut Fourier*, 5:131–295, 1954.
- [Dasarathy, 1991] B.V. Dasarathy, editor. *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques*. IEEE Computer Society Press, Los Alamitos, California, 1991.
- [Dudani, 1976] S.A. Dudani. The distance-weighted k-nearest-neighbor rule. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-6(4):325–327, 1976.
- [Grabisch, 1997] M. Grabisch. k-order additive discrete fuzzy measures and their representation. *Fuzzy Sets and Systems*, 92:167–189, 1997.
- [Hüllermeier, 2002] E. Hüllermeier. On the representation and combination of evidence in instance-based learning. In *Proc. ECAI–2002*, pages 360–364, Lyon, France, 2002.
- [Hume, 1999] D. Hume. *An Enquiry concerning Human Understanding*. Oxford Univ. Press Inc., New York, 1999.
- [Kolodner, 1993] J.L. Kolodner. *Case-based Reasoning*. Morgan Kaufmann, San Mateo, 1993.
- [Lindenbaum *et al.*, 1999] M. Lindenbaum, S. Marcovich, and D. Rusakov. Selective sampling for nearest neighbor classifiers. *AAAI-99*, pages 366–371, Orlando, 1999.
- [McSherry, 2002] D. McSherry. Diversity-conscious retrieval. *ECCBR-02*, pages 219–233. Springer, 2002.
- [Murofushi and Soneda, 1993] T. Murofushi and S. Soneda. Techniques for reading fuzzy measures (iii): interaction index. In *9th Fuzzy System Symposium*, pages 693–696. Sapporo, Japan, 1993.
- [Oliver and Webster, 1990] M. Oliver, R. Webster. Kriging: a method of interpolation for geographical information system. *Int. J. Geogr. Inform. Syst.*, 4(3):313–332, 1990.
- [Rendell, 1986] L. Rendell. A general framework for induction and a study of selective induction. *Machine Learning*, 1:177–226, 1986.
- [Riesbeck and Schank, 1989] C.K. Riesbeck and R.C. Schank. *Inside Case-based Reasoning*. Hillsdale, New York, 1989.
- [Schapire *et al.*, 1998] RE. Schapire, Y. Freund, P. Bartlett, and WS. Lee. Boosting the margin: a new explanation for the effectiveness of voting methods. *Annals of Statistics*, 26:1651–1686, 1998.
- [Sugeno, 1974] M. Sugeno. *Theory of Fuzzy Integrals and its Application*. PhD thesis, Tokyo Inst. of Techn., 1974.
- [Wettschereck *et al.*, 1997] D. Wettschereck, D.W. Aha, and T. Mohri. A review and empirical comparison of feature weighting methods for a class of lazy learning algorithms. *AI Review*, 11:273–314, 1997.
- [Zhang *et al.*, 1997] J. Zhang, Y. Yim, and J. Yang. Intelligent selection of instances for prediction in lazy learning algorithms. *Art. Intell. Review*, 11:175–191, 1997.

Automating the Discovery of Recommendation Knowledge

David McSherry and Christopher Stretch

School of Computing and Information Engineering
University of Ulster, Coleraine BT52 1SA, Northern Ireland
{dmg.mcsherry, ct.stretch}@ulster.ac.uk

Abstract

In case-based reasoning (CBR) systems for product recommendation, the retrieval of acceptable products based on limited information is an important and challenging problem. As we show in this paper, basic retrieval strategies such as nearest neighbor are potentially unreliable when applied to incomplete queries. To address this issue, we present techniques for automating the discovery of recommendation rules that are provably reliable and non-conflicting while requiring minimal information for their application in a rule-based approach to the retrieval of recommended cases.

1 Introduction

In CBR recommender systems, descriptions of the available products are stored as *cases* in a case library, and retrieved in response to a query representing the user's known requirements. In approaches related to conversational CBR (CCBR) [Aha *et al.*, 2001], a query is incrementally elicited in a *dialogue* with the user, often with the aim of minimizing the number of questions the user is asked before an acceptable product is retrieved [e.g., Doyle and Cunningham, 2000; Kohlmaier *et al.*, 2001; McSherry, 2003].

Increasing the efficiency of recommendation dialogues is also a major focus of research interest in *critiquing* approaches to navigation of complex product spaces [e.g., Burke, 2002; Reilly *et al.*, 2004]. However, an aspect of product recommendation that appears to have received little research attention is the ability of experienced salespersons to make reliable recommendations based on *minimal* information and without engaging in a recommendation dialogue with the customer.

For example, an estate agent may recommend property X to a customer she knows to be interested in a 3-bedroom property in location A without asking about other requirements the customer may have. To be confident about recommending property X, the estate agent must take account of *all* features that may affect its acceptability as

well as the relative merits of other available properties. Her recommendation is essentially a prediction that property X is likely to be the *most* acceptable of the available properties regardless of the customer's unknown preferences with respect to attributes other than location and bedrooms.

Providing CBR recommender systems with a comparable ability to make reliable recommendations based on minimal information is the goal that motivates the work presented in this paper. As we show in Sections 3 and 4, basic retrieval strategies such as nearest neighbor are potentially unreliable when applied to incomplete queries because of their failure to take account of all features of a recommended case. To address this issue, we present techniques for automating the discovery of recommendation rules that are provably reliable and non-conflicting while requiring minimal information for their application in a rule-based approach to the retrieval of recommended cases.

In Sections 2 and 3, we present a prototype system for rule-based retrieval of recommended cases and techniques for automating the discovery of *identification* rules that uniquely identify a case from its partial description. In Section 4, we present techniques for the discovery of more reliable recommendation rules that we refer to as *dominance* rules. In Section 5, a theoretical upper bound for the size of the discovered rule sets in our approach is established and confirmed by empirical results based on publicly available datasets. Related work is discussed in Section 6 and our conclusions are presented in Section 7.

2 Recommendation Rules

Given a case library representing a collection of available products, our aim is to automate the discovery of recommendation rules to support rule-based retrieval of recommended products. The rules targeted by our discovery algorithms are of the form **if** Q **then** C , or $Q \rightarrow C$, where Q is a simple query in the form of a list of required features and C is the case that will be retrieved in response to Q . Table 1 shows a small case library in the property domain that we use to illustrate the discussion. Attributes in the example case library are location (loc), style, bedrooms (beds), and reception rooms (RRs).

Table 1. An example case library in the property domain.

Case No.	Loc	Style	Beds	RRs
1	C	semi	5	two
2	C	terraced	4	three
3	B	semi	4	three
4	A	detached	4	two
5	B	terraced	2	three
6	A	detached	3	two
7	C	semi	2	three
8	C	semi	3	two
9	B	detached	3	three

Our discovery algorithms are based on recommendation criteria that give rise to recommendation rules of different types. To ensure that the discovered rules are *non-conflicting*, we insist that for $Q \rightarrow C$ to be a recommendation rule, C must be *strictly* better than any other case according to the underlying recommendation criterion. The recommendation criterion on which we focus in Section 3 is that the recommended case is the *only* case that exactly matches the user’s known requirements. A similar or weaker criterion is used in some CBR approaches [e.g., Doyle and Cunningham, 2000; McSherry, 2001]. We will refer to recommendation rules based on exact matching as *identification* rules.

Given a query Q over a subset A_Q of the case attributes A , we refer to $|A_Q|$ as the *length* of the query, and define $\text{exact-matches}(Q) = \{C : \pi_a(C) = \pi_a(Q) \text{ for all } a \in A_Q\}$, where for each $a \in A_Q$, $\pi_a(C)$ is the value of a in C and $\pi_a(Q)$ is the value of a in Q . We say that a given query Q is a *sub-query* of another query Q^* , or that Q^* is an *extension* of Q , if $A_Q \subseteq A_{Q^*}$ and $\pi_a(Q) = \pi_a(Q^*)$ for all $a \in A_Q$. We denote the relationship by writing $Q \subseteq Q^*$. For any case C , we refer to the query Q^C such that $\pi_a(Q) = \pi_a(C)$ for all $a \in A$ as the *characteristic* query for C . Clearly for any case C and query Q , $C \in \text{exact-matches}(Q)$ if and only if $Q \subseteq Q^C$.

Definition 1. For any case C and query Q , we say that $Q \rightarrow C$ is an *identification rule* if $\text{exact-matches}(Q) = \{C\}$.

The length of a recommendation rule is the length of the query on its left-hand side (LHS). A recommendation rule $Q \rightarrow C$ is *maximally general* (MG) if there is no proper sub-query Q° of Q such that $Q^\circ \rightarrow C$ is also a recommendation rule (of the same type). To ensure that the discovered rules require *minimal* information for their application, we focus on the discovery of MG recommendation rules. As we show in Section 3, the only MG identification rules for Case 6 in our example case library are:

Rule 1. **if** loc = A **and** beds = 3 **then** Case 6

Rule 2. **if** style = det **and** beds = 3 **and** RRs = two **then** Case 6

By focusing on MG rules, we also aim to maximize *coverage* of the product space provided by the discovered rules. However, an important point to note is that an

identification rule cannot simply be applied to any query that includes the conditions on its LHS without risking violation of the exact-matching criterion. For example, Rule 1 does not apply for a user looking for a 3-bedroom property in location A if it is also *known* that she prefers a terraced property. However, if $Q \rightarrow C$ is an identification rule and Q^* is any query such that $Q \subseteq Q^* \subseteq Q^C$, then it is clear that $Q^* \rightarrow C$ is also an identification rule.

In *Rubric*, our prototype system for rule-based retrieval of recommended cases, a recommendation rule is applied only to queries that it *covers* in the following sense.

Definition 2. A recommendation rule $Q \rightarrow C$ covers a given query Q^* if $Q \subseteq Q^*$ and $Q^* \rightarrow C$ is also a recommendation rule.

For example, an identification rule $Q \rightarrow C$ covers any query Q^* such that $Q \subseteq Q^* \subseteq Q^C$. Given a set of recommendation rules, and a query representing a user’s known requirements, *Rubric* checks through the rules and retrieves the case recommended by the first rule that covers the target query. If none of the available rules covers the target query, *Rubric* simply *abstains* from making a recommendation.

Our rule-based approach to retrieval is related to CBR approaches in which a decision tree is used to guide the retrieval of recommended cases [e.g., McSherry, 2001]. In such a decision tree, each path to a leaf node at which a single case is recommended is an identification rule. However, rule-based retrieval has the potential to provide greater coverage, as a decision tree constructed by standard partitioning methods can have at most one rule for each case, and some of the rules may not be MG.

3 Identification Rule Discovery

Our algorithm for the discovery of MG identification rules, *MGIRules*, is shown in Figure 1. *SubQueries* is a list of all sub-queries, in order of increasing length, of the characteristic query Q^C for a target case C . Each such sub-query is a candidate to appear on the LHS of a discovered identification rule. For any sub-query Q_1 such that $\text{exact-matches}(Q_1) = \{C\}$, *MGIRules* adds $Q_1 \rightarrow C$ to the list of discovered rules and eliminates all sub-queries Q_2 of which Q_1 is a sub-query from the remaining list of candidate sub-queries.

To illustrate the approach, Figure 2 shows all sub-queries of the characteristic query (A, det, 3, two) for Case 6 in our example case library. The first identification rule to be discovered is Rule 1 (A, 3). Following the elimination of the underlined sub-queries in Figure 2, the only other identification rule for Case 6 is Rule 2 (det, 3, two).

With each case in turn as the target case, *MGIRules* can be used to discover all MG identification rules in a given case library. The worst-case complexity of applying *MGIRules* to all n cases in a case library with k attributes is $O(k \times n^2 \times 2^k)$ if $n \geq 2^k$. If $n < 2^k$, the worst-case complexity is $O(k \times n \times 2^{2k})$.

```

algorithm MGIRules( $C, SubQueries$ )
begin
   $Rules \leftarrow \emptyset$ 
  while  $|SubQueries| > 0$  do
    begin
       $Q_1 \leftarrow first(SubQueries)$ 
       $Deletions \leftarrow \{Q_1\}$ 
      if  $exact\_matches(Q_1) = \{C\}$ 
      then begin
         $Rules \leftarrow Rules \cup \{Q_1 \rightarrow C\}$ 
        for all  $Q_2 \in rest(SubQueries)$  do
          begin
            if  $Q_1 \subseteq Q_2$ 
            then  $Deletions \leftarrow Deletions \cup \{Q_2\}$ 
          end
        end
      end
       $SubQueries \leftarrow SubQueries - Deletions$ 
    end
  return  $Rules$ 
end

```

Figure 1. Algorithm for the discovery of all MG identification rules for a target case.

It is worth noting that an identification rule $Q \rightarrow C$ is excluded by MGIRules only if it has already discovered an MG identification rule $Q^\circ \rightarrow C$ such that $Q^\circ \subseteq Q$. As any query covered by $Q \rightarrow C$ is also covered by $Q^\circ \rightarrow C$, the exclusion of $Q \rightarrow C$ causes no loss of coverage in *Rubric*.

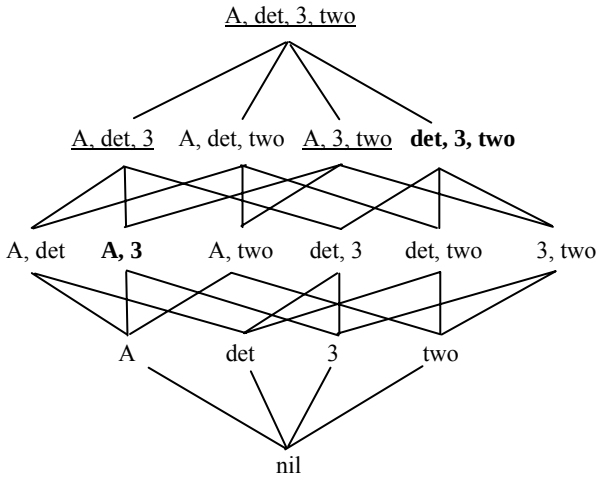


Figure 2. Identification rule discovery for Case 6 in the example case library.

Identification rules have the important advantage that recommendations can be *justified* on the basis that the recommended case exactly matches the known requirements of the user. However, a limitation they share with decision trees is their failure to take account of *all* features of a

recommended case that may affect its acceptability. The extent to which this affects their reliability is likely to depend on the importance of the user's *unknown* requirements.

For example, according to Rule 2 in Section 2, Case 6 can be recommended to a user who is known to be looking for a detached property with 3 bedrooms and two reception rooms. But given the importance of location in the property domain, Case 6 is unlikely to be the most acceptable case if the user happens to prefer location B. In fact, Case 9 now looks a better alternative in light of the user's preference for location B. In Section 4, we present techniques for the discovery of recommendation rules that do take account of all features of a recommended case, including those with respect to which the user's preferences are unknown.

4 Dominance Rules

As nearest-neighbor (NN) retrieval is a common approach to product recommendation in CBR, it is natural to consider NN rules as an alternative to identification rules. Given a query Q over a subset A_Q of the case attributes A , the similarity of any case C to Q is typically defined to be:

$$Sim(C, Q) = \sum_{a \in A_Q} w_a sim_a(C, Q)$$

where for each $a \in A$, w_a is an importance weight assigned to a , and $sim_a(C, Q)$ is a *local* measure of the similarity of $\pi_a(C)$, the value of a in C , to $\pi_a(Q)$, the value of a in Q . When discussing actual similarity scores, we will divide $Sim(C, Q)$ by the sum of all the importance weights to give a *normalized* similarity score. As usual in practice, we assume that for all $a \in A$, $0 \leq sim_a(x, y) \leq 1$ and $sim_a(x, y) = 1$ if and only if $x = y$. We also assume that for all $a \in A$, the distance measure $1 - sim_a$ satisfies the triangle inequality. For any query Q , we define $most_similar(Q) = \{C : Sim(C, Q) \geq Sim(C^\circ, Q) \text{ for all cases } C^\circ\}$.

Definition 3. For any case C and query Q , we say that $Q \rightarrow C$ is an NN rule if $most_similar(Q) = \{C\}$.

As in the case of an identification rule, an NN rule cannot simply be applied to any query that includes the conditions on its LHS. If $Q \rightarrow C$ is an NN rule and Q^* is an extension of Q then there is no guarantee that $most_similar(Q^*) = \{C\}$. It is worth noting, though, that $Q^* \rightarrow C$ is an NN rule if $Q^* - Q \subseteq Q^C$. Clearly, any identification rule is also an NN rule.

As we shall see, however, NN rules offer no obvious improvement over identification rules in terms of their reliability when applied to incomplete queries. Again we use the example case library in Table 1 to illustrate the discussion. The importance weights we assign to loc, style, beds, and RRs are 4, 3, 2, and 1. We define the similarity of two values x and y of a numeric attribute a to be $1 - \frac{|x - y|}{\max(a) - \min(a)}$ where, for example, $\max(a)$ is the

maximum value of a in the case library. Our similarity measure for style is equivalent to applying our similarity measure for numeric attributes to the corresponding number of adjoining properties (det = 0, semi = 1, ter = 2). Finally, our similarity measures for location and reception rooms assign a similarity score of 1 if the two values are the same and 0 if they are not the same.

Returning to our example in Section 3 of a user looking for a detached property with 3 bedrooms and two reception rooms, we can now use Rule 2 as an NN rule to retrieve Case 6 as the recommended case. It can easily be checked that the cases that are most similar to the user's known requirements are Case 6 (0.60) and Case 4 (0.53). But if the user happens to prefer location B, the most similar case in light of this *unknown* preference would be Case 9 (0.90). In fact, Case 6 is likely to be the most acceptable case only if the user happens to prefer location A, as a preference for location C would see Case 8 (0.85) emerging as the most similar case.

As this example illustrates, the reliability of NN rules (and NN retrieval) is open to question when applied to incomplete queries. However, one example of a reliable NN rule is:

Rule 1. **if** loc = A **and** beds = 3 **then** Case 6

It can be seen that the similarity of Case 6 to any query that includes the conditions on the LHS cannot be equaled by any other case regardless of the user's preferences with respect to style or reception rooms. For example, Case 4 reaches its maximum similarity of 0.93 if the user happens to prefer a detached property with two reception rooms, but these additional preferences also increase the similarity of Case 6 from 0.60 to 1.00.

Rules 1 and 2 are positive and negative examples of the type of recommendation rule we refer to as *dominance* rules.

Definition 4. For any case C and query Q , we say that $Q \rightarrow C$ is a dominance rule if $\text{most-similar}(Q^*) = \{C\}$ for all extensions Q^* of Q .

As well as being more reliable, dominance rules provide more coverage than identification or NN rules. It can easily be seen that if $Q \rightarrow C$ is a dominance rule, then $Q^* \rightarrow C$ is also a dominance rule for any query Q such that $Q \subseteq Q^*$. That is, a dominance rule covers *any* query that includes the conditions on its LHS. An important role in our approach to the discovery of dominance rules is played by the concept of case dominance proposed by McSherry [2003] as a basis for recognizing when recommendation dialogues can be terminated without loss of solution quality.

Definition 5. A given case C_1 dominates another case C_2 with respect to a query Q if $\text{Sim}(C_1, Q^*) > \text{Sim}(C_2, Q^*)$ for all extensions Q^* of Q .

It can be seen that $Q \rightarrow C$ is a dominance rule if and only if C dominates all other cases with respect to Q . McSherry [2003] uses the triangle inequality to show that a given case

C_1 dominates another case C_2 with respect to a query Q if and only if:

$$\text{Sim}(C_1, Q) - \text{Sim}(C_2, Q) > \sum_{a \in A-Q} w_a(1 - \text{sim}_a(C_1, C_2))$$

We focus on the discovery of MG dominance rules $Q \rightarrow C$ such that $Q \subseteq Q^C$, where Q^C is the characteristic query for C . As well as reducing the complexity of the discovery process, this ensures that recommendations based on the discovered rules can be justified on the grounds that the recommended case exactly matches *some* of the user's known requirements, and that there is no other case that exactly matches those requirements. It can be seen from the following theorem that our exclusion of dominance rules that do not have this property cannot result in failure to discover dominance rules of the shortest possible length for a given target case.

Theorem 1. For any dominance rule $Q \rightarrow C$, there exists a dominance rule $Q' \rightarrow C$ of equal length such that $Q' \subseteq Q^C$.

Proof. Let $Q \rightarrow C_1$ be a dominance rule and let Q' be the query such that $\pi_a(Q') = \pi_a(C_1)$ for all $a \in A_Q$. To establish that $Q' \rightarrow C_1$ is also a dominance rule, it suffices to show that C_1 dominates any other case C_2 with respect to Q' . For any $a \in A_Q$, we know from the triangle inequality that: $1 - \text{sim}_a(C_2, Q) \leq 1 - \text{sim}_a(C_2, Q') + 1 - \text{sim}_a(Q', Q) = 2 - \text{sim}_a(C_2, Q') - \text{sim}_a(C_1, Q)$. So $\text{sim}_a(C_1, Q') - \text{sim}_a(C_2, Q') = 1 - \text{sim}_a(C_2, Q') \geq \text{sim}_a(C_1, Q) - \text{sim}_a(C_2, Q)$. As C_1 dominates C_2 with respect to Q and $A_Q = A_{Q'}$, it can now be seen that: $\text{Sim}(C_1, Q') - \text{Sim}(C_2, Q') \geq \text{Sim}(C_1, Q) - \text{Sim}(C_2, Q) > \sum_{a \in A-A_{Q'}} w_a(1 - \text{sim}_a(C_1, C_2))$. It follows as required that C_1

dominates C_2 with respect to Q' . $\square$

To convert MGIRules to a new algorithm called *MGDRules* for the discovery of all MG dominance rules $Q \rightarrow C$ for a target case C such that $Q \subseteq Q^C$, it is necessary only to replace the condition underlined in Figure 1 by the condition:

C dominates all other cases with respect to Q_1

As in the case of MGIRules, the worst-case complexity of applying MGDRules to all n cases in a product case library with k attributes is $O(k \times n^2 \times 2^k)$ or $O(k \times n \times 2^{2k})$ depending on whether $n \geq 2^k$.

5 Discovered Rules

As might be expected, our algorithms discovered fewer dominance rules (12) than identification rules (24) in the example case library. However, the 12 dominance rules cover 34% of all possible queries in the example product space compared to 28% for the 24 identification rules, and 17% for a set of 9 decision-tree rules of the shortest possible length for each case. In this section, we examine the behavior of our algorithms when applied to case libraries of more realistic size. As the coverage gains provided by dominance rules are measurable only in finite product

spaces, our analysis focuses on the number and length of the discovered rules.

It can be seen that for any case C , $Q^C \rightarrow C$ is both an identification rule and a dominance rule provided no other case has the same value as C for every attribute. Two cases with identical descriptions in a product case library are said to be *inseparable* [McSherry, 2002]. At least one rule of each type must therefore be discovered for a target case provided there is no case from which it is inseparable.

We now establish an upper bound for the number of rules that can be discovered by *MGIRules* or *MGDRules* for a given target case.

Theorem 2. For any case C , the number of MG recommendation rules $Q \rightarrow C$ such that $Q \subseteq Q^C$ can never be more than $^k C_{\lfloor k/2 \rfloor}$, where k is the number of attributes in the case library and $\lfloor k/2 \rfloor$ is the integer part of $k/2$.

Proof. First we note that if $Q_1 \rightarrow C$ and $Q_2 \rightarrow C$ are distinct MG recommendation rules such that $Q_1, Q_2 \subseteq Q^C$, then A_{Q_1} and A_{Q_2} are incomparable subsets of A . For example, if $A_{Q_1} \subseteq A_{Q_2}$ then contrary to our assumption $Q_2 \rightarrow C$ cannot be MG. The result immediately follows from Sperner's [1928] proof that the maximum number of incomparable subsets of any set of size m is $^m C_{\lfloor m/2 \rfloor}$. $\square$

For a case library with 8 attributes, the maximum number of rules that can be discovered by our algorithms for a given target case is $^8 C_4 = 70$. Table 2 shows the corresponding limits for attribute numbers in the range from 4 to 10. However, we now present empirical evidence which suggests that the discovered rule sets tend to be much smaller in practice than their maximum possible sizes.

Table 2. Maximum rule-set size for a single case.

No. of Attributes:	4	5	6	7	8	9	10
Maximum Size:	6	10	20	35	70	126	252

Both of our experimental case libraries have 8 attributes, and include continuous as well as nominal attributes. Based on the *AutoMPG* dataset from the UCI Repository, our first case library contains the descriptions of 392 automobiles in terms of attributes one might expect to see in a recommender system for previously-owned automobiles (e.g., year, origin, mpg). Our second case library is *Travel* (www.ai-cbr.org), a standard benchmark containing the descriptions of over 1,000 holidays in terms of attributes such as price, destination, and transport.

Figure 3 shows the numbers of rules discovered by our algorithms over all (complete) cases in *AutoMPG* and *Travel*, apart from two inseparable cases in *Travel* for which no rules were discovered. The largest rule-set size for any case (20) is considerably smaller than the maximum rule-set

size for 8 attributes (70). On the whole, the results for *AutoMPG* and *Travel* are remarkably similar, with fewer dominance rules than identification rules discovered in both case libraries. The average rule-set size of 4 for dominance rules in *Travel* is based on a total of 4,127 discovered rules.

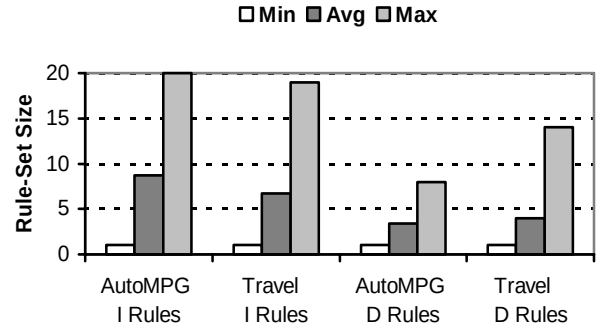


Figure 3. Numbers of discovered identification (I) and dominance (D) rules for each case.

Table 3 shows the *lengths* of the identification and dominance rules discovered by our algorithms in *AutoMPG* and *Travel*. As might be expected, the discovered dominance rules are longer on average than the identification rules. In both case libraries, though, the discovered dominance rules provide clear benefits in terms of reducing the number of attributes whose preferred values must be known for a *reliable* recommendation to be made. These benefits are particularly evident in *Travel*, with reductions in query length of up to 63%, and 38% on average, relative to queries involving all eight attributes.

Table 3. Discovered rule lengths in *AutoMPG* and *Travel*.

		Min	Avg	Max
Identification Rules:	AutoMPG	1	2.0	4
	Travel	1	2.9	6
Dominance Rules:	AutoMPG	4	5.6	7
	Travel	3	5.0	7

6 Related Work

Recent work by McSherry [2004a] provides a different perspective on recommendation rule discovery in which the discovered rules (one for each case) are used to describe the *behavior* of an existing recommender system in localized areas of the product space. For example, the discovered rules can be used to identify conditions in which a given product will be recommended by the system, or regions of the product space that are sparsely represented. However, the discovered rules may not be MG and there is no discussion of their possible use for rule-based retrieval.

Burke and Kass [1996] propose a rule-based approach to retrieval in a system for case-based teaching of advertising sales techniques. In *Spiel*, stories relating lessons learned by

experienced salespersons are retrieved in response to a student's actions in a simulated sales environment. Retrieval in Spiel is *opportunistic*, *conservative* and *non-mandatory*; that is, stories are retrieved at the system's initiative and only if highly relevant to the student's current situation.

By design, our rule-based approach to retrieval is also *conservative* and *non-mandatory*, though it can easily be combined with NN retrieval of a less strongly recommended case if none of the available recommendation rules covers the user's known requirements. We also propose to investigate its potential role as an *opportunistic* and *complementary* retrieval strategy in recommender systems based on CCBR or critiquing approaches.

Potential benefits include enabling recommender systems to recognize when recommendation dialogues can be safely discontinued without affecting solution quality. Often in critiquing, for example, an initially recommended case is retrieved in response to an initial query entered by the user [e.g., Burke, 2002; Reilly *et al.*, 2004]. In this situation, the existence of a dominance rule that covers the user's initial query may be a good indication that a more acceptable case is unlikely to be found no matter how the user chooses to critique the initially recommended case.

Given the importance of *explanation* in recommender systems [e.g., Herlocker *et al.*, 2000; McSherry, 2004b], simplifying explanations of why a given product is recommended is another potential benefit in approaches that aim, but cannot guarantee, to minimize the length of recommendation dialogues [e.g., Doyle and Cunningham, 2001; Kohlmaier *et al.*, 2001; McSherry, 2003].

7 Conclusions

Aiming to improve the reliability of recommendations based on incomplete queries in CBR recommender systems, we have investigated two possible approaches to the discovery of recommendation knowledge to support a *rule-based* approach to the retrieval of recommended cases. While having the potential to provide greater coverage than decision-tree approaches and enabling recommendations to be easily justified, the *identification* rules discovered by our first discovery algorithm offer no obvious improvement in terms of their reliability when applied to incomplete queries. On the other hand, retrieval based on *dominance* rules is provably more reliable than decision-tree approaches and NN retrieval when applied to incomplete queries in that no competing case can equal the similarity of a recommended case regardless of the user's unknown preferences.

References

- [Aha *et al.*, 2001] David Aha, Leonard Breslow, and Héctor Muñoz-Avila. Conversational case-based reasoning. *Applied Intelligence*, 14:9-32, 2001.
- [Burke, 2002] Robin Burke. Interactive critiquing for catalog navigation in e-commerce. *Artificial Intelligence Review*, 18:245-267, 2002.
- [Burke and Kass, 1996] Robin Burke and Alex Kass. Retrieving stories for case-based teaching. In David Leake, editor. *Case-Based Reasoning: Experiences, Lessons & Future Directions*, pages 93-109. AAAI Press, Cambridge, Massachusetts, 1996.
- [Doyle and Cunningham, 2000] Michelle Doyle and Pádraig Cunningham. A dynamic approach to reducing dialog in on-line decision guides. In *Proceedings of the Fifth European Workshop on Case-Based Reasoning*, pages 49-60, Trento, Italy, 2000. Springer-Verlag.
- [Herlocker *et al.*, 2000] Jonathan Herlocker, Joseph Konstan, and John Riedl. Explaining collaborative filtering recommendations. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work*, pages 241-250, Philadelphia, Pennsylvania, USA, 2000.
- [Kohlmaier *et al.*, 2001] Andreas Kohlmaier, Sasha Schmitt, and Ralph Bergmann. A similarity-based approach to attribute selection in user-adaptive sales dialogues. In *Proceedings of the Fourth International Conference on Case-Based Reasoning*, pages 306-320, Vancouver, Canada, 2001. Springer-Verlag.
- [McSherry, 2001] David McSherry. Minimizing dialog length in interactive case-based reasoning. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, pages 993-998, Seattle, Washington, 2001. Morgan Kaufmann.
- [McSherry, 2002] David McSherry. The inseparability problem in interactive case-based reasoning. *Knowledge-Based Systems*, 5:293-300, 2002.
- [McSherry, 2003] David McSherry. Increasing dialogue efficiency in case-based reasoning without loss of solution quality. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, pages 121-126, Acapulco, Mexico, 2003.
- [McSherry, 2004a] David McSherry. Automating the discovery of recommendation rules. In *Proceedings of AI-2004, the Twenty-fourth BCS SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence*, pages 87-100, Cambridge, UK, 2004. Springer-Verlag.
- [McSherry, 2004b] David McSherry. Explanation in recommender systems. In *Proceedings of the ECCBR 2004 Workshops*, pages 125-134, Madrid, Spain. Universidad Complutense Madrid.
- [Reilly *et al.*, 2004] James Reilly, Kevin McCarthy, Lorraine McGinty, and Barry Smyth. Dynamic critiquing. In *Proceedings of the Seventh European Conference on Case-Based Reasoning*, pages 763-777, Madrid, Spain, 2004. Springer-Verlag.
- [Sperner, 1928] Emanuel Sperner. Ein Satz über Untermengen einer endlichen Menge. *Mathematische Zeitschrift*, 27: 544-548, 1928.

Sophia: A novel approach for Textual Case-based Reasoning

David Patterson¹, Niall Rooney¹, Vladimir Dobrynin², Mykola Galushka¹

¹Northern Ireland Knowledge Engineering Laboratory
University of Ulster, Jordanstown, BT37OQB, U.K.
{wd.patterson, nf.rooney, mg.galushka}@ulster.ac.uk

²St Petersburg State University
198904 Petrodvoretz, St Petersburg, Russia
vdobr@oasis.apmath.spbu.ru

Abstract

In this paper we present a novel methodology for textual case-based reasoning. This technique is unique in that it automatically discovers case and similarity knowledge, is language independent, is scaleable and facilitates semantic similarity between cases to be carried out inherently without the need for domain knowledge. In addition it provides an insight into the thematic content of the case-base as a whole, which enables users to better structure queries. We present an analysis of the competency of the system by assessing the quality of the similarity knowledge discovered and show how it is ideally suited to case-based retrieval (querying by example).

1 Introduction

Textual Case-based reasoning (CBR) is very different in many respects from more conventional CBR applications where case knowledge is usually more easily acquired, structured and adequately (and often simplistically) represented as simple vectors or objects. In textual CBR (TCBR) the knowledge to be embodied within a case is much more intricate in that it contains complex linguistic terms and concepts on various topics which are often encompassed within the same case. Acquiring and representing this knowledge adequately, without losing its meaning and with a low knowledge engineering overhead to users, remains a challenging prospect. Equally challenging is the prospect of discovering, implementing and maintaining useful similarity knowledge within such systems where it is often vital to identify similar cases that perhaps do not necessarily contain the same words but semantically similar themes or concepts. It may initially seem that the problem of TCBR could be adequately addressed by standard Information retrieval (IR) techniques. However there are limitations with these approaches. Perhaps the best known IR approach, commonly known as the bag of words, transforms docu-

ments into a vector space model [Sebastiani, 2002], whereby the case knowledge representation formalism is a vector of weights representing the individual words present in each document and similarity is based on a simplistic comparison of overlap among case vectors. However, as is clear, word order and negation information is lost in the transformation, which may be vital for reasoning in certain domains of interest. Additionally, word sense disambiguation is lost in the process e.g. the word “Java” can have different meanings depending on its context of usage. Most importantly of all, there is a loss in transparency to users, which is a major drawback during case retrieval. NLP techniques can determine word order but they can be brittle and computationally expensive. Some researchers in the CBR community have followed this approach and developed novel but often domain specific approaches to the task of TCBR. Examples of such systems include work by Brün- inghaus & Ashley, [2001], Cunningham et al., [2004], Kunze & Hubner [1998] and Lenz [1998]. These systems are very domain-dependent, require considerable knowledge engineering effort and are usually designed for the situation where all documents have a similarly structured content.

Other researchers have instead focused on advancing IR techniques using technologies such as Latent Semantic Indexing [Deerwester et al., 1990] and Probabilistic Latent Semantic Indexing [Hoffman, 1999]. These go beyond the bag-of-words model providing a more advanced view of the document space that may require less knowledge engineering activity within a TCBR context [Zelikovitz and Hirsh, 2002]. Unfortunately these have crucial drawbacks also. For example they use a transformation to project documents into cases and as such they become non transparent to a user. Transparency is an important component of CBR systems to enable the user gain insight into the reasoning and explanation processes of the system. As such they provide little insight into the knowledge contained within a case or how similarity is determined during retrieval. A further limitation is that these approaches are essentially dimensionality reduction techniques and as such important information may

be lost. Additionally word order and negation knowledge is also lost. Wiratunga et al., [2004], combine IR and Machine Learning techniques in their approach. A strength is its ability to facilitate automated semantic similarity determination but unfortunately their approach is completely supervised and as such the class of each document within the collection must be known apriori to enable the discovery of similarity knowledge. This limits the application of the technology. In addition the effectiveness of the technique to multi class domains has not been addressed and remains an open question. Cunningham et al., [2004], have also recently proposed a very different approach to TCBR based on graph theory, that maintains the original document structure and word order. A disadvantage of their approach is that they require an expert to identify domain dependant indexes, which they rely on to assess case similarity using graph distance techniques. Therefore there is a significant knowledge engineering burden in terms of similarity knowledge. In addition it cannot determine semantic similarity between cases or cope with ambiguities that may occur. One final open question with this technique is its efficiency with medium to large case-bases as assessing sub graph similarity is a complex matter.

In this research, we present a new approach for TCBR called SOPHIA CBR, based upon a scaleable contextual document clustering approach [Dobrynin et al., 2004], which facilitates an advanced and rich knowledge discovery framework for case-based retrieval. SOPHIA's case representation formalism is similar to a classical vector space model except, rather than using word frequencies, it is based on the conditional probability distributions of terms within documents. It then intelligently discovers important contexts within the case-base and organizes cases into one of a large number of clusters which have these contexts as attractors. This produces groups of semantically related cases (i.e. cases which are on the same or similar subjects but use different terminology, can be recognized as similar) for a given cluster context. This process of forming clusters, allows both a very efficient and competent case retrieval process. It is this unique feature that provides much of the power behind the technology.

SOPHIA CBR is advantageous in that, it is domain independent, and has low knowledge engineering overheads as it does not require any user intervention to acquire domain knowledge. As such, all knowledge can be discovered automatically (although if background knowledge is already present it can be utilized). Additionally it uses a transparent case knowledge representation, automatically discovers and provides users with additional knowledge about the domain that they can use to refine queries, is language independent, is scaleable and can differentiate between the different contexts of potentially ambiguous terms. The novel technology presented in SOPHIA CBR is useful for both classification tasks and for retrieval, browsing and searching by example. SOPHIA does not have a mechanism to identify word order or negation, features which undoubtedly are important for

document collections where each document has a similar internal structure. However in terms of its application to document collections such as presented in this paper (where each document does not necessarily have a similar internal content structure), we show that SOPHIA is capable of providing a competent TCBR system. Firstly we present the SOPHIA technology, then we carry out an initial experiment to demonstrate the quality of the discovered case and similarity knowledge. This is followed by an additional experiment which investigates the potential of the system to case-based retrieval (query by example). Finally we discuss the results and present future work.

2 SOPHIA CBR Methodology

Knowledge is automatically discovered at various stages within the SOPHIA Framework. In step 1 we describe how case knowledge is discovered and represented. In step 2 we show how numerous specific narrow context words are automatically identified. These narrow contexts act as attractors for clustering cases and this step can be regarded as *global* similarity knowledge discovery. In step 3 *cluster level* similarity knowledge is discovered and used to determine which narrow context each case should be assigned to. Step 4, is strictly not part of the clustering algorithm itself but is an additional processing step that provides extra knowledge about the internal case structure of clusters and provide a means for visualizing them. This can be regarded as *localized* similarity knowledge discovery. As will become apparent, not only does this empower the user with extra domain knowledge about the problem area but it improves both the case index and the ability of the system to identify similar cases.

Step 1 Case knowledge Discovery. Here we describe how case knowledge is automatically extracted from a document corpus. In the following definitions, a term refers to a word in the document corpus. Let Ξ denote the set of all documents in the document corpus and Ψ denote the set of all terms present in Ξ . For every document in the corpus a case is automatically extracted and represented by a probability distribution over all terms occurring in that document.

$$p(y|x) = \frac{tf(x,y)}{\sum_{t \in \Psi} tf(x,t)}$$

where $tf(x,y)$ is the term frequency of the term y in document x and t is a term from the set of all terms present in the document collection. Although by itself this does not provide a richer case representation than using conventional IR approaches, it does facilitate the process of grouping, indexing and retrieving semantically similar cases, which forms the centerpiece of the power of this technology.

Step 2 Global Similarity Knowledge Discovery. Given a term $z \in \Psi$, we define its context as the probability distribution of a set of words which co-occur with the given term. More specifically the context of the term z is represented in the form of a conditional probability distribution $p(Y|z)$, where the random variable Y takes values from Ψ and $p(y|z)$ is equal to the probability of randomly selecting the term y in a randomly selected case within which the term z co-occurs. We can approximate this distribution as:

$$p(y|z) = \frac{\sum_{x \in \Xi(z)} tf(x, y)}{\sum_{x \in \Xi(z), t \in \Psi} tf(x, t)}$$

where $tf(x, y)$ is the term frequency of the term y in case x and $\Xi(z)$ is the set of all cases from the corpus which contain the term z . It is obvious that in most cases the context of the term z is too general in scope to present useful information about the corpus. So we are interested only in identifying narrow context terms z . The narrowness of the term z is estimated by the entropy of the probability distribution $p(Y|z)$:

$$H(Y|z) = -\sum_y p(y|z) \log(p(y|z))$$

Let $\Psi(z)$ denote the set of all different terms from cases in $\Xi(z)$. When there is a uniform distribution of terms from $\Psi(z)$ the entropy $H(Y|z)$ is equal to $\log |\Psi(z)|$. According to Heaps Law $\log |\Psi(z)| = O(\log(|\Xi(z)|))$ [Baeza-Yates & Ribeiro-Neto, 1999] there is a relationship between the case frequency $df(z) = |\Xi(z)|$ of the term z and the entropy of its context. To allow for this dependency, we divide the whole set of words into r disjoint subsets:

$$\Psi = \bigcup_i \Psi_i$$

$$\Psi_i = \{z : z \in \Psi, df_i \leq df(z) < df_{i+1}\}$$

$$i = 1..r$$

Here the threshold df_i satisfies the condition $df_{i+1} = \alpha df_i$ where $\alpha > 1$ is a constant. Choosing narrow word contexts are based on the assumption that in total there are N narrow word contexts. For every $i = 1..r$ a set $Z_i \subset \Psi_i$, is selected such that

$$|Z_i| = \frac{N \cdot |\Psi_i|}{\sum_{j=1..r} |\Psi_j|}$$

and $z_1 \in Z_i, z_2 \in \Psi_i - Z_i \rightarrow H(Y|z_1) \leq H(Y|z_2)$. Then

$Z = \bigcup_i Z_i$, where Z is the set of selected narrow contexts. These contexts form the seeds for clustering semantically

related cases, where cluster membership (similarity) is measured using the Jensen-Shannon (JS) divergence [Lin, 1991]. In this respect contexts are regarded as global similarity knowledge.

Step 3 Cluster Level Similarity Knowledge Discovery. Narrow contexts $\{p(Y|z)\}_{z \in Z}$, discovered in step 2, are considered as cluster attractors. Within this study all cases are grouped into at most one cluster based on the case similarity to the attractor, i.e. this is a hard clustering approach but equally a softer approach could be applied. In this way cases are associated with the contexts they most closely match to form clusters of cases that are on similar (closely related) subjects or themes. The similarity between a case x and the context for the term z is estimated by the JS divergence between the probability distributions p_1 and p_2 representing the case and the context respectively:

$$JS_{\{0.5, 0.5\}}[p_1, p_2] = H[\bar{p}] - 0.5H[p_1] - 0.5H[p_2]$$

where $H[p]$ denotes the entropy of the probability distribution p and $\bar{p}$ denote the average probability distribution $= 0.5p_1 + 0.5p_2$. A case x is therefore assigned to a cluster with attractor z if,

$$z = \arg \min_{t \in Z} JS_{\{0.5, 0.5\}}[p(Y|x), p(Y|t)]$$

in other words a case is assigned to the cluster whose attractor it has the highest semantic similarity to.

Once these three stages are completed, we have discovered all case knowledge, all narrow contexts within the case-base and assigned cases to the context they are most semantically similar to. In an equivalent fashion, the similarity between cases within a cluster can be discovered using the JS divergence. As such the lower the JS divergence, the higher the similarity and as will be seen in step 4, it is this similarity knowledge that forms the key to discovering semantically related cases.

Step 4 Localized Similarity Knowledge Discovery. Up to this point all knowledge discovered has been at the global/cluster level. In this step we discover localized similarity knowledge that defines the inner case structure of each cluster. We represent the relationships (similarities) between cases by a graph where each vertex in the graph represents a case. Any two vertices are connected by an undirected edge, whose weight denotes the distance between corresponding cases. This weight is determined as before using the JS divergence. The standard Kruskal's algorithm is used to find the minimum spanning tree (MST) which spans all graph vertices and has the minimum total weight for its edges. The knowledge within the MST can then be presented to a user as a complete description of the internal structure of a cluster relating to a narrow context. Useful knowledge includes,

cases nearer to the top of the tree are most similar to the narrow context, while those further away are less similar. Cases in close proximity within the MST are more similar than cases that are far apart. Cases in one branch of the tree are more similar than those in separate branches. This localized similarity knowledge can either be used to interactively browse the relevant cluster structure looking for useful cases or as a means of accurately classifying a new case.

3 Experiments

In this work we demonstrate the efficacy of the SOPHIA CBR system to textual case retrieval. In the first experiment we investigate the quality of the case and similarity knowledge by demonstrating the high degree of (semantic) similarity between cases within clusters. In the second, we investigate the quality of the retrieval process itself and show how it is ideally suited to case-based retrieval.

3.1 Case base description

In our experiments we use the well known Modified Apte ("ModApte") split of the Reuters-21578 collection (daviddlewis.com/resources/testcollections/reuters-21578) containing 9,603 training documents and 3,299 test documents. Not all documents are actually assigned a category. Therefore we only use those that have at least one topic (7,775 from the training set and 3,019 from the test set) in our experiments. All other documents were considered as background information. We use these documents to accumulate information about the document corpus. All training and background documents were used for case knowledge and similarity knowledge discovery (clustering & MST formation). Test documents were used as queries in the retrieval experiments only. It should be noted at this point that the majority of research carried out with this corpus in the past (mostly within the Information Retrieval community) has tended to focus on evaluations using the 10 most popular categories only [Sebastiani, 2002]. As such our task is much more challenging in that we attempt to use all 120 categories to evaluate our technique. The experiments described in this section could equally be applied to other document collections as SOPHIA CBR is domain and language independent.

Initially documents were parsed (transformed to cases) by converting all words from the title and body of a document into lower case, deleting stop-words using a standard stop-words list (SMART, 571 words) and using the Porter algorithm for stemming. We consider a word as any maximal sequence of symbols which start with a symbol in the range a-z, and ends with any symbol between a and z or any symbol between 0 and 9 and in between contains symbols from the set {a-z0-9_-/}. It should be noted that no additional information about the document set is used, apart from the title and body of the documents themselves. In particular, expert assigned knowledge such as category labels are not

used in the clustering/retrieval process and as such the approach is totally unsupervised in nature and therefore has as a consequence, no manual knowledge engineering overheads. In total, 38,088 different terms remained after parsing.

3.2 Experiment on similarity knowledge quality

In the first experiment we use the topic categories of cases as a means of independently assessing the quality of our similarity knowledge and hence our clustering process. Note that this topic categorization domain knowledge is *not* utilized as part of the process of case similarity determination (which is based solely on the textual content of cases) but as a means for *assessing* the effectiveness of the discovered similarity knowledge only. This experiment provides us with insight into the quality and meaningfulness of the SOPHIA TCBR clustering process. Our hypothesis is that if the system identifies cases as semantically similar then there should be a high probability that they share many of the same categories. To facilitate this we determine the degree of overlap between adjacent cases in the MST. We therefore evaluate the actual similarity of 2 cases a and b based on the similarity of the topics assigned to them by experts. Let $T(x)$ be the set of topics assigned to case x by an expert. To evaluate the similarity between $T(a)$ and $T(b)$ we use the Jaccard coefficient (JC).

$$JC = \frac{|T(a) \cap T(b)|}{|T(a) \cup T(b)|}$$

A coefficient of 1 indicates that both cases have identical topics assigned to them and have maximum similarity, while a value of 0 signifies no overlap between topics, indicating minimum similarity. Figure 1 shows the results of this experiment. The most striking observation from this graph is that the vast majority of cases (5796 pairs –75%) have an extremely high JC (0.9-1.0). This provides extremely strong evidence for the high quality of our similarity knowledge and confirms that semantically similar cases, i.e. those linked in the MST, have a large degree of overlap in their categories and therefore must of necessity be genuinely very similar. This also confirms that forming clusters based on narrow contexts combined with a MST based on JS divergence is a powerful approach for determining textual case similarity which also provides real meaning and transparency to users. That is users can easily see the context of all cases within a cluster and also which cases are most similar to others within the cluster. At the opposite end of the graph, where we can observe case pairs with poor category overlap, it can be seen that there are 1472 pairs of cases (19%) with very poor JC and almost no overlap between their categories. An interesting phenomenon can be observed at a JC value of 0.5-0.6. Here we see a slight rise in the number of case pairs. The reason for this is that many cases have only 2 categories and this peak represents the situation whereby they agree on one category and disagree

on the other (e.g. one case could have 2 categories and another only one, which they share).

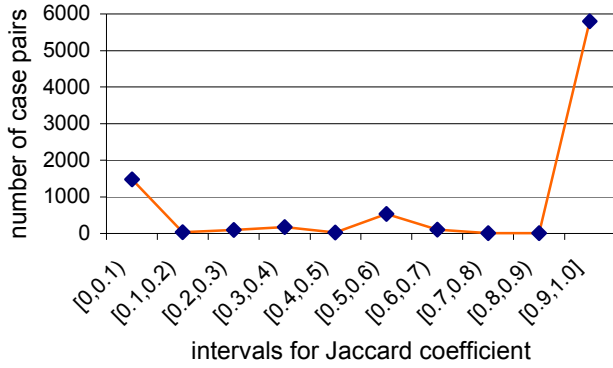


Figure 1 Assessment of similarity for MST edges Using JC

For all other JC values there are practically no case pairs. It is reasonable to conclude that clusters are not entirely homogeneous in terms of their topics. That is, each cluster may contain cases on different topics. Therefore it is to be expected that, within the MST when traversing from one case to another, topic shifts should be encountered. Whenever this occurs adjacent cases in the MST will differ on their topics. These shifts may be gradual, as evidenced by the rise in the graph around a JC of 0.5 (ie adjacent cases overlap on 50% of their topics) or radical, as evidenced by the rise at a JC of 0 (adjacent cases overlap on none of their topics). These results provide compelling evidence for quality of the case and similarity knowledge discovered and utilized by SOPHIA CBR.

3.3 Experiment on Case Retrieval

In this section, we propose a case retrieval approach designed to provide flexible querying plus a high retrieval accuracy and a good explanatory facility. SOPHIA enables queries to be generated using the traditional key word approach or by using query by example, where entire documents (or parts thereof) can be used as the query. SOPHIA then supplies the user with quality knowledge about “all” possibly relevant cases within the case base. It is important that this knowledge should be presented to the user in the context of the whole case collection. In other words, the user should have the facility to estimate:

- which region of the whole case base contains the most relevant cases,
- how large this region is,
- which semantically similar documents are relevant

We propose that the MST is ideally suited to presenting this knowledge to the user. Through a process of system supported browsing, the user can evaluate the relevancy of different parts of the tree, estimate its size and estimate the

topics of neighboring cases. This process is demonstrated in the subsequent experiment where we consider the following indexing and browsing scenario. The user has a target case d (test case/query example) and the system presents the MST of the most relevant cluster, based on the closest matching context attractor, for browsing. This tree is the MST of a cluster $C(z(d))$ whose context $z(d)$ is the smallest distance from the target case (as before we use JS divergence to evaluate distances). The nearest neighbor, $NN(z(d), d)$, to case d from within cluster $C(z(d))$, is selected from the MST. The user then starts browsing from this case. We will consider three notions of relevance in this experiment. Let $T(x)$ be the set of all topics assigned by the expert to case x . Then

Predicate $R_{=}(x, y)$ means that $T(x) = T(y)$

Predicate $R_{\subseteq}(x, y)$ means that $T(x) \subseteq T(y)$

Predicate $R_{\cap}(x, y)$ means that $T(x) \cap T(y) \neq \emptyset$

In other words if $R_{=}(x, y)$ then cases x and y are considered to be very relevant as they have exactly the same set of topics assigned by experts. If $R_{\subseteq}(x, y)$ then the relevance of case x to case y is slightly weaker than in the previous definition but the user can be sure that all topics of case x are also contained in case y . If $R_{\cap}(x, y)$ then we have the weakest notion of relevance in that both cases share at least one common topic. We say that the target case d is successfully matched with an existing case if, in the minimum spanning tree $MST(C(z(d)))$ of the cluster $C(z(d))$ there exists at least one relevant training case y within a distance of k edge links from the nearest neighbor case $NN(z(d))$. In this evaluation we include the nearest neighbor (NN) case as part of the evaluation and only consider edges which connect to training cases (edges connecting to background cases are ignored). Let predicate $S1(d, k)$ indicate that case d is successfully matched with an existing case within distance k , where relevance is determined by predicate $R_{=}(x, y)$. Similar predicates $S2(d, k)$ and $S3(d, k)$ are used when relevance is determined by predicates $R_{\subseteq}(x, y)$ and $R_{\cap}(x, y)$ respectively. Let $P1(k)$, $(P2(k), P3(k))$ be the probability that for a randomly selected test case d , the predicate $S1(d, k)$ ($S2(d, k)$, $S3(d, k)$) is true. Figure 2 shows how values of $P1(k)$, $P2(k)$ and $P3(k)$ depend on k . These results show that the knowledge discovered and utilized by SOPHIA CBR, when exploited by the process of interactive browsing, provides an ideal medium for locating and retrieving relevant cases. Using the most stringent definition of relevance, where all cases must have exactly the same topics (P1), it can be seen that 75% of the NN cases themselves are relevant (distance 0). It can be seen that this rises to almost 85% when cases within a vicinity of 3 links are considered. Considering larger vicinities, add little more to the relevancy. Defining relevancy as in P2, a similar picture is observed. This time the NN cases are relevant 80% of the time, rising to 87%

when $k=3$. Again little benefit is added by increasing k further.

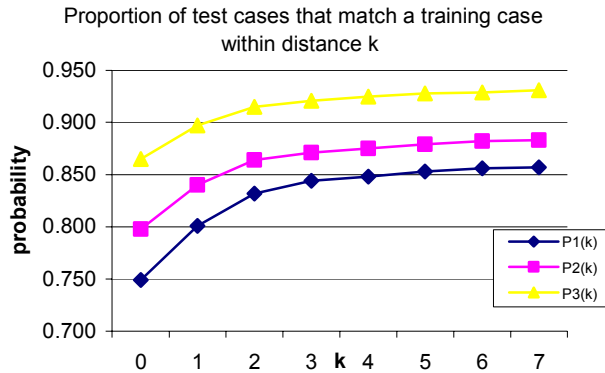


Figure 2 Showing how retrieval relevancy depends on k

Finally the least stringent definition of relevancy provides the best results. 86% of NN cases are relevant, rising to 92% when $k=3$. There is little improvement by increasing k further. These results are particularly encouraging, especially when considering that there are 120 possible categories a case can take and many cases have more than 1 category. It should be noted that the least stringent definition of similarity is the truest reflection of similarity within a TCBR system as the goal is to retrieve cases which are semantically similar (ie have some similar topics). This type of retrieval can be regarded as querying by example. A user can cut a piece of text from any other document (sourced from the web for example) and paste it into the SOPHIA system and through the process described retrieve documents that are semantically similar. The authors are unaware of any other scaleable TCBR system (or document clustering algorithm) that can accomplish this.

3 Conclusions

In this paper we present a novel approach for discovering case and similarity knowledge within a TCBR system. We describe how the cases are automatically grouped into semantically related clusters focused around discovered central contexts or themes. We show initially how the case and similarity knowledge discovered is of a very high quality and go on to show how the natural organization of the cases within clusters into a MST, provides a very natural environment to enable case based retrieval (query by example). Important advantages of this technique include the fact it is completely automated, requires no domain knowledge (and therefore no manual knowledge acquisition), it is language independent, can be used for case classification [Dobrynin et al., 2004], facilitates semantic similarity determination and very importantly, unlike all other clustering based approaches for document collections, it is scaleable for very large case bases. This is due to the fact that similarity between cases is only calculated at a local cluster level as op-

posed to the global level. Future work will include investigating the formation of sub clusters to aid retrieval and case based classification.

References

- [Baeza-Yates & Ribeiro-Neto, 1999] Baeza-Yates, R. & Ribeiro-Neto, B. *Modern Information Retrieval*. ACM Press, 1999.
- [Brüninghaus & Ashley, 2001] Brüninghaus, S. & Ashley, K., The role of Information Extraction for Textual CBR. In *Proceedings of the 4th International Conference on Case-Based Reasoning*, LNCS 2080, pp. 74-89, Springer, 2001.
- [Cunningham et al. 2004] Cunningham, C.M., Weber, R., Proctor, J.M, Fowler, C. & Murphy, M., Investigating Graphs in Textual Case-Based Reasoning. In *Proceedings of the 7th European Conference on Case-based Reasoning*, LNCS, pp. 573-586, Springer, 2004
- [Deerwester et al., 1990] Deerwester, S., Dumais, S. T., Furna, G.W., Landauer, T. K. & Harshman, R.. Indexing by Latent Semantic Analysis. *Journal of the American Society for Information Science*, 1990.
- [Dobrynin et al., 2004] Dobrynin, V., Patterson, D. & Rooney, N., Contextual Document Clustering. In *Proceedings of 26th European conference on Information Retrieval Research*, LNCS 2297, pp. 167-180, Springer, 2004.
- [Lin, 1991] Lin, J. Divergence Measures Based on the Shannon Entropy. *IEEE Transactions on Information Theory*, 37(1), pp. 145-151, 1991.
- [Hofmann, 1999] Hofmann, T. Probabilistic latent semantic analysis. In *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence*, pp. 289—296, 1999.
- [Kunze & Hubner, 1998] Kunze, M. & Hubner, A. CBR on Semi-structured Documents: The Experience Book and the FAIR Project. In *Proceedings of 6th German Workshop on CBR*, 1998
- [Lens, 1998] Lenz, M. Knowledge sources for textual CBR applications. In *Proceedings of the Workshop on Textual Case-Based Reasoning*, pp. 24-29. Menlo Park, CA, AAAI Press. 1998
- [Sebastiani, 2002] Sebastiani, F. Machine learning in automated text categorization. *ACM Computing Surveys*, 34(1):1-47, 2002.
- [Wiratunga et al., 2004] Wiratunga, N., Koychev, I., & Massie, S. *Feature Selection and Generalisation for Textual Case Retrieval*. In *Proceedings of 7th European Conference on Case-Based Reasoning*, pp. 806-820. LNAI, Springer, 2004.
- [Zelikovitz, S. & Hirsh 02] Zelikovitz, S. & Hirsh, H. Integrating Background Knowledge into Nearest-Neighbor Text Classification. In *Proceedings of 6th European Conference on Case-based Reasoning*, pp. 1-5, Springer, 2002.

Partial and Vague Knowledge for Similarity Measures

Timo Steffens

Institute of Cognitive Science
Kolpingstr. 7, 49074 Osnabrueck, Germany
tsteffen@uos.de

Abstract

This paper proposes to enhance similarity-based classification by virtual attributes from imperfect domain theories. We analyze how properties of the domain theory, such as partialness and vagueness, influence classification accuracy. Experiments in a simple domain suggest that partial knowledge is more useful than vague knowledge. However, for data sets from the UCI Machine Learning Repository, we show that vague domain knowledge that in isolation performs at chance level can substantially increase classification accuracy when being incorporated into similarity-based classification.

1 Introduction

One of the most prominent challenges in machine learning is to identify appropriate features for representing instances, since learning performance depends heavily on the features used. Particularly, the performance of similarity-based classification degrades with the number of irrelevant features [Griffiths and Bridge, 1996]. It is also known from work on constructive induction (CI) that adding features can improve classification accuracy [Aha, 1991]. While CI is a bottom-up approach, this paper proposes a top-down approach on identifying abstract features. The focus of CI was mainly on logical and rule-based processes, whereas this paper shows how additional features can extend similarity measures for similarity-based classification.

The main contribution of this paper is to show that additional features can be derived from domain theories that are imperfect. This will alleviate the knowledge acquisition bottleneck, as it reduces the requisites of obtaining expert knowledge. Although similarity-based classification is only used in domains where no perfect domain theories exist, usually there is imperfect domain knowledge and isolated chunks of knowledge [Aamodt, 1994; Bergmann *et al.*, 1994; Cain *et al.*, 1991; Porter *et al.*, 1990]. For example, in [Aamodt, 1994] open and weak domain theories were integrated into a case-based reasoning system. Similarly, matching knowledge was used to improve the performance of the well-known PROTOS system [Porter *et al.*, 1990]. Furthermore, it was shown that the combination of CBR and

a domain theory outperforms both CBR and the theory itself [Cain *et al.*, 1991]. In contrast to those weak theories, strong domain theories were used to filter irrelevant features [Bergmann *et al.*, 1994].

We present a new approach that exploits imperfect domain knowledge in similarity-based classification by inferring additional abstract features. Furthermore, we analyze the impact of the knowledge's vagueness and partialness.

The next section specifies the representation of cases, the similarity measure, and domain theories. Section 3 gives an overview over how additional features can improve classification accuracy. Section 4 reports experiments with two domains from the UCI Machine Learning Repository [Blake and Merz, 1998]. Finally, the last section concludes and outlines future work.

2 Representation of the CBR modules

2.1 Cases and the similarity measure:

A case C is made up of a set of attributes $\{A_1, A_2, \dots, A_n\}$. While the original attributes can be either discrete or numeric, the additional virtual attributes in this paper are binary.

Following the well-known local-global principle, we compute the similarity between two cases as the weighted average aggregation of the attributes' similarities:

$$\text{sim}(A, B) = \sum_{i=1}^n (\omega_i * s_i)$$

where $s_i = \text{sim}(A_i, B_i)$ are the local similarity values, and the ω_i are the corresponding weights.

2.2 Domain knowledge:

In this paper, we examine only domain knowledge that can be represented as a domain theory of the following form: A domain theory is a set of inference rules that relate concepts to each other. These rules specify which concepts exist in the domain and describe how abstract concepts can be inferred from more primitive ones [Bergmann *et al.*, 1994]. For example, consider the relation in the rule $A_1 \leftarrow A_2 \wedge A_3 < A_4$ which says that there are the binary attributes A_1, A_2 and the ordinal attributes A_3, A_4 . Also, the rule states that A_1 is satisfied, if A_2 is true and A_3 is smaller than A_4 . We assume that the case representation language is compatible with the

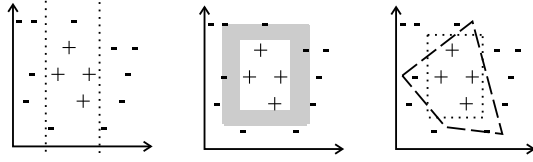


Figure 1: Properties of domain theories. The theories describe parts of the target concept, of which there are positive (+) and negative (-) instances. Left: Partial knowledge, only parts of the concept boundaries are known. Middle: Vague knowledge, concept boundaries are believed to be somewhere within the shaded areas. Right: Inconsistent knowledge, different rules make differing predictions.

language of the domain theory, either by sharing primitives or by using a bridging language. The formal definition of the domain theory is skipped here, since it is equivalent to Horn clauses, including logical connectors, equality, and comparison operators.

According to [Mooney and Ourston, 1991], the concepts in a domain theory can be divided into three types: observables are attributes that are directly represented in the cases. Classification goals are attributes that are to be inferred or approximated. All other attributes are called intermediates.

Intermediate attributes are the focus in this paper, because they are natural candidates for virtual attributes, that is, they can be added to the similarity measure in order to enhance the classification accuracy.

2.3 Properties of domain theories:

Domains in which CBR is applied usually lack a perfect domain theory. Hence, the domain theories (or parts thereof) that we work with have at least one of the following properties (cf. Figure 1):

- **Partialness:** This is the case if some parts of the domain are not modelled, for example a) if conditions are used but not defined, or b) the relation of intermediates or observables to the classification goal is not known, or c) the classification goal does not exist in the rulebase at all. Note that these situations correspond to gaps at the "top" or "bottom" of the domain theory [Mooney and Ourston, 1991].
- **Vagueness:** Values can only be given within a certain confidence interval. If a value is picked from such an interval, it is likely to be incorrect.
- **Inconsistency:** There are two or more rules (or even alternative theories) that make different classifications and it is not known which one is correct. CBR is often used to overcome this problem, because the cases provide knowledge on which classification is correct for certain cases.

In this paper, we focus on partial and vague theories.

3 Virtual attributes

Virtual attributes [Richter, 2003] are attributes that are not directly represented in the cases but can be inferred from the al-

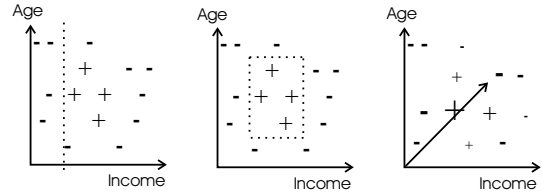


Figure 2: Types of virtual attributes. Left: A binary virtual attribute divides the instance space into instances satisfying or not satisfying it. Middle: A conjunction of binary attributes. Right: The most general type of virtual attributes is to add a dimension to the instance space.

ready existing attributes. They are useful if the monotonicity-principle is violated. If $sim(A, B) > sim(A, C)$ is necessary to reflect class membership, then there must at least be one pair of local similarities, so that $sim(A_i, B_i) > sim(A_i, C_i)$. If such a pair does not exist, the similarity measure must make use of interdependencies between attributes. For example, the similarity may not depend on two attributes A_1, A_2 themselves, but on their difference $A_1 - A_2$. Virtual attributes can express such interdependencies (e.g., $deposit(A) = income(A) - spending(A)$) and can also encapsulate non-linear relations.

We propose to use intermediate concepts of domain theories as virtual attributes. Virtual attributes can easily be added to the set of attributes of each instance.

Every virtual attribute forms an additional dimensions of the instance space (see Figure 2 (right)). This is most intuitive for numerical attributes. An example is the concept $expectedWealthTillRetirement(C) = (65 - age(C)) * income(C)$. Unfortunately, these dimensions can change assumptions about instance distributions and are most likely not orthogonal to the other dimensions, since they are inferrable from other attributes.

In this paper we focus on binary virtual attributes. Although formally they are additional dimensions, they can be visualized as separating lines within the original instance space (see Figure 2 (left)). They divide the instance space into two regions. For example, $taxFree(C) \leftarrow income(C) < 330$ may divide some instance space into salaries that are or are not subject to paying taxes in Germany. We will show that virtual attributes that describe target concept boundaries are especially useful.

Intermediate attributes that are fully defined (i.e., that do not have gaps at the bottom of the domain theory) can be computed from the values of observables and other intermediates. In order to use an intermediate as a virtual attribute, it is added to the local similarities of the similarity measure, that is, $s_i = 1$, iff both instances satisfy the intermediate concept or both do not satisfy it, and $s_i = 0$ otherwise. In the following, virtual attributes are assumed to be discrete.

Let us look at how binary virtual attributes influence classification. Assume for sake of illustration that the instance space is formed by the attributes *temp* and *press* denoting the temperature and pressure of a manufacturing oven. Let us assume furthermore that the (to be approximated) target

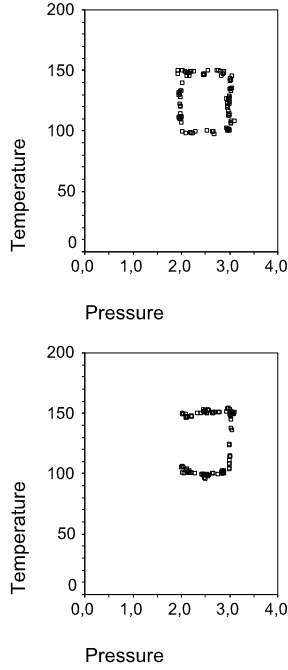


Figure 3: Distribution of errors for the target concept $hardened(C) \leftarrow temp(C) > 100 \wedge temp(C) < 150 \wedge press(C) > 2 \wedge press(C) < 3$ without virtual attributes (top) and with the virtual attribute $V(C) \leftarrow press(C) \leq 2$ (bottom).

concept is $hardened(C) \leftarrow temp(C) > 100 \wedge temp(C) < 150 \wedge press(C) > 2 \wedge press(C) < 3$.

The error distribution of an unweighted kNN-classifier for the target concept is depicted in Figure 3 (top). Not surprisingly, the misclassifications occur at the boundaries of the target concept.

Now let us analyze the effect of different amounts and different qualities of domain knowledge on the classification. In order to control the independent variables like partialness and correctness of the domain knowledge, we created a simple test domain. There were two continuous attributes X and Y , uniformly distributed over the interval $[0,100]$. The target concept was $T(C) \leftarrow X(C) > 30 \wedge X(C) < 70 \wedge Y(C) > 30 \wedge Y(C) < 70$. We used a square centered in the instance space as target concept, because it is one of the few concepts for which the optimal weight setting for kNN-classification can be calculated analytically. The optimal weight setting for the target concept is to use equal weights [Ling and Wang, 1997]. Thus, the accuracy of 1-NN with equal weights is the optimal accuracy that can be achieved without adding additional attributes. There were 100 randomly generated cases in the case-base and 200 test cases were used. Each experiment was repeated 1000 times with random cases in the case-base and random test cases.

3.1 Partialness of the domain theory:

We operationalize the partialness of the domain knowledge as number of known target concept boundaries. The more

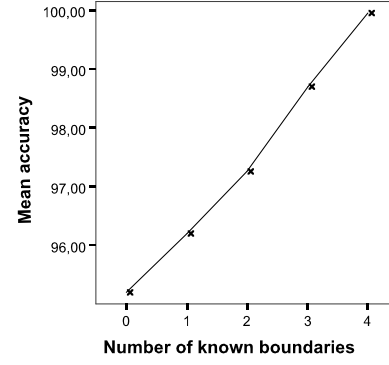


Figure 4: Percentage of correctly classified cases with different numbers of target concept boundaries described by virtual attributes.

boundaries are known, the less partial it is.

Adding virtual attributes that correctly specify a boundary of the target concept makes the misclassifications at those boundaries disappear (see Figure 3 (bottom)). Thus, by adding virtual attributes that describe a boundary correctly, the classification accuracy is increased (see Figure 4).

Obviously, even partial knowledge (e. g. adding only one virtual attribute) can improve classification accuracy. However, in this experiment we assumed that the virtual attributes were correct. In the next experiment we analyzed the influence of the correctness of virtual attributes.

3.2 Correctness:

Vague knowledge can be informally described as knowing that an attribute should be more or less at a certain value. The higher the vagueness, the higher is the probability for high incorrectness. We operationalize correctness of a virtual attribute as its distance from the correct value. We created virtual attributes of the form $V(C) \leftarrow X(C) < c$, where c was varied from 0 to 100 at steps of 5. Remember that the correct X -value (which was used in the domain theory to generate the cases) was 30. The accuracy of classification when adding these virtual attributes is depicted in Figure 5.

The results are a bit disappointing. The accuracy drops rapidly if the virtual attribute is inaccurate. Fortunately, the accuracy with inaccurate virtual attributes is not much lower than using no virtual intermediates (the similarity measure with no virtual attribute is equivalent to setting $c = 0$ or $c = 100$). The second peak at $X = 70$ which is the other boundary on the X -attribute is due to the fact that similarity-based classification is direction-less: only the position of the concept boundary has to be known, the side on which positive and negative instances are located is encoded in the cases.

These experiments with a simple domain suggest that partial knowledge is more useful than vague knowledge. Adding partial knowledge is likely to increase the classification accuracy, whereas vague knowledge is only useful if there is good evidence that the knowledge is correct.

In the next section we will evaluate the influence of virtual

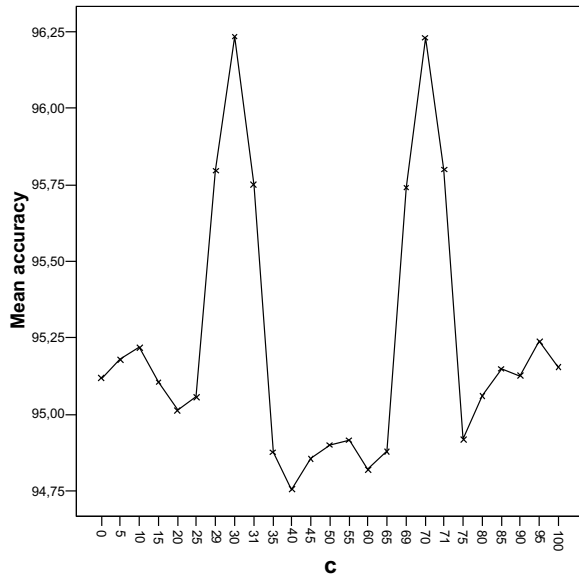


Figure 5: Accuracy of similarity measures using the virtual attribute $V(c) \leftarrow X(C) < c$, where c forms the horizontal axis. c -axis is stretched at the positions of the concept boundaries.

attributes in several domains from the UCI Machine Learning Repository [Blake and Merz, 1998].

4 Experiments

4.1 The domains:

The domain of the previous section allowed us to vary the correctness and partialness of the domain theory. However, since the domain was handcrafted and simple, we ran additional experiments with two data sets from the UCI Machine Learning Repository. Note that some data sets in the repository come along with perfect domain models, as the instances were created by those models. However, we used only data sets whose domain theories were imperfect.

- Japanese Credit Screening (JCS): This domain comes with a domain theory that was created by interviewing domain experts. Accordingly, the theory is imperfect and classifies only 81% of the cases correctly.
- Promoter gene sequences (PGS): This domain theory reflects the knowledge of domain experts in the field of promoter genes. It is highly inaccurate and performs at chance level when used in isolation [Towell *et al.*, 1990]. We included this domain to serve as a worst case scenario, since the domain knowledge is most inaccurate.

It is known that not all intermediate concepts will increase classification accuracy when used as virtual attributes [Steffens, 2004]. Hence, mechanisms to select or weight virtual attributes are necessary. In this paper we investigate whether weighting virtual attributes is more appropriate than selecting them. In the experiments we apply several existing weighting approaches which will be described in section 4.3.

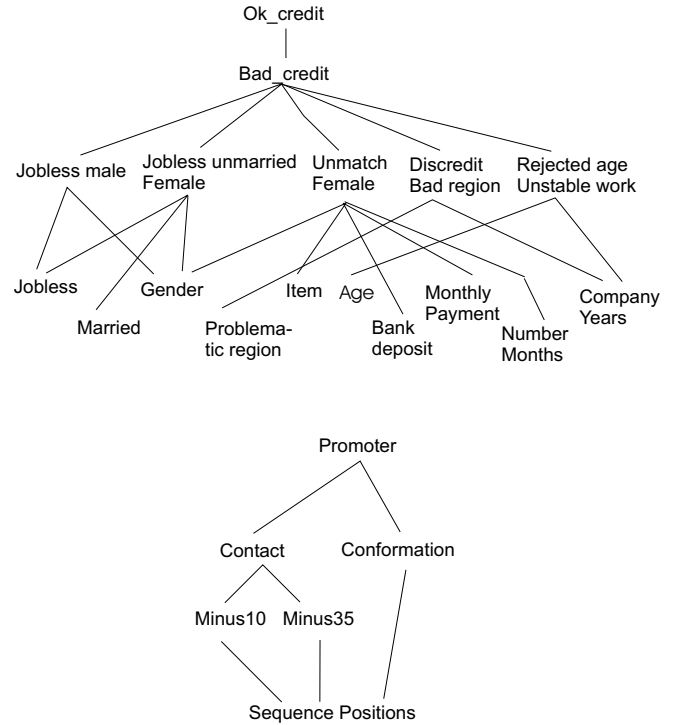


Figure 6: The domain theory of the JCS domain (top) and of the PGS domain (bottom).

4.2 The virtual attributes:

The domain theories of JCS and PGS have been created by domain experts for real world applications. Hence, they do not separate positive from negative instances in a perfect way. The accuracy of the JCS domain theory is 81%, the accuracy of the PGS domain theory is only 50%. The structure of both theories is depicted in Figure 6.

Most of the concepts process several observables. For example, *rejected_age_unstable_work* processes the observables *age* and *number_years*¹:

```
rejected_age_unstable_work(S) :-
    age_test(S, N1),
    59 < N1,
    number_years_test(S, N2),
    N2 < 3.
```

Although the concepts are very imperfect (i. e., they mis-categorize training cases), our experiments described in section 4.4 show that these concepts can improve classification accuracy when used as virtual attributes.

4.3 Weighting methods:

According to the classification of weighting methods as proposed in [Wettschereck *et al.*, 1997], we selected four methods with performance bias, and six with preset bias (i. e., statistical and information-theoretic methods).

¹This attribute denotes the number of years that the applicant worked at the same company.

- Performance bias: Weighting methods with a performance bias classify instances in a hill-climbing fashion. They update weights based on the outcome of the classification process. The performance bias performs well if there are many irrelevant features [Wettschereck *et al.*, 1997]. Since the intermediate concepts of the domain theories can be assumed to be relevant, we expected performance bias methods to perform badly.

1. EACH [Salzberg, 1991] increases the weight of matching features and decreases the weight of mismatching features by a hand-coded value.
 2. IB4 [Aha, 1992] is a parameter-free extension of EACH. It makes use of the concept distribution and is thus sensitive to skewed concept distributions. It assumes that the values of irrelevant features are uniformly distributed.
 3. RELIEF [Kira and Rendell, 1992] is a feature selection- rather than feature weighting-algorithm. It calculates weights based on the instance's most similar neighbors of each class and then filters attributes whose weights are below a hand-coded threshold. We used extensions for non-binary target classes and kNN with $k > 1$ as proposed in [Kononenko, 1994].
 4. ISAC [Bonzano *et al.*, 1997] increases weights of matching attributes and decreases weights of mismatching attributes by a value that is calculated from the ratio of the prior use of the instance. The more often the instance was retrieved for correct classifications, the higher the update value.
- Preset bias: The bias of the following methods is based on probabilistic or information-theoretic concepts. They process each training instance exactly once.
 1. CCF [Creedy *et al.*, 1992] binarizes attributes and weights them according to the classes' probability given a feature.
 2. PCF [Creedy *et al.*, 1992] is an extension of CCF which takes the distribution of the feature's values over classes into account. It calculates different weights for different classes.
 3. MI [Daelemans and van den Bosch, 1992] calculates the reduction of entropy in the class distribution by attributes and uses it as the attribute weight.
 4. CD [Nunez *et al.*, 2002] creates a correlation matrix of the discretized attributes and the classes. The weight of an attribute increases with the accuracy of the prediction from attribute value to class.
 5. VD [Nunez *et al.*, 2002] extends CD in that it considers both the best prediction for a class and the predictions of all attributes.
 6. CVD [Nunez *et al.*, 2002] combines CD and VD.

4.4 Results:

For brevity, we will refer to the similarity measure which uses only observables as the *non-extended* measure. The similarity measure which uses virtual attributes will be called *extended*. We used the leave-one-out evaluation method.

Table 1: Classification accuracies of the non-extended and the extended similarity measures. The columns report the accuracies for the unweighted classification and for several weighting methods.

Domain	unw.	EACH	RELIEF	IB4	ISAC
JCS (w/o)	74.19	74.19	78.23	74.19	72.58
JCS (w/)	74.19	72.58	<u>79.03</u>	72.58	<u>79.03</u>
PGS (w/o)	86.79	89.62	96.23	88.68	50.0
PGS (w/)	85.85	<u>93.40</u>	96.23	<u>90.57</u>	<u>96.23</u>
CCF	PCF	MI	CD	VD	CVD
72.58	72.58	74.19	74.19	72.58	71.77
<u>73.39</u>	<u>75.0</u>	<u>75.0</u>	<u>77.42</u>	<u>75.0</u>	<u>75.0</u>
85.85	87.74	68.87	88.68	77.36	83.02
<u>91.51</u>	86.79	<u>98.11</u>	88.68	<u>97.17</u>	<u>87.74</u>

For most of the weighting methods, the extended similarity measure performs better than the non-extended one. In table 1 we underline the accuracy of the extended similarity measure if it outperformed the non-extended similarity measure when using the same weighting method. In the PGS domain, seven of ten weighting methods perform better if the similarity measure is extended with virtual attributes. Even more so, in the JCS domain the accuracies of eight of ten weighting methods were improved by using virtual attributes.

In its optimal setting, with an accuracy of 98.11% our approach performs also better than the results from the literature reported for the PGS domain. The accuracy of KBANN in [Towell *et al.*, 1990] is 96.23%, which to our knowledge was the highest accuracy reported so far and also used the leave-one-out evaluation. We found no classification accuracy results for JCS in the literature².

Obviously, these improvements are not restricted to a certain class of weighting methods. Methods with performance bias (most notably ISAC), information-theoretic bias (i. e. MI), and with a statistical correlation bias (e. g. VD) benefit from processing virtual attributes.

Even in the PGS domain, the improvements are substantial. This is surprising, since the domain knowledge is the worst possible and classifies at chance level when used for rule-based classification. This is a promising result as it shows that adding intermediate concepts may increase accuracy even if the domain theory is very inaccurate. We hypothesize that this is due to the fact that even vague rules-of-thumb provide some structure in the instance space which will be exploited by the similarity measure.

5 Conclusion and future work

The main contribution of this paper is to show that imperfect domain knowledge can benefit similarity-based classification. This facilitates knowledge elicitation from domain experts as it removes the requirements of completeness and accuracy. Our experiments in a simple domain suggest that partial knowledge is more useful than vague knowledge. How-

²The domain often referred to as 'credit screening' with 690 instances is actually the credit card application domain.

ever, we showed in the domains from the Machine Learning Repository that even highly inaccurate domain knowledge can be exploited to drastically improve classification accuracy. Future work includes experiments in further domains and transforming intermediate attributes by feature generation [Fawcett and Utgoff, 1992].

References

- [Aamodt, 1994] Agnar Aamodt. Explanation-driven case-based reasoning. In Stefan Wess, Klaus-Dieter Althoff, and Michael M. Richter, editors, *Topics in Case-Based Reasoning*, pages 274–288. Springer, 1994.
- [Aha, 1991] David W. Aha. Incremental constructive induction: An instance-based approach. In Lawrence Birnbaum and Gregg Collins, editors, *Proceedings of the Eighth International Workshop on Machine Learning*, pages 117–121, Evanston, IL, 1991. Morgan Kaufmann.
- [Aha, 1992] David W. Aha. Tolerating noisy, irrelevant and novel attributes in instance-based learning algorithms. *International Journal of Man-Machine Studies*, 36(2):267–287, 1992.
- [Bergmann et al., 1994] Ralph Bergmann, Gerhard Pews, and Wolfgang Wilke. Explanation-based similarity: A unifying approach for integrating domain knowledge into case-based reasoning. In Stefan Wess, Klaus-Dieter Althoff, and Michael M. Richter, editors, *Topics in Case-Based Reasoning*, pages 182–196. Springer, 1994.
- [Blake and Merz, 1998] C. L. Blake and C. J. Merz. UCI repository of machine learning databases, 1998.
- [Bonzano et al., 1997] Andrea Bonzano, Padraig Cunningham, and Barry Smyth. Using introspective learning to improve retrieval in cbr: A case study in air traffic control. In David Leake and Enric Plaza, editors, *Proceedings of the second ICCBR conference*, pages 291–302, Berlin, 1997. Springer.
- [Cain et al., 1991] Timothy Cain, Michael J. Pazzani, and Glenn Silverstein. Using domain knowledge to influence similarity judgements. In *Proceedings of the Case-Based Reasoning Workshop*, pages 191–198, Washington D.C., U.S.A., 1991.
- [Creedy et al., 1992] Robert H. Creedy, Brij M. Masand, Stephen J. Smith, and David L. Waltz. Trading mips and memory for knowledge engineering. *Communications of the ACM*, 35(8):48–64, 1992.
- [Daelemans and van den Bosch, 1992] Walter Daelemans and Antal van den Bosch. Generalization performance of backpropagation learning on a syllabification task. In *Proceedings of the Third Twente Workshop on Language Technology: Connectionism and Natural Language Processing*, pages 27–37, Enschede, The Netherlands, 1992. Unpublished.
- [Fawcett and Utgoff, 1992] Tom Elliott Fawcett and Paul E. Utgoff. Automatic feature generation for problem solving systems. In Derek H. Sleeman and Peter Edwards, editors, *Proceedings of the 9th International Conference on Machine Learning*, pages 144–153. Morgan Kaufmann, 1992.
- [Griffiths and Bridge, 1996] Anthony D. Griffiths and Derek G. Bridge. A yardstick for the evaluation of case-based classifiers. In Ian D. Watson, editor, *Proceedings of Second UK Workshop on Case-Based Reasoning*, 1996.
- [Kira and Rendell, 1992] Kenji Kira and Larry A. Rendell. A practical approach to feature selection. In Derek H. Sleeman and Peter Edwards, editors, *Proceedings of the Ninth International Workshop on Machine Learning*, pages 249–256. Morgan Kaufmann Publishers Inc., 1992.
- [Kononenko, 1994] Igor Kononenko. Estimating attributes: Analysis and extensions of RELIEF. In F. Bergadano and L. de Raedt, editors, *Proceedings of the European Conference on Machine Learning*, pages 171–182, Berlin, 1994. Springer.
- [Ling and Wang, 1997] Charles X. Ling and Hangdong Wang. Computing optimal attribute weight settings for nearest neighbour algorithms. *Artificial Intelligence Review*, 11:255–272, 1997.
- [Mooney and Ourston, 1991] Raymond J. Mooney and Dirk Ourston. Constructive induction in theory refinement. In Lawrence Birnbaum and Gregg Collins, editors, *Proceedings of the Eighth International Machine Learning Workshop*, pages 178–182, San Mateo, CA, 1991. Morgan Kaufmann.
- [Nunez et al., 2002] H. Nunez, M. Sanchez-Marre, U. Cortes, J. Comas, I. Rodriguez-Roda, and M. Poch. Feature weighting techniques for prediction tasks in environmental processes. In *Proceedings of the 3rd Workshop on Binding Environmental Sciences and Artificial Intelligence (BESAI 2002)*, 2002.
- [Porter et al., 1990] Bruce W. Porter, Ray Bareiss, and Robert C. Holte. Concept learning and heuristic classification in weak-theory domains. *Artificial Intelligence*, 45(1-2):229–263, 1990.
- [Richter, 2003] Michael M. Richter. Fallbasiertes Schliessen. *Informatik Spektrum*, 3(26):180–190, 2003.
- [Salzberg, 1991] Steven Salzberg. A nearest hyperrectangle learning method. *Machine Learning*, 6(3):251–276, 1991.
- [Steffens, 2004] Timo Steffens. Similarity-measures based on imperfect domain-theories. In Steffen Staab and Eva Onainda, editors, *Proceedings of STAIRS 2004*, pages 193–198. IOS Press, Frontiers in Artificial Intelligence and Applications, 2004.
- [Towell et al., 1990] Geoffrey G. Towell, Jude W. Shavlik, and Michael O. Noordenier. Refinement of approximate domain theories by knowledge based neural network. In *Proceedings of the Eighth National Conference on AI*, volume 2, pages 861–866, 1990.
- [Wettschereck et al., 1997] Dietrich Wettschereck, David W. Aha, and Takao Mohri. A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms. *Artificial Intelligence Review*, 11:273–314, 1997.

A flexible and robust similarity measure based on contextual probability

Hui Wang *

School of Computing and Mathematics
University of Ulster at Jordanstown
Northern Ireland
h.wang@ulster.ac.uk

Werner Dubitzky

School of Biomedical Science
University of Ulster at Coleraine
Northern Ireland
w.dubitzky@ulster.ac.uk

Abstract

Arguably, analogy is one of the most important aspects of intelligent reasoning. It has been hypothesized that, given suitable background knowledge, analogy can be viewed as a logical inference process. This study follows another school of thought that argues that similarity can provide a probabilistic basis for inference and analogy. Most *similarity measures* (which are frequently viewed as being conceptually equivalent to distance measures) are restricted to either nominal or ordinal attributes, and some are confined to classification tasks. This paper proposes a flexible similarity measure that is task-independent and applies to both nominal and ordinal data in a conceptually uniform way. The proposed similarity measure is derived from a probability function and corresponds to the intuition that if we consider all neighborhoods around a data point, the data points closer to this point should be included in more of these neighborhoods than more distant points. Experiments we have conducted to demonstrate the usefulness of this measure indicate that it fares very competitively with commonly used similarity measures.

1 Introduction

Natural and artificial intelligent agents rely on different inference and reasoning strategies to achieve their goals and maintain their intended behavior. Arguably, two of the more successful strategies are based on the notions of *analogy* ¹

*Also LITA, Université de Metz, Ile du Saulcy, 57045 Metz Cedex, France.

¹Here we consider the subject of analogy in the following, relatively narrow, sense: objects are represented by feature vectors rather than the typical graphs; we do not consider transfer of knowledge from one object to another; we do not consider cross-domain knowledge but assume all objects share common set of attributes.

and *similarity*. This is evidenced by the large volume of AI research in the areas of *analogical reasoning* and *case-based reasoning* [Vosniadou and Ortony, 1989; Kolodner, 1993]. In the context of available background knowledge, analogy can be viewed as a logical inference process [Davies and Russell, 1987]. Another way of looking at analogy holds that similarity can provide a probabilistic basis for inference, and that a quantitative framework can be developed for the probability that an analogy is correct as a function of the degree of similarity measured or observed [Russell, 1986]. This study is concerned with the latter view of modeling analogy.

Distance measures (or, equivalently, similarity measures) are central to many areas related to AI, including reasoning under uncertainty, knowledge-based systems, machine learning, pattern recognition, data mining, analogical, case-based, instance-based reasoning, and to other fields such as statistics, operations research and decision theory. A large number of distance metrics and measures are available and a particular choice may have considerable implications for the success of the problem that is to be addressed.

Distance functions are broadly categorized into those that can handle *ordinal* input data, *nominal* input data, and *heterogeneous* input data, consisting of both ordinal and nominal data ². Ordinal distance functions make use of the intrinsic total ordering relation in the underlying attribute values, which are either continuous (e.g. weight of an object) or discrete (e.g. number of obstacles). A nominal or symbolic attribute is a discrete attribute whose values do not necessarily exhibit a total order relation. For example, an attribute representing the role of crew member may have the values *scientist*, *mission specialist* and *commander*. Using an ordinal distance measurement on such values is meaningless.

Because the notion of ‘distance’ is intrinsically numerical, most available distance measures are defined for data with ordinal attributes. However, distance measures that can

²Here we consider two scales of measurement: ordinal and nominal. In an ordinal scale, values (discrete or continuous) are ordered whereas in a nominal scale, (discrete) values are unordered.

process nominal attributes are required by many modern AI algorithms. Some measures that handle nominal attributes do exist. The most well known measure of this kind is the Value Difference Metric (VDM) [Stanfill and Waltz, 1986]. It is defined in terms of attribute values that are conditioned by posterior probabilities of a class. Hence, like some other distance measures, the VDM is restricted to classification tasks.

To cope with heterogeneous data, containing both nominal and ordinal attributes, two approaches are commonly taken: (1) The data is transformed into one of the two data types that complies with the distance measure used, and (2) two types of distance measures are combined and handle the data separately in accordance with the data type (see, for example, [Wilson and Martinez, 1997]). However, both approaches are problematic when it comes to the interpretation of the results.

The novel distance measure proposed in this study is called *Neighborhood Counting Metric* (NCM). It is derived from a probability function and it can handle both nominal and ordinal attributes in a conceptually uniform way. Intuitively, it can be understood or interpreted as follows (see Figure 1): If we consider all neighborhoods, N , around a data point, t , then those data points closer to t should be included in more of these neighborhoods, N , than points that are not close to t . Usually neighborhoods are interpreted in terms of distance. However, if we adopted this interpretation, we would define one distance function in terms of another. To avoid this dilemma, we interpret neighborhood without distance through the concept of *hypertuples* [Wang *et al.*, 1999] for both nominal and ordinal attributes. As a result, our distance measure applies to both ordinal and nominal attributes in a uniform way.

The new NCM is conceptually simple, it is straightforward to implement, and it has the added property that it is independent of the underlying analytical or reasoning task (e.g. classification). The measure's clear and unambiguous meaning is defined by *the number of neighborhoods of a query that include or cover a given data point*.

Paper outline: Section 2 presents a short review of distance measures relevant to this study; it is followed by Sections 3 and 4, which present mathematical details of the new similarity measure. In Section 5 the empirical evaluation of the method is presented and its results discussed. The paper concludes with a summary and a brief discussion of future work.

2 A brief review of important distance functions

Two of the most common distance functions are *Euclidean distance* and the *Hamming distance*. The former is restricted to ordinal and the latter is usually used for nominal attributes. The *Heterogeneous Euclidean-Overlap Metric* (HEOM) [Wilson and Martinez, 1997] combines the Ham-

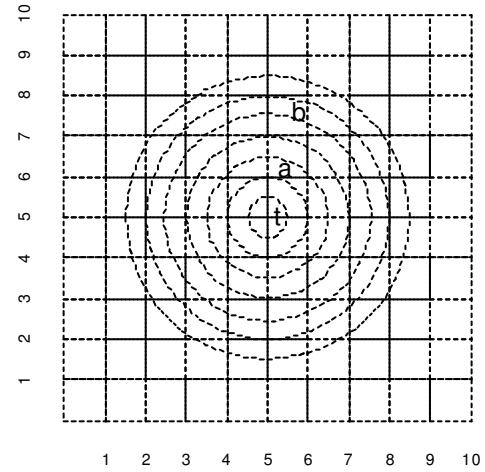


Figure 1: An illustrative example. Each of the 7 concentric circles represents a neighborhood of data point t defined by some distance measure. Data point a is covered by 5 neighborhoods whereas b by only 2. Geometrically, a is clearly closer (more similar) to t than b .

ming and the normalized Euclidean distance functions and can therefore be used on heterogeneous data.

The *Value Difference Metric* (VDM) [Stanfill and Waltz, 1986] is designed to handle nominal attributes in classification tasks only. It uses pre-computed statistical properties from available training data and can be interpreted in terms of probabilities. The *Heterogeneous Value Difference Metric* (HVD) [Wilson and Martinez, 1997] combines the Euclidean distance and the VDM. It is therefore able to handle heterogeneous data, but because of its VDM heritage it is confined to classification tasks.

The *Interpolated Value Difference Metric* (IVDM) [Wilson and Martinez, 1997] extends the VDM to scenarios involving ordinal attributes. In the learning phase, it employs a discretization step to collect statistics and determine the probability for the discretized values ($P_{a,x,c}$ in the VDM formula), but then retains the continuous values in the training data for later use in the application or testing phase. The IVDM requires a non-parametric probability density estimation to determine the probability values for each class. The *Discretized Value Difference Metric* (DVDM) is the same as the IVDM except that it avoids the retention of the original continuous values but uses only the discretized values.

[Blanzieri and Ricci, 1999] introduced the *Minimal Risk Metric* (MRM), a probability-based distance measure for classification and case-based reasoning. It minimizes the risk of misclassification and depends on probability estimation techniques. As a result, the MRM exhibits a high computational complexity and is task-dependent.

3 A probability function

Let V be a (non-empty) universe of discourse. A Σ -field $\mathcal{F}$ on V is a collection of subsets of V , such that:

1. $V \in \mathcal{F}$;
2. If $A \in \mathcal{F}$ then $A' \in \mathcal{F}$, where A' is the complement of A ;
3. If $A, B, \dots \in \mathcal{F}$, then $A \cup B \cup \dots \in \mathcal{F}$.

A probability function P over V is a mapping from $\mathcal{F}$ to $[0, 1]$ satisfying the three axioms of probability [Ash, 1972]. If P is restricted to V , it is called a probability mass function (discrete) or probability density function (continuous).

Suppose we have a probability function P as defined above. For $X \in \mathcal{F}$ let $f(X)$ be a non-negative measure of X satisfying $f(X_1 \cup X_2) = f(X_1) + f(X_2)$ if $X_1 \cap X_2 = \emptyset$. As an example, we can take $f(X)$ for the cardinality of X .

Consider a function $G: \mathcal{F} \rightarrow [0, 1]$ such that, for $X \in \mathcal{F}$,

$$G(X) = \sum_{E \in \mathcal{F}} P(E) f(X \cap E) / K$$

where $K = \sum_{E \in \mathcal{F}} P(E) f(E)$. It can be easily shown that G is a probability function.

$G(X)$ is calculated from all those $E \in \mathcal{F}$ that overlap with X (i.e., $f(X \cap E) \neq 0$). These E 's are relevant to X and serve as the *contexts* in which $G(X)$ is induced. Each such E is called a *neighborhood* of X . In other words a neighborhood of X is an element E of $\mathcal{F}$, i.e., a subset of V , such that E overlaps X .

In practice P is usually not known, but a *sample* of data drawn from the data space according to P is commonly available. P can be estimated from the sample by either parametric or non-parametric techniques. In some tasks (e.g., classification) the point-wise probability is needed. When there is insufficient data, which is not uncommon, the point-wise probability may be difficult to estimate using non-parametric methods. However, it is likely that probability can be estimated for some *regions* (contexts) in the data space based on the sample. Using the G probability function, we can estimate the G probability for any single data point from knowledge of the P probabilities of various regions or contexts. This provides us with a formal way of inference about probability under incomplete (hence uncertain) situations.

To calculate the G probability for $X \in \mathcal{F}$ in practice, we need to

1. Find the set of all neighborhoods appropriate to the problem at hand,
2. Estimate the P probability for every neighborhood,
3. Finally, estimate (calculate) the G probability for X according to its definition.

In classification tasks, we can further calculate the conditional G probability of a class given a single data point in a similar

fashion. While this may appear to be computationally expensive, it is often possible to derive a simple formula for G .

Depending on what $\mathcal{F}$ is and how neighborhoods are defined, we can use the G probability for different tasks. In Section 4 we take $\mathcal{F}$ to be the set of all regions in a multi-dimensional space definable as hypertuples and our similarity measure is derived from this interpretation.

3.1 Estimation of G

This section describes how G is estimated for a data point from data samples. Let D be a sample of data drawn from V according to an unknown probability distribution P . Our aim is to calculate $\hat{G}(t|D)$ for any $t \in V$.

To calculate G we need P , which can be estimated from data assuming the *principle of indifference* as follows: For any $E \in \mathcal{F}$,

$$\hat{P}(E|D) = |E|/n$$

where $|E|$ is the number of elements in E and n is the number of elements in D . Additionally we assume that $f(X) = |X|$ for $X \in \mathcal{F}$.

We then have,

$$\begin{aligned} \hat{G}(t|D) &= \sum_{E \in \mathcal{F}} \frac{\hat{P}(E|D) \times f(E \cap t)}{K} && \text{definition} \\ &= \sum_{E \in \mathcal{F}} \frac{\hat{P}(E|D) \times |E \cap t|}{K} && \text{specification of } f \\ &= \sum_{E \in \mathcal{F}, t \in E} \frac{\hat{P}(E|D)}{K} && \text{assumption that } t \text{ is in } E \text{ so } E \cap t = t \\ &= \frac{1}{nK} \sum_{E \in \mathcal{F}, t \in E} |E| && \text{estimation of } P \text{ by principle of indifference} \\ &= \frac{1}{nK} \sum_{x \in D} \text{cov}(t, x) && \text{expansion and then re-organisation} \end{aligned}$$

where K is a normalisation factor, and $\text{cov}(t, x)$ is the number of such $E \in \mathcal{F}$ that covers both t and x . We call this number the *cover* of x with respect to t .

To obtain $\text{cov}(t, x)$, a straightforward approach would be to iterate over all $E \in \mathcal{F}$ and check if E covers both t and x . Clearly, such a process is undesirable because of its exponential complexity. In Section 4 we will present an efficient way of calculating $\text{cov}(t, x)$.

4 Measuring distance through counting

This section considers an interpretation of neighborhood and derives a formula for calculating $\text{cov}(t, x)$, which is then taken as a measure of similarity or distance.

4.1 Neighborhood

Pursuing a non-distance-based conceptualization of neighborhood, we interpret a *neighborhood* as a *hypertuple* [Wang et al., 1999], and a *neighbor* as a data point (or tuple) covered by some neighborhood. So a neighborhood of a tuple is a hypertuple that covers the tuple.

Hypertuples

Let $R = \{a_1, a_2, \dots, a_n\}$ be a set of attributes, and $\text{dom}(a)$ be the domain of attribute $a \in R$. Furthermore let $V \stackrel{\text{def}}{=} \prod_{i=1}^n \text{dom}(a_i)$ and $L \stackrel{\text{def}}{=} \prod_{i=1}^n 2^{\text{dom}(a_i)}$. V is the *data space* defined by R , and L an *extended data space*. A (given) *data set* is $D \subseteq V$ – a sample of V .

The attributes can be either *ordinal* or *nominal*. For simplicity, we assume the domain of any attribute is finite, but the results are not limited to finite domains.

If we write an element $t \in V$ by $\langle v_1, v_2, \dots, v_n \rangle$ then $v_i \in \text{dom}(a_i)$. If we write $h \in L$ by $\langle s_1, s_2, \dots, s_n \rangle$ then $s_i \subseteq \text{dom}(a_i)$ or $s_i \subseteq \text{dom}(a_i)$.

An element of L is called a *hypertuple*, and an element of V a *simple tuple*. The difference between the two is that a field within a simple tuple is a value while a field within a hypertuple is a set. If we interpret $v_i \in \text{dom}(a_i)$ as a singleton set $\{v_i\}$, then a simple tuple is a special hypertuple. Thus we can embed V into L , so $V \subseteq L$.

Consider two hypertuples h_1 and h_2 , where $h_1 = \langle s_{11}, s_{12}, \dots, s_{1n} \rangle$ and $h_2 = \langle s_{21}, s_{22}, \dots, s_{2n} \rangle$. We say h_1 is *covered* by h_2 (or h_2 *covers* h_1), written $h_1 \leq h_2$, if for $i \in \{1, 2, \dots, n\}$,

$$\begin{cases} \forall x \in s_{1i}, \min(s_{2i}) \leq x \leq \max(s_{2i}) & \text{if } a_i \text{ is ordinal} \\ s_{1i} \subseteq s_{2i} & \text{if } a_i \text{ is nominal} \end{cases}$$

Furthermore the *sum* of h_1 and h_2 , written by $h_1 + h_2 \stackrel{\text{def}}{=}$ $\langle s_1, s_2, \dots, s_n \rangle$, is: for each $i \in \{1, 2, \dots, n\}$,

$$s_i = \begin{cases} \{x \in \text{dom}(a_i) : \min(s_{1i} \cup s_{2i}) \leq x \leq \max(s_{1i} \cup s_{2i})\}, & \text{if } a_i \text{ is ordinal} \\ s_{1i} \cup s_{2i}, & \text{if } a_i \text{ is nominal} \end{cases}$$

The *product* operation $*$ can be similarly defined. It turns out that $\langle L, \leq, +, * \rangle$ is a lattice.

For a simple tuple t , $t(a_i)$ represents the projection of t onto attribute a_i . For a hypertuple h , $h(a_i)$ is similarly defined.

A hypertuple can be generated by taking one subset from each attribute. Let a_i be an attribute, $i = 1, 2, \dots, n$; S_i be the set of all subsets of the domain of a_i ; and $N_i'' \stackrel{\text{def}}{=} |S_i|$. Then a hypertuple h is an element of $\prod_i S_i$. Therefore the number of all hypertuples is $\prod_i N_i''$.

If a_i is nominal, then $S_i = 2^{\text{dom}(a_i)}$ and so $N_i'' = 2^{m_i}$, where $m_i = |\text{dom}(a_i)|$. If a_i is ordinal, then an element $s \in S_i$ corresponds to an interval $[\min(s), \max(s)]$. As a result, some elements of S_i may correspond to the same interval, and hence become equivalent. In general we have the following number of *distinctive* intervals for an ordinal attribute: $N_i'' = \sum_{j=0}^{m_i-1} (m_i - j) = \sum_{j=1}^{m_i} j = \frac{m_i(m_i+1)}{2}$.

Neighborhoods as hypertuples

For any simple tuple t , a neighborhood of t is taken to be a hypertuple h such that $t \leq h$; and a neighbor of t with respect

to h is $x \in V$ such that $x \leq h$. By this definition any simple tuple has a neighborhood. At least the maximal hypertuple in the extended data space is a neighborhood of any simple tuple since the maximal hypertuple covers all simple tuples.

For a query $t \in V$, not all hypertuples in $\prod_i S_i$ are neighborhoods of t . For a hypertuple h to be a neighborhood of t we must have $t(a_i) \in h(a_i)$ for all i . Therefore, to generate a neighborhood of t , we can take an $s_i \in S_i$ such that $t(a_i) \in s_i$ for all i , resulting in a hypertuple $\langle s_1, s_2, \dots, s_n \rangle$. If a_i is nominal, the number of $s_i \in S_i$ such that $t(a_i) \in s_i$ is $N_i' = \sum_{i=0}^{m_i-1} \binom{m_i-1}{i} = 2^{m_i-1}$ since s_i is any subset of $\text{dom}(a_i)$ that is the super set of $t(a_i)$. If a_i is ordinal, this number is $N_i' = (\max(a_i) - t(a_i) + 1) \times (t(a_i) - \min(a_i) + 1)$ since $(\max(a_i) - t(a_i) + 1)$ is the number of ordinal values above $t(a_i)$, and $(t(a_i) - \min(a_i) + 1)$ is such a number below $t(a_i)$. Any pair of values from the two parts respectively forms an interval.

To summarize, the number of neighborhoods of t is $\prod_i N_i'$, where

$$(1) \quad N_i' = \begin{cases} 2^{m_i-1}, & \text{if } a_i \text{ is nominal} \\ (\max(a_i) - t(a_i) + 1) \times (t(a_i) - \min(a_i) + 1), & \text{if } a_i \text{ is ordinal.} \end{cases}$$

Cover of simple tuples

Under the above interpretation of neighborhood, we know exactly the number of all neighborhoods of a given simple tuple t . Here we present an efficient way of calculating $\text{cov}(t, x)$, the number of neighborhoods of t that cover x , which is needed in Section 3.1.

Consider two simple tuples $t = \langle t_1, t_2, \dots, t_n \rangle$ and $x = \langle x_1, x_2, \dots, x_n \rangle$. A neighborhood h of t covers t by definition, i.e., $t \leq h$. What we need to do is to check if h covers x as well. In other words, we want to find all hypertuples that cover both t and x .

Eq.1 specifies the number of all simple tuples that cover t only. We take a similar approach here by looking at each attribute and explore the number of subsets that can be used to generate a hypertuple covering both t and x . Multiplying these numbers across all attributes gives rise to the number we require.

Consider attribute a_i . If a_i is ordinal, then the number of intervals that can be used to generate a hypertuple covering both x_i and t_i is as follows:

$$N_i = (\max(a_i) - \max(\{x_i, t_i\}) + 1) \times (\min(\{x_i, t_i\}) - \min(a_i) + 1).$$

If a_i is nominal, the number of subsets for the same purpose is:

$$N_i = \begin{cases} 2^{m_i-1}, & \text{if } x_i = t_i \\ 2^{m_i-2}, & \text{otherwise} \end{cases}$$

Recall that $m_i = |dom(a_i)|$.

To summarize, the number of neighborhoods of t covering x is $cov(t, x) = \prod_i N_i$, where

$$(2) \quad N_i = \begin{cases} (\max(a_i) - \max(\{x_i, t_i\}) + 1) \times (\min(\{x_i, t_i\}) - \min(a_i) + 1), & \text{if } a_i \text{ is ordinal} \\ 2^{m_i-1}, & \text{if } a_i \text{ is nominal and } x_i = t_i \\ 2^{m_i-2}, & \text{if } a_i \text{ is nominal and } x_i \neq t_i \end{cases}$$

4.2 Use of cover as similarity measure

We use $cov(t, x)$ as a measure of similarity between t and x , which we call the *Neighborhood Counting Metric* or simply *NCM*. That is, for any two tuples x and y , the NCM between them is

$$(3) \quad NCM(x, y) = cov(x, y) = \prod_{i=1}^n N_i$$

where n is the number of attributes and N_i is given by Eq.2. It is clear that $NCM(x, y) \geq 0$, $NCM(x, x) \geq NCM(x, y)$ and $NCM(x, y) = NCM(y, x)$. Therefore the NCM is reflexive and symmetric, the properties generally required for a similarity measure [Osborne and Bridge, 1997].

This measure can be interpreted intuitively as follows. If we consider all neighborhoods around a data tuple, those tuples closer to this tuple should be included in more neighborhoods and those farther away should be included in fewer neighborhoods (see Figure 1). As a result, closer tuples should be assigned higher cover values.

In contrast to its usual interpretation in terms of distance, we interpret the notion of a neighborhood without distance through the concept of *hypertuples* for both nominal and ordinal attributes. As a consequence, our novel distance measure applies to both ordinal and nominal attributes in a conceptually uniform way.

Incidentally, the NCM intrinsically handles missing values in a fashion consistent with other measures. Recall, that the NCM is a product of all N_i 's, where i is attribute index. For two data tuples, t and x , if there is a missing value in t or x for attribute i , then N_i is set to 1. As a result, this attribute does not contribute towards the NCM value.

5 Evaluation

The new Neighborhood Counting Metric is task-independent and can therefore be used for classification, clustering and other analytical tasks involving distances or similarities. We empirically evaluated the NCM in the context of a classification task, using the *k-nearest neighbor* (*k*-NN) classification algorithm with and without distance-based (neighbor

weighting [Baily and Jain, 1978]. The purpose of the evaluation is to compare the new measure with some of the commonly used distance measures in a setup involving *heterogeneous* data. The evaluation uses public benchmark data sets from UC Irvine Machine Learning Repository, which were selected with respect to their balance of ordinal and nominal attributes.

We implemented a *k*-NN algorithm with the novel NCM as well as the measures HEOM, HVDM, IVDM, and DVDVDM. The computational runtimes of the MRM turned out too excessive, so it was excluded from the study. In the experiments k was set to 1, 6, 11, 16, 21, 16, 31 and to 'MaxK' (i.e., k = the number of data tuples in the training data). We adopted a 10-fold cross-validation procedure, and for each measure and each k value we ran the cross-validation 10 times and recorded the results for subsequent analysis.

Due to space limitations, we report only some of the results in detail. Table 1 shows the results when weighting was used and $k = \text{MaxK}$.

A statistical significance analysis was carried out using the Student *t*-test (two samples, assuming unequal variances) with $\alpha = 0.05$ (at a 95% confidence level). For each data set, each measure and each k value, we ran a 10-fold cross-validation using *k*-NN 10 times with random partitioning of data, resulting in a sample of 10 values. For a pair of samples, by two different measures, we have a total of 20 values. So the critical value is 2.1. We then calculate the 't' values. If $t \geq 2.1$ or $t \leq -2.1$ the two samples are significantly different (the classification rate of one sample is significantly higher or lower than that of the other). Notice that every value in Table 1 is the average of a sample of 10 values.

From Table 1 we can see that based on our experimental design the NCM achieved 17 out of 20 'significantly' higher classification success rates compared to the other methods (last row in table). Looking at the subtotals for 'nominal', 'ordinal' and 'mixture' categories (reflected by subtotals from top to bottom), the NCM achieves the largest margin over its competitors for the 'nominal' data sets, followed by 'mixture' and then 'ordinal'. This suggests that, when all data tuples are taken into consideration, the NCM is clearly superior under all circumstances.

The details of the results involving other values for k are not shown. In terms of k -changes, we observe that without weighting there was no significant difference between the used measures. When weighting was used, there was still little difference for small k values. The general trend for all measures was: as k got larger the classification success rate increased slightly but soon started to decline. The difference then showed up as the NCM displayed a much slower rate of decline and, after $k > 11$, the NCM consistently outperformed the other four measures (see $k = \text{MaxK}$ as discussed

above). We can conclude that the NCM produces less variable or more robust results than the other four measures.

6 Conclusion

Starting from a probability function, we have developed a novel similarity or distance measure, the Neighborhood Counting Metric, which can be used with ordinal, nominal and heterogeneous attributes in a conceptually uniform way. This measure is defined without reference to any particular analytical task (e.g., classification). This means that the measure is potentially useful for many reasoning, inferential and reasoning tasks and systems modeling analogy or similarity. Because the NCM is based on a simple, easy-to-implement mathematical formulation and has a computational complexity in the same order as the Euclidean distance measure, it is a prime candidate for practical AI methods and tools.

Our empirical evaluation demonstrates that in a k -NN-based classification task, the measure significantly outperforms its competitors when distance-based neighbor weighting is used and when k is not too small, in particular when all data tuples are taken into consideration. As k gets larger, the NCM displayed a consistently superior performance over the reference methods. The difference is most significant when all data tuples in a training data set are considered. This implies that the NCM is less sensitive to k than the other measures considered in this study. Given an application based on the k -NN algorithm, if the optimal value for k is not known, we can simply consider all or a relatively large set of data tuples without a significantly compromising performance. Therefore, the performance of this measure is more predictable than the other methods investigated in this study.

In our experiments, all attributes were assumed to have equal weights. Future work will investigate how to determine the best attribute weights to achieve improved performance, as well as an application of the NCM to clustering tasks.

References

- [Ash, 1972] R.B. Ash. *Real Analysis and Probability*. Academic Press, New York, 1972.
- [Baily and Jain, 1978] T. Baily and A. K. Jain. A note on distance-weighted k -nearest neighbor rules. *IEEE Trans. Syst. Man Cyber.*, 8(4):311–313, 1978.
- [Blanzieri and Ricci, 1999] Enrico Blanzieri and Francesco Ricci. Probability based metrics for nearest neighbor classification and case-based reasoning. *Lecture Notes in Computer Science*, 1650:14–29, 1999.
- [Davies and Russell, 1987] Todd R. Davies and Stuart J. Russell. A logical approach to reasoning by analogy. In *Proc. of IJCAI87*, pages 264–270, Milan, Italy, 1987.

Data	DVDM	NCM	HEOM	IVDM	HVDM
Audiology (Standardized)	22.9	66.3	44.2	23.0	22.9
Bridges2 (Standardized)	44.3	60.8	52.9	44.3	44.3
Primary Tumor	25.6	44.9	26.0	25.6	25.6
Soybean (Large)	84.6	86.7	84.6	84.6	84.6
TTT	65.8	68.0	65.3	65.8	65.8
Vote	90.9	91.0	88.8	90.9	90.9
Yeast	32.3	33.5	32.0	33.0	32.0
Zoo	78.8	95.0	69.5	80.0	78.8
avg (nominal)	59.3	70.5	61.3	59.5	59.3
Diabetes	63.5	63.6	63.5	63.5	63.5
Ecoli	61.1	50.8	43.1	50.1	43.1
Glass	47.9	61.1	47.2	55.6	47.2
Iono	68.1	80.5	63.2	64.3	63.2
Iris	93.2	91.3	90.7	94.3	90.7
Pima	63.0	63.0	63.0	63.1	63.0
Sonar	74.3	85.7	73.1	71.6	73.1
Vehicle	46.3	60.1	42.3	48.2	42.3
Wine	94.7	96.7	94.6	98.1	94.6
avg (ordinal)	73.9	77.3	70.1	74.0	70.1
Anneal	75.3	75.3	75.3	75.3	75.3
Australian	85.5	87.5	82.8	85.3	82.8
Auto	49.1	67.9	46.4	48.2	46.4
Breast-Cancer	75.7	76.0	75.7	75.7	75.7
Bridges1	44.3	62.2	49.1	44.3	44.3
Credit Screening	60.8	88.3	82.0	64.7	59.5
German	68.7	69.0	68.7	68.7	68.7
Heart	79.2	82.3	80.6	81.7	72.0
Hepatitis	76.6	81.7	76.6	76.6	76.6
Hors-Colic	77.9	84.7	64.4	79.4	64.4
avg (heterogeneous)	69.3	77.5	70.2	70.0	66.6
Sig. count	1	17	0	2	0

Table 1: Average results (classification success rates) over 10 runs of the algorithms, where neighbor weighting was used and $k = \text{MaxK}$. Best values are shown in bold, and statistically significant values in bold italic.

- [Kolodner, 1993] Janet Kolodner. *Case-Based Reasoning*. Morgan Kaufmann Publishers, San Mateo, CA, 1993.
- [Osborne and Bridge, 1997] Hugh Osborne and Derek Bridge. Models of similarity for case-based reasoning. In *Procs. of the Interdisciplinary Workshop on Similarity and Categorisation*, pages 173–179, 1997.
- [Russell, 1986] Stuart J. Russell. Quantitative analysis of analogy. In *Proc of AAAI86*, pages 284–288, Philadelphia, PA, 1986. Morgan Kaufmann.
- [Stanfill and Waltz, 1986] C. Stanfill and D. Waltz. Toward memory-based reasoning. *Communication of ACM*, 29:1213–1229, 1986.
- [Vosniadou and Ortony, 1989] Stella Vosniadou and Andrew Ortony, editors. *Similarity and Analogical Reasoning*. Cambridge University Press, New York, USA, 1989.
- [Wang et al., 1999] H. Wang, W. Dubitzky, I. Düntsch, and D. Bell. A lattice machine approach to automated case-base design: Marrying lazy and eager learning. In *Proc. IJCAI99*, pages 254–259, Stockholm, Sweden, 1999.
- [Wilson and Martinez, 1997] D. Randal Wilson and Tony R. Martinez. Improved heterogeneous distance functions. *Journal of Artificial Intelligence Research*, 6:1–34, 1997.

CONSTRAINT SATISFACTION AND SEARCH

Propagating Logical Combinations of Constraints

Fahiem Bacchus*

University of Toronto
Toronto, Canada
fbacchus@cs.toronto.edu

Toby Walsh†

NICTA and UNSW
Sydney, Australia
tw@cse.unsw.edu.au

Abstract

Many constraint toolkits provide logical connectives like disjunction, negation and implication. These permit complex constraint expressions to be built from primitive constraints. However, the propagation of such complex constraint expressions is typically limited. We therefore present a simple and light weight method for propagating complex constraint expressions. We provide a precise characterization of when this method enforces generalized arc-consistency. In addition, we demonstrate that with our method many different global constraints can be easily implemented.

1 Introduction

Real world problems often contain logical combinations of numerical and symbolic constraints. For example, many configuration problems are naturally specified using implication and other logical connectives (e.g. if the car is a coupe or a convertible then a roof rack is not a valid option). To facilitate the modeling of such problems, constraint toolkits allow logical combinations of primitive constraints to be posted. However, such combinations are typically not propagated very effectively. For example, the propagation of a disjunction of constraints is generally delayed until all but one of the disjuncts are falsified after which the remaining disjunct (which must now hold) is propagated.

We will present a simple and light weight method to propagate constraint expressions built up from logical connectives and primitive constraints. This method can be incorporated into any current constraint toolkit by a simple extension to the propagators for the primitive constraints. We can therefore provide the user with a rich language for specifying problems, whilst preserving the ability to prune the search space. To demonstrate the usefulness of such a facility, we show that many global constraints can be easily specified and, in some cases, effectively propagated, using simple constraint expressions. Thus our method can often provide the toolkit user with a very low cost alternative to the enterprise of designing and implementing special purpose propagators for unusual global constraints that might appear in their problem.

*This research funded by NSERC Canada.

†This research funded in part by National ICT Australia.

2 Inconsistent and Valid Assignments

A *constraint satisfaction problem* consists of a set of variables, each with a domain of values, and a set of constraints. Each constraint consists of a scope of variables to which it is applied, and a relation of allowed values for those variables. For convenience, we represent the domains $\mathcal{D}$ of the variables by the set of possible assignments. For example, if we have two 0/1 variables, X and Y then $\mathcal{D} = \{X = 0, X = 1, Y = 0, Y = 1\}$. We let $\text{domain}(X)$ be the set of values in the domain of the variable X : $\text{domain}(X) = \{a \mid X = a \in \mathcal{D}\}$. An **assignment set** τ is a set of assignments to variables such that no variable is assigned more than one value by τ . The **scope** of an assignment set τ (constraint C) is the set of variables in τ (C) and is denoted by $\text{scope}(\tau)$ ($\text{scope}(C)$). Given a constraint C and an assignment set τ with $\text{scope}(C) \subseteq \text{scope}(\tau)$, we write $C(\tau)$ iff the assignments in τ satisfies C . That is $C(\tau)$ iff there exists $X_1 = a_1, \dots, X_k = a_k \in \tau$ with $\text{scope}(C) = \{X_1, \dots, X_k\}$ and $X_1 = a_1, \dots, X_k = a_k$ satisfies C . We write $\neg C(\tau)$ otherwise.

An assignment is (generalized arc) **inconsistent** for a constraint iff no assignment set containing it satisfies the constraint. That is, $X = a$ is inconsistent for C iff $\forall \tau. (\text{scope}(C) \subseteq \text{scope}(\tau) \wedge X = a \in \tau) \rightarrow \neg C(\tau)$. A constraint C has a unique maximal set of inconsistent assignments $\text{MaxInc}(C)$. For example, given the constraint $X < Y$ with $X = \{0, 1, 2\}$ and $Y = \{1, 2\}$, then $\{X = 2\}$ is the maximal set of inconsistent assignments. Assignments that are **consistent** have at least one witness falsifying the above condition; i.e., $X = a$ is consistent iff there is an assignment set τ (called a **support**) with $\text{scope}(\tau) = \text{scope}(C) \wedge X = a \in \tau \wedge C(\tau)$. A constraint C is **GAC** (Generalized Arc Consistent) iff every value of every variable in $\text{scope}(C)$ has at least one support. If $X = a$ is inconsistent, we can prune a from the domain of X . A constraint C can be made GAC by simply pruning all values in $\text{MaxInc}(C)$ from the domains of their respective variables.

An essential notion for our approach is the concept of a **valid assignment**. Valid assignments are the dual of inconsistent assignments. An assignment is **valid** for a constraint iff all assignment sets containing it satisfy the constraint. That is, $X = a$ is valid for C iff $\forall \tau. (\text{scope}(C) \subseteq \text{scope}(\tau) \wedge X = a \in \tau) \rightarrow C(\tau)$. As with inconsistent values every constraint C has a unique maximal set of valid assignments, $\text{MaxValid}(C)$. For example, given the constraint $X < Y$

with $X = \{0, 1, 2\}$ and $Y = \{1, 2\}$, the maximal set of valid assignments is $\{X = 0\}$. All possible extensions of $X = 0$ satisfy the constraint $X < Y$, but all other assignments to X can be extended so that they fail to satisfy the constraint.

All of the concepts presented for inconsistent assignments have dual versions for valid assignments. For example, the dual of consistent assignments is the notion of non-valid assignments. An assignment $X = a$ is **non-valid** if there is at least one assignment set τ with $scope(\tau) = scope(C) \wedge X = a \in \tau \wedge \neg C(\tau)$. Another example is that the dual of a support is a non-support. An assignment set τ is a **non-support** for an assignment $X = a$ in a constraint C iff $scope(\tau) = scope(C) \wedge X = a \in \tau \wedge \neg C(\tau)$. A non-support witnesses the non-validity of $X = a$.

3 Constraint Expressions

To build complex constraints, we combine primitive constraints using negation, disjunction and conjunction. A **constraint expression** is either a primitive constraint C or any well-founded Boolean expression of the form: *true*, *false*, *not*(C_1), *or*($C_1, \dots, C_k$) or *and*($C_1, \dots, C_k$), where each C_i is itself a constraint expression. *true* is the primitive constraint which is always valid, whilst *false* is always inconsistent. We also allow the expressions *implies*(C_1, C_2), *iff*(C_1, C_2), *xor*(C_1, C_2) and *ifthen*(C_1, C_2, C_3), but regard these additional connectives to be abbreviations:

$$\begin{aligned} \text{implies}(C_1, C_2) &\leftrightarrow \text{or}(\text{not}(C_1), C_2) \\ \text{iff}(C_1, C_2) &\leftrightarrow \text{and}(\text{or}(\text{not}(C_1), C_2), \text{or}(\text{not}(C_2), C_1)) \\ \text{xor}(C_1, C_2) &\leftrightarrow \text{and}(\text{or}(C_1, C_2), \text{or}(\text{not}(C_1), \text{not}(C_2))) \\ \text{ifthen}(C_1, C_2, C_3) &\leftrightarrow \text{and}(\text{or}(\text{not}(C_1), C_2), \text{or}(C_1, C_3)) \end{aligned}$$

Each constraint expression C represents a new constraint whose scope is equal to the union of the scopes of the primitive constraints in C . An assignment set τ satisfies C iff $scope(C) \subseteq scope(\tau)$ and the Boolean expression representing C evaluates to true given the truth values of the component primitive constraints under τ . For example, an absolute value constraint $X = \text{abs}(Y)$ can be written as the constraint expression *ifthen*($Y \geq 0, X = Y, X = -Y$). Similarly, a max constraint $X = \text{max}(Y, Z)$ can be written as the constraint expression *and*($X \geq Y, X \geq Z, \text{or}(X = Y, X = Z)$).

Since a constraint expression is itself a constraint, associated with every constraint expression C is a maximal set of inconsistent assignments, $\text{MaxInc}(C)$, and a maximal set of valid assignments $\text{MaxValid}(C)$. We can make the constraint expression C generalized arc consistent by pruning all assignments in $\text{MaxInc}(C)$. It is also useful to observe that the duality between valid assignments and inconsistent assignments implies $\text{MaxInc}(C) = \text{MaxValid}(\text{not}(C))$: $X=a \in \text{MaxInc}(C) \leftrightarrow (\forall \tau. X=a \in \tau \rightarrow (\neg C(\tau) \leftrightarrow \text{not}(C)(\tau))) \leftrightarrow X=a \in \text{MaxValid}(\text{not}(C))$.

4 Constraint Propagation

Not surprisingly, it is NP-hard to compute MaxInc for an arbitrary constraint expression. More precisely, deciding if a set of assignments is the maximal inconsistent set for an arbitrary

$\text{Inc}(\text{not}(C_1), \mathcal{D})$	$= \text{Valid}(C_1, \mathcal{D})$
$\text{Valid}(\text{not}(C_1), \mathcal{D})$	$= \text{Inc}(C_1, \mathcal{D})$
$\text{Inc}(\text{or}(C_1, \dots, C_k), \mathcal{D})$	$= \bigcap_i \text{Inc}(C_i, \mathcal{D})$
$\text{Valid}(\text{and}(C_1, \dots, C_k), \mathcal{D})$	$= \bigcap_i \text{Valid}(C_i, \mathcal{D})$
$\text{Inc}(\text{and}(C_1, \dots, C_k), \mathcal{D})$	$= \text{ItInc}(\text{and}(C_1, \dots, C_k), \mathcal{D})$
$\text{Valid}(\text{or}(C_1, \dots, C_k), \mathcal{D})$	$= \text{ItValid}(\text{or}(C_1, \dots, C_k), \mathcal{D})$

$\text{ItInc}(\text{and}(C_1, \dots, C_k), \mathcal{D})$	$\text{ItValid}(\text{or}(C_1, \dots, C_k), \mathcal{D})$
$\text{inc} := \emptyset$	$\text{valid} := \emptyset$
repeat	repeat
$\text{inc}' := \bigcup_i \text{Inc}(C_i, \mathcal{D})$	$\text{valid}' := \bigcup_i \text{Valid}(C_i, \mathcal{D})$
$\mathcal{D} := \mathcal{D} - \text{inc}'$	$\mathcal{D} := \mathcal{D} - \text{valid}'$
$\text{inc} := \text{inc} \cup \text{inc}'$	$\text{valid} := \text{valid} \cup \text{valid}'$
until ($\text{inc}' = \emptyset$)	until ($\text{valid}' = \emptyset$)
return (inc)	return (valid)

Table 1: Functions for computing valid and inconsistent assignments of a constraint expression. In addition, $\text{Inc}(\text{true}, \mathcal{D}) = \text{Valid}(\text{false}, \mathcal{D}) = \emptyset$, and $\text{Valid}(\text{true}, \mathcal{D}) = \text{Inc}(\text{false}, \mathcal{D}) = \mathcal{D}$.

constraint expression is D^P -complete in general.¹ There are results on tractable languages that identify tractable cases for computing MaxInc and MaxValid . However, we are interested here in computing inconsistent assignments for arbitrary constraint expressions, which is intractable in general.

We propose a simple and light weight method for computing subsets of MaxInc and MaxValid in polynomial time. The method is compositional, computing the inconsistent and valid assignments of a constraint expression from the inconsistent and valid assignments of its parts. For a constraint expression C and variable domains $\mathcal{D}$, the functions $\text{Inc}(C, \mathcal{D})$ and $\text{Valid}(C, \mathcal{D})$ return subsets of $\text{MaxInc}(C)$ and $\text{MaxValid}(C)$ respectively. These functions recursively apply the rules in Table 1, until they reach the primitive constraints. We assume that each primitive constraint has a poly-time algorithm to compute inconsistent and valid assignments.

The algorithm can be optimized by a simple caching scheme in which we remember the previously computed value $\text{Inc}(C_i, \mathcal{D})$ for each subexpression C_i . If in a subsequent call $\text{Inc}(C_i, \mathcal{D}')$, $\mathcal{D}'$ is identical to $\mathcal{D}$ when restricted to the variables in $scope(C_i)$, then we can reuse the previously computed result for $\text{Inc}(C_i, \mathcal{D})$. A similar optimization works for $\text{Valid}(C_i, \mathcal{D})$. In addition, if we compute and prune inconsistent values incrementally, we can stop as soon as any variable has a domain wipeout.

4.1 Entailment and Disentailment

A constraint expression is **entailed** iff it holds for all possible assignments. A constraint is **disentailed** iff it does not hold for any possible assignment. As we show below, $\text{Valid}(C, \mathcal{D})$ returns only valid assignments. Hence, if $\text{Valid}(C, \mathcal{D})$ equals the domains of all of the variables in the scope of C , C must be entailed. In such a situation, we modify the computation of Valid so that $\text{Valid}(C, \mathcal{D}) = \mathcal{D}$ (note that $\mathcal{D}$ might include domains of other variables besides those in the $scope(C)$). Similarly, if $\text{Inc}(C, \mathcal{D})$ equals the domains of all of the vari-

¹The complexity class D^P contains problems which are the conjunction of a problem in NP and one in coNP [Papadimitriou and Yannakakis, 1984].

ables in the scope of $\mathcal{C}$ then $\mathcal{C}$ is disentailed; and we modify the computation of Inc so that $Inc(\mathcal{C}, \mathcal{D}) = \mathcal{D}$.

To show the benefit, consider $implies(even(X), odd(Y))$ with $\mathcal{D} = \{X=0, X=2, Y=1, Y=2\}$. Now $Inc(implies(even(X), odd(Y)), \mathcal{D}) = Valid(even(X), \mathcal{D}) \cap Inc(odd(Y), \mathcal{D})$. Using the unmodified versions, $Valid(even(X), \mathcal{D})$ will return just valid values for X , whilst $Inc(odd(Y), \mathcal{D})$ returns just inconsistent values for Y . Inc would then compute the empty set of inconsistent assignments for $implies(even(X), odd(Y))$. Note, however, that $domain(X)$ only contains even numbers. Hence $even(X)$ is entailed. Therefore the modified $Valid(even(X), \mathcal{D})$ can return $\mathcal{D}$, in which case $Inc(implies(even(X), odd(Y)), \mathcal{D}) = \mathcal{D} \cap Inc(odd(Y), \mathcal{D}) = \{Y=2\}$. As required, this is the maximal set of inconsistent assignments.

4.2 Correctness

We prove that Inc and $Valid$ are correct. That is, they only return inconsistent and valid assignments respectively.

Theorem 1 *Inc and Valid are correct assuming that inconsistent and valid assignments are correctly computed for the constituent primitive constraints.*

Proof: By induction on the structure of the constraint expression. The base case holds by assumption. The step case uses case analysis.

For a constraint expression $not(\mathcal{C}_1, \mathcal{D})$, we have that $Inc(not(\mathcal{C}_1), \mathcal{D}) = Valid(\mathcal{C}_1, \mathcal{D})$. By induction, the assignments in $Valid(\mathcal{C}_1, \mathcal{D})$ are valid. Hence all of these assignments are inconsistent for $not(\mathcal{C}_1, \mathcal{D})$. A dual argument shows that the assignments in $Valid(not(\mathcal{C}_1), \mathcal{D})$ are valid.

For a constraint expression $\mathcal{C} = and(\mathcal{C}_1, \dots, \mathcal{C}_k)$, we have $Valid(\mathcal{C}, \mathcal{D}) = \bigcap_i Valid(\mathcal{C}_i, \mathcal{D})$. Suppose $X = a \in Valid(\mathcal{C}, \mathcal{D})$. Then for all i , $X = a \in Valid(\mathcal{C}_i, \mathcal{D})$. By the induction hypothesis, the assignments in each $Valid(\mathcal{C}_i)$ are valid. Consider any assignment set τ such that $X = a \in \tau$ and $scope(\mathcal{C}) \subseteq scope(\tau)$. Since $X = a$ is valid for each $\mathcal{C}_i$, τ must satisfy all $\mathcal{C}_i$ and thus must satisfy the conjunction. Hence $X = a$ is also valid for $\mathcal{C}$. For $\mathcal{C} = Inc(and(\mathcal{C}_1, \dots, \mathcal{C}_k), \mathcal{D})$ a similar argument shows that the assignments in $\bigcup_i Inc(\mathcal{C}_i, \mathcal{D})$ are inconsistent. Deleting these assignments from $\mathcal{D}$ cannot cause any consistent assignment to lose its support, hence $Inc(\mathcal{C}_i, \mathcal{D}')$ on the reduced domain $\mathcal{D}'$ must still return inconsistent assignments. $ItInc$ then recomputes $\bigcup_i Inc(\mathcal{C}_i, \mathcal{D})$ until we reach a fixed point.

Similar arguments hold for constraint expressions of the form $or(\mathcal{C}_1, \dots, \mathcal{C}_k)$. $\square$

4.3 Termination

The $ItInc$ and $ItValid$ functions only require a linear (in the size of the CSP) number of iterations to reach their fixed point in the worst case.

Theorem 2 *ItInc and ItValid take $O(nd)$ iterations to reach their fixed points for a constraint expression with n variables and domains of size d . There exist constraint expressions which take $\Theta(nd)$ iterations to reach the fixed point.*

Proof: As each iteration removes at least one value, we must reach the fixed point in at most nd steps. We can give

a simple example in which this bound is reached. Consider $and(\mathcal{C}_1, \dots, \mathcal{C}_n)$ where $\mathcal{C}_i$ is $X_i = X_{i+1}$ for $i < n$ and $X_1 - X_n = 1$ for $i = n$. Suppose $domain(X_i) = \{1, \dots, d\}$ for every i . Then in the first iteration, $ItInc$ returns $\{X_1 = 1\}$ as this value is not supported in $X_1 - X_n = 1$. After this is pruned from $\mathcal{D}$, a second iteration returns $\{X_2 = 1\}$ as this value is now not supported in $X_1 = X_2$. And so on up to the n th iteration which returns $\{X_n = 1\}$. After this is pruned, the $n + 1$ th iteration returns $\{X_1 = 2\}$ as this value is now not supported in $X_1 - X_n = 1$. Hence, there are nd iterations before all the values of all the variables are removed. Note that even if we stop when the first variable has a domain wipeout, it will still take $(n - 1)d + 1$ iterations. $\square$

4.4 Maximality

These functions do not always compute maximal sets, even if maximal sets are computed for the primitive constraints from which they are composed. This is not surprising given that computing maximal sets for an arbitrary constraint expression is intractable in general. The following result precisely characterizes when Inc returns the maximal inconsistent set of assignments. In other words, the following result identifies exactly when pruning the values returned by Inc ensures that a constraint expression is GAC.

We start with a number of definitions. A *hypergraph* $\mathcal{H} = (H, E^H)$ is a set of vertices H and hyperedges E^H each of which is a subset of H . A hypergraph $\mathcal{H}$ has an **acyclic tree decomposition** [Flum et al., 2002] iff there is a tree T satisfying: (1) there is a one-to-one correspondence between the hyperedges of $\mathcal{H}$ and the nodes of T ; the hyperedge corresponding to a tree node t is called t 's label ($label(t)$); (2) for every vertex $v \in \mathcal{H}$ the set of nodes t of T such that $v \in label(t)$ form a subtree of T . The hypergraph of a conjunctive or disjunctive constraint expression, $\mathcal{C} = and(\mathcal{C}_1, \dots, \mathcal{C}_k)$ or $\mathcal{C} = or(\mathcal{C}_1, \dots, \mathcal{C}_k)$, has the variables in $scope(\mathcal{C})$ as vertices and the sets of variables $scope(\mathcal{C}_i)$, $i = 1, \dots, k$ as hyperedges. We will relax this definition to take account of (dis)entailment. If $\mathcal{C} = and(\mathcal{C}_1, \dots, \mathcal{C}_k)$ then we ignore any entailed subexpression when constructing the hypergraph. Similarly, if $\mathcal{C} = or(\mathcal{C}_1, \dots, \mathcal{C}_k)$ then we ignore any disentailed subexpression. Under this relaxation we define a conjunctive or disjunctive constraint expression to be **acyclic** if its corresponding hypergraph has an acyclic tree decomposition. For example, a conjunction in which the primitive constraints are in a chain, and each has only one variable in common with the previous and next constraint is acyclic. Acyclicity is, however, more general than being a chain. We will use acyclicity to characterize when Inc computes $MaxInc$.

Theorem 3 *For any constraint expression $\mathcal{C}$ and any variable domains $\mathcal{D}$, $Inc(\mathcal{C}, \mathcal{D}) = MaxInc(\mathcal{C}, \mathcal{D})$ if:*

1. $\mathcal{C}$ is a primitive constraint and $Inc(\mathcal{C}, \mathcal{D}) = MaxInc(\mathcal{C}, \mathcal{D})$;
2. $\mathcal{C} = not(\mathcal{C}_1)$ and $Valid(\mathcal{C}_1, \mathcal{D}) = MaxValid(\mathcal{C}_1, \mathcal{D})$;
3. $\mathcal{C} = or(\mathcal{C}_1, \dots, \mathcal{C}_k)$ and $Inc(\mathcal{C}_i, \mathcal{D}) = MaxInc(\mathcal{C}_i, \mathcal{D})$ for $i \in (1, \dots, k)$;
4. $\mathcal{C} = and(\mathcal{C}_1, \dots, \mathcal{C}_k)$ and (a) $Inc(\mathcal{C}_i, \mathcal{D}) = MaxInc(\mathcal{C}_i, \mathcal{D})$ for $i \in (1, \dots, k)$; (b) $\mathcal{C}$ is acyclic; and (c) $|scope(\mathcal{C}_i) \cap scope(\mathcal{C}_j)| \leq 1$ for $i, j \in (1, \dots, k)$;

5. $Inc(C, \mathcal{D}) = \mathcal{D}$.

Proof: 1. Immediate.

2. Suppose $Valid(\mathcal{C}_1, \mathcal{D})$ is maximal. Then for any $X = a \notin Valid(\mathcal{C}_1, \mathcal{D})$, there exists τ with $X = a \in \tau$ and $\neg \mathcal{C}_1(\tau)$. Hence $X = a$ cannot be in $Inc(not(\mathcal{C}_1), \mathcal{D})$ as τ is one assignment that prevents it being inconsistent. Hence $Inc(not(\mathcal{C}_1), \mathcal{D})$ is maximal.

3. Suppose $Inc(\mathcal{C}_i, \mathcal{D})$ are maximal. Consider $X = a \notin Inc(or(\mathcal{C}_1, \dots, \mathcal{C}_k), \mathcal{D})$. Then $X = a \notin \bigcap_{1 \leq i \leq k} Inc(\mathcal{C}_i, \mathcal{D})$. That is, $X = a \notin Inc(\mathcal{C}_j, \mathcal{D})$ for some $j \in (1, \dots, k)$. As $Inc(\mathcal{C}_j, \mathcal{D})$ is maximal, there exists τ with $X = a \in \tau$ and $\mathcal{C}_j(\tau)$. Thus $X = a$ cannot be in $Inc(or(\mathcal{C}_1, \dots, \mathcal{C}_k), \mathcal{D})$ as τ is one assignment that prevents it being inconsistent. Hence $Inc(or(\mathcal{C}_1, \dots, \mathcal{C}_k), \mathcal{D})$ is maximal.

4. Suppose $X \in scope(\mathcal{C})$ with $X = a \in \mathcal{D}$, but $X = a \notin Inc(\mathcal{C}, \mathcal{D})$. We must show that $X = a \notin MaxInc(\mathcal{C}, \mathcal{D})$. Let $\mathcal{D}^c = \mathcal{D} - Inc(\mathcal{C}, \mathcal{D})$, i.e., the consistent assignments remaining in the variable domains. From Table 1 we observe that $Inc(\mathcal{C}_i, \mathcal{D}^c) = \emptyset$ and by condition (a) $MaxInc(\mathcal{C}_i, \mathcal{D}^c) = \emptyset$ for all $i \in (1, \dots, k)$. Consider the acyclic tree decomposition associated with $\mathcal{C}$. Orient this tree so that the root is labeled with $scope(\mathcal{C}_i)$ for some $\mathcal{C}_i$ with $X \in scope(\mathcal{C}_i)$. Note that by property (2) of an acyclic tree decomposition and condition (c), each of the subtrees below $\mathcal{C}_i$ can have at most one variable in common with the other subtrees. Furthermore if two subtrees do have a variable in common that variable must be in the $scope(\mathcal{C}_i)$.

Since $X = a \in \mathcal{D}^c$ it must have some support τ on $\mathcal{C}_i$ such that $\tau \in \mathcal{D}^c$. Now we extend this support downwards in the tree decomposition to the children of $\mathcal{C}_i$: $\mathcal{C}_j^1, \dots, \mathcal{C}_j^\ell$. Each such child $\mathcal{C}_j$ shares only one variable with $\mathcal{C}_i$, say Y , and Y must be assigned some value in τ , say $Y = b$. Since $Y = b \in \mathcal{D}^c$ it must have a support τ_j in $\mathcal{C}_j$ such that $\tau_j \in \mathcal{D}^c$. Thus we can extend τ to a support for $and(\mathcal{C}_i, \mathcal{C}_j)$ for each child of $\mathcal{C}_i$. Furthermore the supports τ_j for the individual children of $\mathcal{C}_i$ cannot be in conflict: $\mathcal{C}_j$ and $\mathcal{C}_{j'}$ can only share a variable already assigned by τ , hence τ_j and $\tau_{j'}$ must agree with τ and with each other on the value assigned to this variable. Thus we can extend τ to a support for all of $\mathcal{C}_i$'s children. Furthermore, by the same argument each support τ_j for the child $\mathcal{C}_j$ can be extended to a support for all of the conjuncts in the subtree below $\mathcal{C}_j$. Hence, τ can be extended to a support for all of $\mathcal{C}$, and since $X = a \in \tau$, $X = a \notin MaxInc(\mathcal{C}, \mathcal{D})$.

Note that if $and(\mathcal{C}_1, \dots, \mathcal{C}_k)$ contains any entailed conjuncts, these can be eliminated without changing the maximal set of inconsistent assignments. We can then apply the argument above to the remaining acyclic part of the conjunction.

5. Immediate since $MaxInc(\mathcal{C}, \mathcal{D}) \subseteq \mathcal{D}$. $\square$

In fact, we can show that these five cases are the only ones in which Inc is always guaranteed to be maximal. This reverse direction needs a little care as Inc may compute $MaxInc$ by chance. However, these five cases are the only ones in which, irrespective of the constraint subexpressions, Inc is guaranteed to compute $MaxInc$. A dual result holds, and characterizes precisely when $Valid$ computes $MaxValid$.

Previous related results have shown that acyclic database queries of bounded tree width are tractable. Bounded tree width (tree width one) is also known to be the condition

that characterizes when arc consistency achieves GAC on conjunctions of binary constraints. The main difference between our result and these previous results is that we do not place any restriction on the arity of the primitive constraints. Hence, our result does not depend on bounded tree width (the tree width of the constraint graph is at least as large as the arity of the primitive constraints). Given primitive constraints for which GAC can be efficiently computed (no matter their arity), our result characterizes when this efficiency can be lifted to complex combinations of these constraints.

5 Some Applications

To demonstrate the usefulness of constraint expressions and our propagation algorithm, we show that a wide range of global and other constraints can be specified and propagated using simple constraint expressions. In many cases, our light weight propagation algorithm is able to achieve GAC. Our method is then an effective and low cost means to implement these constraints. This is especially valuable when the constraints useful for a particular problem are too specialized to be in the user's constraint toolkit.

DOMAIN constraint. This channels between a variable and a sequence of 0/1 variables representing the possible values taken by the variable. More precisely, $DOMAIN(X, [X_1, \dots, X_n])$ ensures $X = i$ iff $X_i = 1$ [Refalo, 2000]. We can decompose this into a disjunction:

$$or(and(X = 1, X_1 = 1, \dots, X_n = 0), \dots, \\ and(X = n, X_1 = 0, \dots, X_n = 1))$$

The equality constraints in each conjunct again have no variables in common. Hence pruning the values returned by Inc enforces GAC.

ELEMENT constraint. This indexes into an array with a variable [Van Hentenryck and Carillon, 1988]. More precisely, $ELEMENT(I, [X_1, \dots, X_n], J)$ holds iff $X_I = J$. We can use this to look up the price of a particular component included in a configuration problem. It decomposes into a simple disjunction:

$$or(and(I = 1, J = X_1), \dots, and(I = n, J = X_n))$$

The equality constraints within each conjunct share no variables. Hence pruning the values returned by Inc enforces GAC.

MEMBER constraint. This ensures that a particular value is used. More precisely, $MEMBER(I, [X_1, \dots, X_n])$ holds iff there exists j with $X_j = I$. We can decompose this into a disjunction:

$$or(X_1 = I, \dots, X_n = I)$$

Pruning the values returned by Inc enforces GAC.

MAX constraint. This computes the maximum value taken by a sequence of variables. More precisely, $MAX(N, [X_1, \dots, X_n])$ holds iff there exists i with $N = X_i$ and $X_i \geq X_j$ for all j . A MAX constraint can be decomposed into a disjunction:

$$or(and(N = X_1, X_1 \geq X_2, \dots, X_1 \geq X_n) \\ and(N = X_2, X_2 \geq X_1, X_2 \geq X_3, \dots, X_2 \geq X_n) \\ \dots, and(N = X_n, X_n \geq X_1, \dots, X_n \geq X_{n-1}))$$

Pruning the values returned by *Inc* on such a decomposition enforces GAC.

CHANNELLING constraints. It is often advantageous to specify multiple viewpoints of a problem. Constraints may be easier to specify in one viewpoint and propagate better in another. Channelling constraints are then needed to link the different viewpoints. For example, multiple viewpoints have proved useful in the orchestra rehearsal problem, `prob039` in `CSPLIB`. The channelling constraints used here can be specified as constraint expressions:

$$\begin{aligned} & \text{iff}(X_i = 1, \text{or}(X_{i-1} = 1, Y_i = 1)) \\ & \text{iff}(Z_k = j, \text{and}(X_j = 1, X_{j-1} = 0)) \end{aligned}$$

The problem also contained a specialized optimality constraint which can be specified as the constraint expression:

$$\text{implies}(X < Y, W_X = 1)$$

Where $W_X = 1$ can itself be specified with an `ELEMENT` constraint.

LEX constraint. This lexicographically orders two vectors of variables. It has many applications when dealing with symmetry [Frisch *et al.*, 2002]. We can decompose it into a disjunction:

$$\begin{aligned} & \text{or}(X_1 < Y_1, \\ & \quad \text{and}(X_1 = Y_1, X_2 < Y_2), \\ & \quad \text{and}(X_1 = Y_1, X_2 = Y_2, X_3 < Y_3), \dots, \\ & \quad \text{and}(X_1 = Y_1, \dots, X_{n-1} = Y_{n-1}, X_n \leq Y_n)) \end{aligned}$$

As the constraints in each conjunct share no variables, pruning the values returned by *Inc* enforces GAC on the `LEX` constraint. In addition, if we memoize previous results, *Inc* will be comparable to the efficient algorithm in [Frisch *et al.*, 2002].

The choice of decomposition of a global constraint is important for ensuring as much propagation as possible. For instance, an alternative decomposition is the conjunction:

$$\begin{aligned} & \text{and}(X_1 \leq Y_1, \\ & \quad \text{implies}(X_1 = Y_1, X_2 \leq Y_2), \dots \\ & \quad \text{implies}(\text{and}(X_1 = Y_1, \dots, X_{n-1} = Y_{n-1}), X_n \leq Y_n)) \end{aligned}$$

Pruning the values returned by *Inc* on this decomposition may not enforce GAC on the `LEX` constraint. Consider X_1 and $Y_1 \in \{0, 1\}$, $X_2 = 1$ and $Y_2 = 0$. Then $X_1 = 1$ and $Y_1 = 0$ are both inconsistent assignments. However, *Inc* applied to this decomposition will return the empty set.

VALUE PRECEDENCE constraint. This breaks symmetries in a sequence of variables, X_1 to X_n caused by two indistinguishable values s and t [Law and Lee, 2004]. We can decompose it into a simple conjunction:

$$\begin{aligned} & \text{and}(X_1 \neq t, \text{implies}(X_2 = t, X_1 = s), \dots, \\ & \quad \text{implies}(X_n = t, \text{or}(X_1 = s, \dots, X_{n-1} = s))) \end{aligned}$$

Theorem 2 of [Law and Lee, 2004] shows that enforcing GAC on each conjunct achieves GAC on the global constraint. As each conjunct expands out into a simple disjunction, pruning the values returned by *Inc* enforces GAC on the global constraint.

AMONG constraint. This limits the number of variables taking values from a set [Beldiceanu and Contejean, 1994]. More precisely, `AMONG` ($[X_1, \dots, X_n], [d_1, \dots, d_m], N$) holds iff $N = |\{i | X_i = d_j\}|$. The constraint is useful in many resource allocation and scheduling problems. By introducing additional variables N_i in which to accumulate a count, we can decompose this into a relatively simple constraint expression:

$$\begin{aligned} & \text{and}(\text{ifthen}(X_1 \in D, N_1 = 1, N_1 = 0), \\ & \quad \text{ifthen}(X_2 \in D, N_2 = N_1 + 1, N_2 = N_1), \\ & \quad \dots, \text{ifthen}(X_n \in D, N = N_{n-1} + 1, N = N_{n-1})), \end{aligned}$$

where $X_i \in D$ is an abbreviation for $\text{or}(X_i = d_1, \dots, X_i = d_m)$.

Unfortunately, *Inc* does not achieve GAC on such a decomposition. Consider, for example, $X_1, X_2 \in \{0, 1\}$, $N \in \{0, 1, 2\}$ and $N_1 \in \{0, 1\}$. Enforcing GAC on `AMONG` ($[X_1, X_2], [0, 1], N$) will prune 0 and 1 from N whilst *Inc* returns the empty set. Similar decompositions and results hold for the `ATMOST`, `ATLEAST`, `CHANGE`, and `COUNT` constraints.

NOTALLEQUAL constraint. This ensures that not all variables take the same value [Beldiceanu and Contejean, 1994]. More precisely, `NOTALLEQUAL` ($[X_1, \dots, X_n]$) holds iff there exists i and j with $X_i \neq X_j$. This can be decomposed into the following constraint expression:

$$\text{or}(X_1 \neq X_2, \dots, X_1 \neq X_n)$$

Pruning the values returned by *Inc* enforces GAC.

NVALUE constraint. This counts the number of values used by a sequence of variables [Pachet and Roy, 1999]. More precisely, `NVALUE` ($[X_1, \dots, X_n], N$) holds iff $|\{X_i | 1 \leq i \leq n\}| = N$. The constraint is useful in a wide range of problems involving resources. The `ALLDIFFERENT` constraint is a special case of the `NVALUE` constraint in which $N = n$. Unfortunately, it is NP-hard in general to enforce GAC on a `NVALUE` constraint [Bessiere *et al.*, 2004].

One way to propagate this constraint in polynomial time is to decompose it with an additional set of 0/1 variables representing the characteristic function of the set of values used:

$$\begin{aligned} & \text{and}(\text{iff}(S_1 = 1, \text{or}(X_1 = 1, \dots, X_n = 1)), \\ & \quad \dots, \text{iff}(S_m = 1, \text{or}(X_1 = m, \dots, X_n = m))), \\ & \quad \sum_{i=1}^m S_i = N) \end{aligned}$$

Where the domains of the X_i are assumed to be from 1 to m . Not surprisingly, *Inc* does not compute the maximal set of inconsistent assignments for this constraint expression. For example, suppose $X_1, X_2 \in \{1, 2\}$, $X_3 \in \{1, 2, 3\}$, $N = 3$ and $S_i = 1$ for $1 \leq i \leq 3$. Then *Inc* returns the empty set, even though $X_3 = 1$ and $X_3 = 2$ are inconsistent.

A similar decomposition and result holds for the `COMMON` constraint, which is also NP-hard to propagate.

6 Related Work

Lhomme has proposed GAC algorithms for logical combinations of primitive constraints, focused primarily on those that are given in extension [Lhomme, 2004]. For example, on a disjunction like $\text{or}(C_1, \dots, C_k)$, the algorithm tests each

assignment for membership in $\bigcap_i \text{MaxInc}(C_i)$. If the assignment is not in $\text{MaxInc}(C_i)$ for some i , the remaining MaxInc sets do not have to be tested. However, in the worst case, we still have to compute MaxInc for all of the primitive constraints, as we require in our method.

Bessière and Régin have proposed an algorithm for enforcing GAC on a conjunction of primitive constraints [Bessière and Régin, 1998]. However, as could be expected, this requires $O(d^n)$ time in the worst case.

Other approaches to propagating constraint expressions are based on reification or delaying. It is not hard, however, to show that Inc provides strictly more pruning. Suppose we have a disjunction like $\text{or}(X < 2, X > 4)$. We can reify this into $B_1 \leftrightarrow X < 2$, $B_2 \leftrightarrow X > 4$, and $B_1 + B_2 > 0$. These constraints are delayed until either inequality is entailed or disentailed. However, Inc can prune values immediately. Consider $X \in \{1, 3, 5\}$. Then neither inequality is entailed or disentailed. However, Inc will return the inconsistent assignment $X = 3$.

To perform more pruning on such disjunctions, $\text{cc}(\text{FD})$ introduced constructive disjunction [Van Hentenryck et al., 1998]. If any of the disjuncts, C_i in a constructive disjunction, $C_1 \vee_c \dots \vee_c C_k$ is entailed by the constraint store, then the constructive disjunction is satisfied. Otherwise, each constraint C_i is added in turn to the constraint store and propagated. The resulting inconsistent assignments are recorded, the state restored, and the next constraint is processed. The intersection of the inconsistent assignments found for each constraint are then taken to be the inconsistent assignments for the disjunction. Constructive disjunction can do more pruning than enforcing GAC. For example, even though both $\text{or}(X = 0, Y = 0)$ and $X = Y$ on 0/1 variable are GAC, propagating $X = 0 \vee_c Y = 0$ and $X = Y$ prunes $X = Y = 1$. This extra pruning arises from interaction between the disjunctive constraint and the other constraints in the constraint store. For this reason, constructive disjunction can be very expensive, and may not justify its costs in practice [J. Würtz and T. Müller, 1996].

The cardinality constraint can be used to implement conjunction, disjunction, negation, as well as a host of other useful constraints [Van Hentenryck and Deville, 1991]. However, only a very restricted form of consistency is enforced on the cardinality constraint, and Inc does more pruning in general. For example, if C_1 is $X = 0$, C_2 is $X = 1$ and $\mathcal{D} = \{X = 0, X = 1, X = 2\}$ then $\text{Inc}(\text{or}(C_1, C_2), \mathcal{D}) = \{X = 2\}$. Pruning $X = 2$ makes the problem GAC. However, the equivalent cardinality constraint, $\text{card}(N, [C_1, C_2])$ where $N \geq 1$ is consistent without any prunings.

In contrast to these previous works, we have provided a tractable method for computing a subset of MaxInc that does a useful amount of constraint propagation. Our algorithm is compositional as it uses the propagators provided for the primitive constraints. Hence it can be applied to complex, nested logical expressions.

7 Conclusion

We have proposed a simple and light weight method for propagating logical combinations of primitive constraints. Since computing the maximal set of inconsistent assignments for

such constraint expressions is intractable in general, we have given a polynomial time function which computes a tractable subset compositionally. We characterized precisely when this function computes the maximal set of inconsistent assignments. Finally we have shown that many different global constraints can be implemented using these methods. There remain many interesting directions to follow both from a theoretical and practical perspective. For example, how do we compute and use nogoods for constraint expressions?

References

- [Beldiceanu and Contejean, 1994] N. Beldiceanu and E. Contejean. Introducing global constraints in CHIP. *Mathl. Comput. Modelling*, 20:97–123, no. 12 1994.
- [Bessière and Régin, 1998] C. Bessière and J.-C. Régin. Local consistency on conjunctions of constraints. In *Proc. of ECAI-98 Workshop on Non-Binary Constraints*. 1998.
- [Bessiere et al., 2004] C. Bessiere, E. Hebrard, B. Hnich, and T. Walsh. The complexity of global constraints. In *Proc. of the 19th National Conf. on AI. AAAI*, 2004.
- [Flum et al., 2002] Jörg Flum, Markus Frick, and Martin Grohe. Query evaluation via tree-decompositions. *J. ACM*, 49(6):716–752, 2002.
- [Frisch et al., 2002] A. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, and T. Walsh. Global constraints for lexicographic orderings. In *8th Int. Conf. on Principles and Practices of Constraint Programming (CP-2002)*. Springer, 2002.
- [Law and Lee, 2004] Y.C. Law and J.H.M. Lee. Global constraints for integer and set value precedence. In *8th Int. Conf. on Principles and Practice of Constraint Programming (CP2004)*, pages 362–376. Springer, 2004.
- [Lhomme, 2004] O. Lhomme. Arc-consistency Filtering Algorithms for Logical Combinations of Constraints. In *Proc. of Int. Conf. on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems (CP-AI-OR'04)*. Springer, 2004.
- [Pachet and Roy, 1999] F. Pachet and P. Roy. Automatic generation of music programs. In J. Jaffar, editor, *Proc. of 5th Int. Conf. on Principles and Practice of Constraint Programming (CP99)*, pages 331–345. Springer, 1999.
- [Papadimitriou and Yannakakis, 1984] C. Papadimitriou and M. Yannakakis. The complexity of facets (and some facets of complexity). *Journal of Computer and System Sciences*, 28(2):244–259, 1984.
- [Refalo, 2000] P. Refalo. Linear formulation of constraint programming models and hybrid solvers. In R. Dechter, editor, *Proc. of 6th Int. Conf. on Principles and Practice of Constraint Programming (CP2000)*, pages 369–383. Springer, 2000.
- [Van Hentenryck and Carillon, 1988] P. Van Hentenryck and J.-P. Carillon. Generality versus specificity: An experience with AI and OR techniques. In *Proc. of 7th National Conf. on Artificial Intelligence*, pages 660–664. AAAI, 1988.
- [Van Hentenryck and Deville, 1991] P. Van Hentenryck and Y. Deville. The cardinality operator: a new logical connective for constraint logic programming. In *Proc. of the Int. Conf. on Logic Programming (ICLP 91)*, pages 745–759, 1991.
- [Van Hentenryck et al., 1998] P. Van Hentenryck, V. Saraswat, and Y. Deville. Design, implementation and evaluation of the constraint language $\text{cc}(\text{fd})$. *Journal of Logic Programming*, 37(1–3):139–164, 1998.
- [J. Würtz and T. Müller, 1996] J. Würtz and T. Müller. Constructive disjunction revisited. In *Proc. of 20th German Annual Conf. on Artificial Intelligence*. Springer-Verlag, 1996.

Improved Knowledge Acquisition for High-Performance Heuristic Search

J.P. Bekmann(1,2) and Achim Hoffmann(1)

(1) School of Computer Science and Engineering,

University of New South Wales, NSW 2052, Australia.

(2) National ICT Australia (NICTA), A.T.P, NSW 1430, Australia.

{jbekmann,achim}@cse.unsw.edu.au

Abstract

We present a new incremental knowledge acquisition approach that incrementally improves the performance of a probabilistic search algorithm. The approach addresses the known difficulty of tuning probabilistic search algorithms, such as genetic algorithms or simulated annealing, for a given search problem by the introduction of domain knowledge. We show that our approach is effective for developing heuristic algorithms for difficult combinatorial problems by solving benchmarks from the industrially relevant domain of VLSI detailed routing.

In this paper we present advanced techniques for improving our knowledge acquisition approach. We also present a novel method that uses domain knowledge for the prioritisation of mutation operators, increasing the GA's efficiency noticeably.

1 Introduction

Searching for good or even optimal solutions for complex combinatorial problems has been a challenge in Artificial Intelligence research ever since AI's earliest days.

While it has been long recognized that the effectiveness of a particular search approach depends on the chosen representation as well as certain characteristics of the problem to be solved, to date there is no systematic approach available that allows one to efficiently engineer a special-purpose search strategy for a given search problem.

In this paper we present an approach that allows humans to articulate their intuition about how to search effectively for a given search problem. This appears to be a promising approach as humans often have good intuitions about how to find acceptable or good solutions to search problems, if they had to search manually.

We embedded our approach in the probabilistic search framework of Genetic Algorithms (GA). While GAs have a reputation as being general purpose, in practice it requires usually substantial adaptation effort to the search problem at hand. For complex problems this process may well take months in order to allow the GA to find an acceptable solution within reasonable time.

We chose a probabilistic search algorithm in which to embed the problem specific knowledge about effective search

strategies as this allows one to stop the knowledge acquisition process at any time. A smaller knowledge base (KB) results in a less effective search process, but may still find a satisfactory or near satisfactory solution for less challenging problem instances. Incremental additions to the KB improve the effectiveness of the search.

As part of our discussion, we present a novel technique of varying GA mutation strategy based on execution history, having implications beyond our work for GAs in general.

For the knowledge acquisition process we build on the idea of Ripple Down Rules [3], which allows us to incrementally refine the given (initially empty) knowledge base. Ripple Down Rules (RDR) ensure that a knowledge base is only amended in a way that does not deteriorate its accuracy on previously seen instances.

Our variant of RDR is a significant extension of the RDR approach. We also introduce an expanded knowledge acquisition process in which we help the expert find examples in past search history where the quality of the KB needs attention. This work is based on [1].

In section 2 we provide a short overview of genetic algorithms and RDR. This is followed by section 3 where we describe our framework *HeurEAKA!*. Section 4 introduces a case study of how *HeurEAKA!* was applied to detailed routing, an industrially relevant problem in VLSI design. We discuss our experiments and results to these in section 5, demonstrating that our approach is successful. The conclusions follow in section 6.

2 Background

Before we discuss our approach to combining GAs and Knowledge Acquisition, we start with a brief summary of these two areas:

2.1 Probabilistic Search with Genetic Algorithms

A number of evolutionary algorithms exist; we base our approach on genetic algorithms which is one such algorithm. Basic GAs are relatively easy to implement. A solution candidate for the given problem instance is encoded into a genome. A set of genomes makes up a population of potential solutions. The GA performs a search through the solution space by modifying a population of genomes, guided by an evolutionary heuristic. When a suitable solution has been identified, the search terminates.

GAs usually start with a randomly initialized population of individuals, and guide their search by a *fitness* function of the individuals. In a probabilistic fashion, using operators for selection, mutation and crossover, the GA attempts to direct the search to promising areas. I.e. the GA maintains a certain population size by replacing less fit individuals by newly generated ones. The new individuals are generated by mutation or cross-over based on parent individuals of high fitness.

Due to their generality and flexibility, GAs have been applied in many domains, demonstrating their effectiveness on hard problems [4; 6]. In very complex domains there are significant challenges where generic GA techniques do not scale well. This includes high sensitivity to problem encoding and operator formulation, selection strategies, as well as selection of operator weightings, population size and running time[7; 13; 6]. Thus in practice substantial tuning of the GA is necessary to augment the generic algorithms.

In our approach, we do not use an encoding of our problem into a binary code as is often seen with conventional GAs. Instead, our encoding is a direct representation of a solution (see below for a better description). Our crossover operator is designed for this representation and thus is very effective in maintaining sub-solution coherence in the genome, and also identifying which areas of the genome appear fitter than others. We have a simple yet effective crossover operator, and chose to concentrate our work on the mutation operator - mainly because it was better suited to our the knowledge acquisition approach.

Our approach integrates with the GA a knowledge base where humans can codify their intuition about useful search steps and strategies for the given type of problem.

In addition, we present a novel method that uses domain knowledge for the prioritisation of mutation operators, increasing the GA's efficiency noticeably. Because we have sufficient domain knowledge in our fitness function, we can identify parts of the genome that are desirable or undesirable and use that knowledge to influence mutation strategy. We also show how using the mutation history of these identified characteristics to change mutation selection strategy.

2.2 Knowledge Acquisition using Ripple Down Rules

RDR [3] allows one to add rules to a knowledge base incrementally without compromising the previously shown satisfactory performance of the KB. An extension of RDR allows hierarchical structuring of RDRs - "nested RDR" (NRDR) [2]. NRDR allows the re-use of definitions in a KB, as well as the abstraction of concepts, which allows for more compact KBs.

Single Classification Ripple Down Rules (SCRDR) use a binary tree where the root node is also called the default node. To each node in the tree a rule is associated, with a condition part and a conclusion which is usually a class - in our case it is an operator application though. A node can have up to two children, one is attached to an *except* link and the other one is attached to the so-called if-not link. The condition of the default rule in the default node is always true and the conclusion is the default conclusion. When evaluating a tree on a case (the object to be classified), a *current conclusion* vari-

able is maintained and initialised with the default conclusion. If a node's rule condition is satisfied, then its conclusion overwrites the *current conclusion* and the except-link, if it exists, is followed and the corresponding child node is evaluated. Otherwise, the if-not link is followed, if it exists, and the corresponding child node is evaluated. Once a node is reached such that there is no link to follow the *current conclusion* is returned as a result.

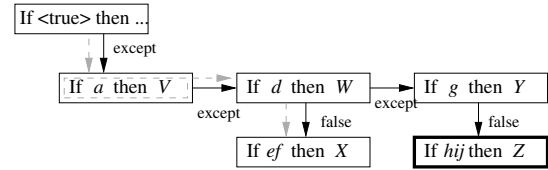


Figure 1: A simple RDR structure. The dashed line indicates the path taken & rule executed (*V*) for *ade*, the bold box indicates how action *Z* would be added for *adghij*.

Figure 2.2 shows a simple RDR tree structure. In bold is a rule that a user might have added for the case where conditions *adghij* hold, causing action *Z* to be executed, instead of *W*.

For the purpose of integrating with the Genetic Algorithm, in our approach each case is a genome and conclusions are actually a sequence of actions that can be applied to the genome.

In typical RDR implementations, any KB modification would be by adding exception rules, using conditions which only apply to the current case for which the current knowledge base is inappropriate. By doing this, it is ensured that proper performance of the KB on previous cases is maintained.

Nesting RDRs allows the user to define multiple RDRs in a knowledge base, where one RDR rule may use another, nested RDR tree in its condition, (and in HeurEAKA! as part of an action, see section 3). I.e. the nested RDR tree is evaluated in order to determine whether the condition is satisfied. A strict hierarchy of rules is required to avoid circular definitions.

3 HeurEAKA! - Our Framework for Improving Search with Domain Knowledge

The HeurEAKA! (Heuristic Evolutionary Algorithm with Knowledge Acquisition) framework is composed of a general GA scheme and a knowledge acquisition (KA) component.

Each genome in the GA population represents a potential solution to the problem to be solved. While searching, the GA needs to perform a fitness evaluation and the mutation of a genome. For this it uses the two knowledge bases, the *Mutation KB* and *Fitness KB*. The GA presents a genome to the relevant KB, which responds by suggesting a mutation or fitness score. (The crossover function is handled differently, more details below).

The KA module handles maintenance of the KBs, which contain collections of NRDRs. The NRDRs are incrementally developed by the expert through the addition of rules

(the next section contains a detailed description of this process).

The *Fitness KB* contains rules based on past examples given by the expert about desirable and undesirable traits of genomes. The GA requires a fitness value, so the rules contain various calculations the expert adds to reflect these traits.

The *Mutation KB* similarly contains rules accumulated from expert recommendations. Each rule contains an action to be executed under certain conditions. Each action is one or more commands which can modify the genome. The KB contains a set of NRDRs which contain these rules. When a genome is presented to the Mutation KB, it is evaluated against these NRDRs. In accordance to the RDR algorithm, actions which satisfy the relevant conditions are selected for execution. A non-deterministic element is introduced by letting the expert attach probabilistic weights to selected NRDRs and letting a random procedure pick which one to execute. This allows the expert leeway to speculate about what the best actions might be, effectively letting the GAs heuristics “decide” the most appropriate action.

As opposed to other methods related to case-based reasoning and GAs (e.g. [12] or [11]), the expert uses past cases to formulate rules to explain improvements, rather than using past examples or similar solutions injected into the evolutionary process, or trying to extract the rules automatically.

The overall architecture of HeurEAKA! is domain neutral. The implementation contains KB management modules, GA, a graphical user interface as well as range of utilities supporting the expert evaluate past searches. This is complemented with a smaller component that needs to be implemented for each problem domain containing a relevant solution representation. The user interface also requires a small addition to translate this representation for visualisation.

A general purpose language is extended with primitives to make it suitable for describing the domain specific representation. Rules entered by the expert are specified using this simple language, which is loosely based on “C” syntax. The rule conditions use logical expressions while actions include a range of commands, including variables, loops and references to other NRDRs which are evaluated analogously to function calls. Section 5 gives examples of rule specification.

3.1 The Knowledge Acquisition process in HeurEAKA!

The overall approach allows the user to run the Genetic Algorithm with the current KBs and monitor or inspect afterwards whether the fitness function computed for an individual as well as the applied mutation operator where appropriate/promising steps to do.

To do that, the genetic algorithm can be started, stopped and reset via a graphical user interface. A snapshot of the GA population is presented visually, from which the user can pick an individual for closer inspection.

Figure 2 shows an individual genome being inspected. A user can step through the process of mutation and evaluation of that genome as if the solution was part of the GA population. Given non-deterministic elements of operator selection, the interactive debugger maintains complete state descriptions, allowing the user to step forward and backwards in

execution history and also recreating conditions for repeated testing.

The expert can review actions applied to the genome and make modifications to the KB if he/she feels they are needed. The modifications are typically done by adding exceptions to the NRDR (in accordance with RDR methodology described above), but may also be an edit of an existing rule or the addition of a new NRDR.

In the traditional Ripple Down Rules approach it is assumed every rule entered is correct and has its reason to be there, modification of a rule may have undesirable side effects which are often difficult to control. While these are valid reasons for not modifying rules, our experiments suggest that it is sensible to modify rules at the beginning of building a knowledge base. In particular, for the definition of mutation operators it proved useful to have this option to modify rules.

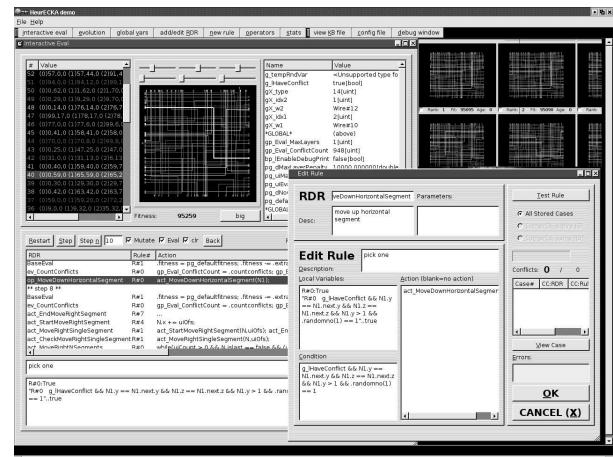


Figure 2: The interactive screen allows the user to inspect the individual genome, step through rule application, and amend the KB as needed.

3.2 Advanced features for supporting Knowledge Acquisition

Once a sufficiently large KB has been created based on small-scale evaluation, more automated methods are needed to support a user’s interaction:

Genome Visualisation: One of the strengths of using RDR is that the approach is successful in eliciting tacit knowledge [3], since rules are formulated while the system is in use and with the justification of an example. With complex problems (such as is described in section 4), the visualisation of a solution is very important in allowing the user to form an intuitive understanding. In the evaluation module, there is immediate visual feedback when rules are applied.

Trigger functions: During the execution of the GA, thousands or even hundreds of thousands of rule applications to all manner of genomes are made. It is not possible for the expert to supervise each case. The user can define a trigger function which is applied to each genome during the execution of the GA. If the function’s condition is met, an exception is

[illegible]

Comparing different KBs

(avg. run for: layout: 30 pin, 25 tracks, density 20)

Conflicts

Minimal KB
Small KB
Large KB

Total Mutations / Population Size

(note: this axis is abridged for compactness;
only the Large KB reaches 0 conflicts)

NRDR usage profile: Execution statistics are kept for each NRDR execution, recording the frequency of use over many GA runs. Comparing the profile of aggregate use of NRDRs by successful vs less successful genomes (judging by their fitness function and how soon they were eliminated from the GA population), provides some impression of an NRDRs “usefulness” and role in GAs lifetime. Figure 3 shows the frequency of a few selected NRDRs relative to the genome’s age (the examples are from the VLSI case study discussed in the next section).

The characterisation of NRDR utility can only provide the expert with a rough idea of its “usefulness”. In that context it did contribute somewhat to the intuition the expert developed when doing KA. Where it was found to be quite useful though, was when the expert introduced some modification to the NRDR which inhibited it from functioning as well as previously - meaning that it was much less used in arriving at fit genomes. This helped catch at least a few of errors. In figure 3 the arrow indicates such an NRDR.

In our experiments we identified a set of NRDRs which would probably be useful for various types of problems. For

Identifying genome modifications of interest: Sometimes a genome mutation plays a role in the long term success of that genome, but the immediate result of the mutation is a reduction or no change in its fitness. In general, GAs cope with this by having a large enough population pool and preventing premature fitness convergence, so as not to throw out such genomes too soon. In traditional GAs it is hard to otherwise account for these cases. Given that we have an expert who can make judgment on individual examples using intuition and hindsight, the fitness function can be augmented to bridge the interval needed to see a promising mutation come to fruition.

We then tried using a discounted reward scheme, and some other methods to help reduce the candidate space, but have not yet been able to improve matters. This aspect of the project are currently being worked on.

Preferences in choosing parts of the genome to mutate: As hinted at in the genetic algorithms section (2.1), we have introduced a novel method of choosing parts of a genome for mutation. In our fitness function we are able to identify parts of the genome that are undesirable.

In the context of our experimental domain, we empirically found a 65% discount factor gave the best results. Compared with no discount scheme (giving all identified problem areas equal chance) the number of successful GA trials was 85% higher, and the successful trials took on average half the time to complete.

CONSTRAINT SATISFACTION AND SEARCH

the rule specification. This function allows for the comparison of a genome before and after a mutation has been made. This effectively allows a local search in the mutation NRDR (similar to a Lamarckian learning mechanism in evolutionary algorithms).

Our knowledge base incorporated local search by comparing the fitness of a genome before and after a candidate mutation. This can be used in an RDR at any point, but was most successfully used for a generalised mutation look-ahead. Here, for each mutation a series of 10 steps, each recommended by the mutation KB was used, while the fitness values were recorded. The point of highest fitness value in the sequence was picked as the final mutation.

When using the look-ahead mechanism, GA trials had a significantly higher success rate. Each successful trial was also faster to converge on a solution (as measured in overall execution time).

4 Case Study - Detailed VLSI Routing

In order to demonstrate that genetic algorithms enhanced with knowledge acquisition can be used to develop algorithms for solving complex combinatorial problems, detailed channel routing as well as switchbox routing, both industrially relevant problems within the realm of VLSI design, were chosen to demonstrate the approach.

4.1 Domain Specifics

A channel routing problem (CRP) is given by a channel of a certain width. On both sides of the channel are connection points. Each connection point belongs to a certain electrical net and all connection points of the same net need to be physically connected with each other by routing a wire through the channel and, of course, without two nets crossing. The width of the channel determines how many wires can run in parallel through the channel. The length of the channel determines how many connection points on both sides of the channel there may be. Furthermore, the layout is done on a small number of different layers (e.g. 2 to 4 layers), to make a connection of all nets without crossing possible at all. Two adjacent layers can be connected at any point using a so-called *via*.

The switchbox routing problem (SRP) is similar to the CRP but usually more difficult, as it deals with connections on all four sides rather than only two. A solution to the CRP and SRP will be referred to as a *layout*.

Genome Encoding: A genome describes the layout of a CRP or SRP solution. This takes the form of a list of straight wire segments.

Initially, a genome will usually not represent a valid solution as some wires are usually crossing. Only when all those crossings have been eliminated and not more than the prescribed number of layers are used would the fitness value of a genome reach a satisfactory level.

Genome Operations: Individuals in the GA are initialised with a random wire layout without regard to conflicts. The GA operates on a genome by crossover, mutation and evaluation. However, the GA's crossover operation is currently not controlled by a knowledge base. The crossover for our

layout problems groups wires into mutually exclusive sets, one of those sets is exchanged between parents (i.e. the corresponding wires are exchanged). The crossover operator is thus highly sensitive to the structure of the problem, as well as interaction between its elements. The formulation of the crossover operator is domain specific, and we plan to make it part of the knowledge acquisition cycle in future versions.

Evaluation is done using the evaluation KB. Typically the user uses as a fitness criteria the number of layers and conflicts in a layout. The length of wires, number of vias and possible cross-talk (electronic interference occurring in parallel wires) are also useful fitness criteria.

The mutation KB contains rules designed to manipulate the layout, typically they would describe the resolution of a conflict identified using the *findconflict* command (a primitive function returning a conflict found in the layout).

Example of rules applied to the Switchbox Routing problem: Initially, a KB is built up defining operators using primitives based on node and wire manipulation. These form the foundation for more high-level concepts which can be used intuitively by an expert.

Assuming we start with a KB with relatively high-level actions defined, e.g. *MoveVerticalSegmentRight*, and *MoveHorizontalSegmentUp*. When applied to the example in figure 5, it seems that the action *MoveVerticalSegmentRight* is unlikely to lead to a promising solution. The expert can amend the KB to suggest an alternative action. (see nodes labeled *N1* and *N2* in figure 5). In this case, the user will find that *MoveVerticalSegmentRight* is undesirable, and formulate a rule with condition *is_Vertical(N1) && is_Horizontal(N2) && right_of(N2.next,N1)*, and action *MoveHorizontalSegmentUp(N1.prev)*. This rule would be added as an exception in the KB. The operators referenced here are defined as RDRs elsewhere in the KB. *is_Vertical(N1)* returns true if the segment between *N1* and its succeeding node is vertical (change in row), *is_Horizontal(N2)* returns true if the segment between *N2* and its successor is horizontal (change in column). *right_of(N2.next,N1)* will return true if the node succeeding *N2* lies to the right of *N1*.

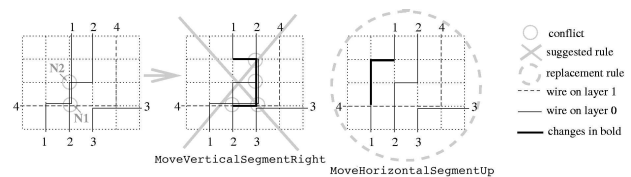


Figure 5: An exception is created by the expert, replacing the action *MoveVerticalSegmentRight* suggested by the KB, with the action *MoveHorizontalSegmentUp(N1.prev)*. (The tags *N1* and *N2* are referenced in the text).

5 Experiments and Results

Experiments: In order to test our approach and the implemented tool, a KB was created. Initial tests were done with a KB containing 2 RDRs and 10 rules, later tests were run with 50 RDRs and 167 rules and 53 RDRs and 182 rules.

In order to show that the introduction of domain knowledge improved the search, we tested three different KBs: A “minimal” KB with 5 rules, a “small” KB containing 10 rules, and a “large” KB containing 167 rules. Using the first two, the GA was unable to solve the given problem, while using the third, it was. Fig.4 shows progressive improvement in search with KB size.

A KB containing 53 RDRs and 182 rules was created. Initial rules were low-level in nature, dealing with manipulation of nodes and wires. This technical work required frequent editing and debugging. Using the validation module with trigger functions (described above) was helpful at this stage.

After the low level rules were defined, rules could be defined at a higher level of abstraction. Because this was a more intuitive abstraction level, it was easier to formulate rules and required less revision. The high level NRDRs were used by the mutation KB, for what the probabilistic algorithm was intended: choosing from the different high level strategies in an attempt to resolve conflicts.

Results: We were able to solve well recognised benchmarks from the domain of switchbox and channel routing. These included Burstein’s difficult channel routing problem, Burstein’s switchbox routing problem, the DENSE switchbox and the JOO_6_16 problem, amongst others. The solutions found were comparable to others’ attempts, including those of the WEAVER, Silk, Packer and Monreale routers [9], [5], [10].

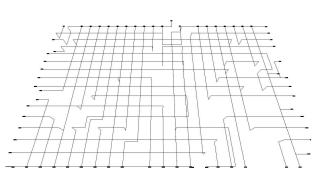


Figure 6: Solution to Burstein’s switchbox problem.

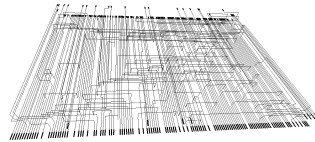


Figure 7:
A sample channel routing solution.

We found it initially difficult to complete some of the more challenging benchmarks until we used the techniques listed in sections 3.2 and 3.3

On average rules took approximately 10 minutes each to formulate, taking about 30 hours for the formulation of a viable knowledge base. The formulation of effective CRP and SRP algorithms has been the subject of much study and industry-standard algorithms took many years to develop [8]. In our case KA was done by a novice, using mainly intuition and being able to incrementally specify rules in a natural way on the knowledge level. Thus the effort and expertise required was significantly less than commercial routing solutions.

6 Conclusion

We have presented an approach to solving complex combinatorial problems by using a generic search algorithm and incrementally introducing domain knowledge in order to make the search tractable. We use knowledge acquisition that supports the exploratory development of solutions, which, when combined with a GA, makes it easier to tackle difficult problems than having to design a conventional algorithm.

While GAs have been successfully used in many domains, in practice they often require considerable augmentation to make the generic architecture effective in complex domains. The unconventional KA technique of RDR formalises this adaptation process, allowing an expert to intuitively develop a domain-specific solution. In order to use RDR for integration with GAs, we have introduced new methods to make them work with probabilistic search.

We have also presented a number of advanced techniques that enhance the KA process. These include the use of trigger functions, NRDR usage profiles and weightings, mutation look-aheads and automatic case identification.

As part of our mutation operators, we have identified a promising technique for improving probabilistic search by using the mutation history of a genome to bias the search into more recent areas of exploration. We have found this heuristic to return promising results.

The application of our framework in the well understood domain of detailed channel routing and switchbox routing, enables us to compare our results against industrially used algorithms. We have shown that we are able to solve accepted benchmarks competitively, using far less effort than needed for the development of conventional algorithms.

In future work we will investigate the application of our techniques to a different domain. Another worthwhile direction would be to explore the applicability of our advanced KA techniques to more traditional KA domains outside of search.

References

- [1] Bekmann, J.P., Hoffmann, A.: Incremental Knowledge Acquisition for Improving Probabilistic Search Algorithms. In: 14th International Conference on Knowledge Engineering and Knowledge Management (2004) 248–264.
- [2] Beydoun, G., Hoffmann, A.: Theoretical basis for hierarchical incremental knowledge acquisition. In: International Journal in Human-Computer Studies. (2001) 407–452
- [3] Compton, P., Jansen, R.: Knowledge in context: A strategy for expert system maintenance. In: 2nd Australian Joint Artificial Intelligence Conference. Volume 1. (1989) 292–306
- [4] De Jong, K., Spears, W.: Using genetic algorithm to solve NP-complete problems. In Schaffer, J.D., ed.: Proc. of the Third Int. Conf. on Genetic Algorithms, San Mateo, CA, Morgan Kaufmann (1989) 124–132
- [5] Gockel, N., Pudenko, G., Drechsler, R., Becker, B.: A hybrid genetic algorithm for the channel routing problem. In: International Symposium on Circuits and Systems, volume IV. (1996) 675–678
- [6] Goldberg, D.E.: The Design of Innovation: Lessons from and for Competent Genetic Algorithms. Volume 7, Kluwer Series on Genetic Algorithms and Evolutionary Computation. Kluwer Academic Publishers (2002)
- [7] Holland, J.: Adaptation in Natural and Artificial Systems. University of Michigan Press. (1975)
- [8] Lengauer, T.: Combinational Algorithms for Integrated Circuit Layout. B.G. Teubner/John Wiley & Sons (1990)
- [9] Lienig, J., Thulasiraman, K.: A new genetic algorithm for the channel routing problem. In: 7th International Conference on VLSI Design, Calcutta (1994) 133–136
- [10] Lin, Y., Hsu, Y., Tsai, F.: Silk: A simulated evolution router. In: IEEE Transactions on CAD. Volume 8.10. (1989) 1108–1114
- [11] Liu, X.: Combining genetic algorithm and case-based reasoning for structure design. Masters Thesis (1996), Department of Computer Science, Univ. of Nevada.
- [12] Perez, E.I., Coello, C., Aguirre, A.H.: Extraction and reuse of design patterns from genetic algorithms using case-based reasoning. In: Soft Computing 9(1) (2005) Springer 44–53.
- [13] Rothlauf, F.: Representations for Genetic and Evolutionary Algorithms. Springer Verlag (2002)

Extracting Certificates from Quantified Boolean Formulas *

Marco Benedetti

Istituto per la Ricerca Scientifica e Tecnologica (IRST)

Via Sommarive 18, 38055 Povo, Trento, Italy

benedetti@itc.it

Abstract

A *certificate of satisfiability* for a quantified boolean formula is a compact representation of one of its models which is used to provide solver-independent evidence of satisfiability. In addition, it can be inspected to gather explicit information about the semantics of the formula. Due to the intrinsic nature of quantified formulas, such certificates demand much care to be efficiently extracted, compactly represented, and easily queried. We show how to solve all these problems.

1 Introduction

The term “certificate” has a fairly general meaning, originating in language recognition and complexity theory. Once *verified*, a certificate proves that the string it refers to actually belongs to a language of interest. Applied to logic, the term denotes any means of providing evidence of (un)satisfiability for a given statement, other than a refutationally-competent deductive approach. In essence, we verify that a given logical formula belongs to the language of (un)satisfiable statements.

The most natural *certificate of satisfiability* (sat-certificate) for a formula is an explicit representation of some of its *models*. A formula is indeed satisfiable if and only if some model makes it evaluate to true. The validity of a certificate can be verified by whoever is knowledgeable about the *evaluation apparatus* of the logic (deductive capabilities are unnecessary), independently of how it was obtained.

In this paper, we focus on sat-certificates for *Quantified Boolean Formulas* (QBFs). Such certificates have never been proposed or used so far for a number of reasons. First, the intrinsic nature of a QBF confers a tree-shaped structure to its models, whose explicit representation may become unaffordable. Second, theoretical arguments exist that make it unlikely to find polynomial-time verification procedures (QBF satisfiability is PSPACE-complete [Stockmeyer and Meyer, 1973]). Finally, present QBF solvers find it either impractical or not straightforward to collect all the information needed to construct a model. As a consequence, QBF models feature no

commonly accepted representation (if any at all), and all the current solvers return a little more than a sat/unsat answer.

Despite these issues, sat-certificates for QBFs are extremely desirable thanks to their potential benefits on applications and solvers. For example, a certificate is a conclusive means to judge conflicting answers given by different solvers on the very same instance. Clearly, this event reveals no finer problem than a bug in the implementation, which we might think is not worth considering. This happens fairly often though, and as long as we treat solvers as black boxes, a proof-of-satisfiability approach is the only realistic way to tell the truth. We quote [Le Berre *et al.*, 2004]:

The question of how to check the answer of the QBF solvers in an effective way is still unanswered [...] the question of what is a good certificate of satisfiability/unsatisfiability [...] remains open. This point is not only an issue for the QBF evaluation, but also for the implementation: [...] we had soundness problems with 4 QBF solvers. [...]

Yet, a certificate is much more than a way to ensure satisfiability: It can be *inspected* to gather semantics from the underlying formula. This is of paramount importance in applications, where certificates add valuable information to a mere sat/unsat answer. For example, a sat answer to the propositional (PROP) encoding of (the negation of) a desired property over a logic circuit means that the circuit is faulty w.r.t. that property. But, it takes a certificate to outline a definite scenario in which the fault shows up. As opposed to QBF certificates, such PROP certificates are easy to represent and verify, hence they have had a wide application.

The relevance of certificates enlarges with the scope of application of the underlying logic. In this respect, QBF is a notable case with plenty of applications. Every problem that can be stated as a two-player finite game can be modeled in QBF. An insightful example is obtained by considering the famous game “Connect-4”. It is known that the player who moves first can always win. The rules of the game and the existence of a winning strategy can be encoded into a QBF instance [Gent and Rowley, 2003], expected to be sat. Which is the winning strategy? A certificate would disclose such information: The first player would prevail by just inspecting the certificate at each move, whatever the opponent does.

The interesting point here is that many real-world applications can be modeled as two-player games: Unbounded

*This work is supported by PAT (Provincia Autonoma di Trento, Italy), under grant n. 3248/2003.

model checking for finite-state systems [Rintanen, 2001] and conformant planning [Rintanen, 1999]—just to name two relevant examples—have handy QBF formulations.

In the rest of this paper, after a brief introduction to QBFs and their models (Section 2), we present a solver-independent representation for QBF sat-certificates (Section 3). As expected, we are able to describe how to verify them (Section 4) before the more complex task of their extraction is addressed (Section 5). We conclude by discussing the implementation of our approach and the future work (Section 6).

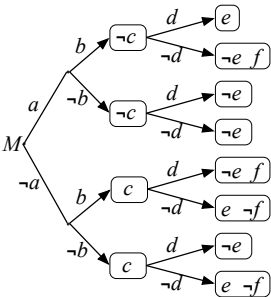
2 QBFs and their models

With no loss of generality, we consider QBFs in *prenex conjunctive normal form* (CNF). They consist in a *prefix* exhibiting an arbitrary number of alternations of existentially and universally quantified variables, followed by a *matrix*, i.e. a conjunction of clauses. For example:

$$\begin{aligned} \forall a \forall b \exists c \forall d \exists e \exists f. & (\neg b \vee e \vee f) \wedge (a \vee c \vee f) \wedge (a \vee d \vee e) \wedge \\ & (\neg a \vee \neg b \vee \neg d \vee e) \wedge (\neg a \vee b \vee \neg c) \wedge (\neg a \vee \neg c \vee \neg f) \wedge \\ & (a \vee \neg d \vee \neg e) \wedge (\neg a \vee d \vee \neg e) \wedge (a \vee \neg e \vee \neg f) \end{aligned} \quad (1)$$

Given a QBF F , we denote by $\tilde{F}$ its matrix, by $\text{var}_{\exists}(F)$ ($\text{var}_{\forall}(F)$) the set of existentially (universally) quantified variables in F , and by $\text{var}_{\forall}(F, e) \subseteq \text{var}_{\forall}(F)$ the set of universal variables preceding (or *dominating*) $e \in \text{var}_{\exists}(F)$ in the prefix (we pose $\delta(e) \doteq |\text{var}_{\forall}(F, e)|$). Given a CNF matrix $\tilde{F}$, the formula $\tilde{F} * l$ is the CNF obtained by *assigning* the literal l , i.e. by removing from F each $\neg l$ literal and each clause containing l . This notation is readily extended to sets of literals. A matrix $\tilde{F}$ is *satisfied* by a set of literals M (written $M \models \tilde{F}$) when $\tilde{F} * M$ is the empty formula.

The alternations of quantifiers in the prefix guide us to extending this notion of satisfiability from matrixes to QBFs. For example, the satisfiability problem on (1) asks whether for each possible (consistent) combination of literals on a and b there exists a way to choose a literal on c such that for both possible literals over d , two literals on e and f exist such that the resulting set satisfies the matrix. Hence, a QBF model is a set of $|\text{var}_{\exists}(F)|$ functions, each one specifying the literal



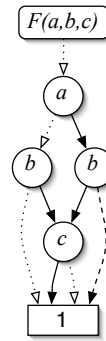
to be chosen (if any) on the existential variable e as a function of the choices on all the universal variables dominating e . To represent all these functions at once we can use a labeled tree, such as the one aside, depicting a model for (1). If we call *universal hypothesis* every consistent set of universal literals (or, equivalently, every assignment

$\{u_1 = \psi_1, \dots, u_n = \psi_n\}$ to the universal variables, with $\psi_i \in \mathfrak{B} = \{0, 1\}$, we may say, informally, that a tree structure like the one above is a model for a QBF with matrix $\tilde{F}$ iff for every universal hypothesis U the set of existential literals collected along the branch individuated by U satisfies $\tilde{F} * U$ (see [Büning and Zhao, 2004] for details).

3 Certificate representation

A QBF model can be represented *explicitly* by employing data structures such as trees or truth tables. Or, we may pursue *compactness* at the expense of managing an *implicit* representation¹ requiring computation to yield values.

An ideal certificate should be compact (easy to manage) and explicit (easy to verify and query). A successful tradeoff is obtained by employing *Binary Decision Diagrams* [Bryant, 1986]. We consider their *reduced ordered* version (ROBDDs, or just BDDs henceforth) with *complemented arcs*. A BDD $\mathcal{E}$ representing a total function $F(u_1, u_2, \dots, u_n)$ from $\mathfrak{B}^n$ to $\mathfrak{B}$ is a directed acyclic graph with one root (labeled by F) and one sink node (labeled by “1”). Each internal node is labeled by one variable in $U = \{u_1, u_2, \dots, u_n\}$, and always has two children, one attached to the outcoming *then-arc*, the other to the *else-arc*. The *else-arc* may or may not be *complemented*. A unique path from the root to the sink is identified by assigning a value to each variable in U : The *then-arc* is chosen for variables assigned to 1, the *else-arc* is followed otherwise. The function F represented by $\mathcal{E}$ evaluates to 1 on $\langle \psi_1, \psi_2, \dots, \psi_n \rangle \in \mathfrak{B}^n$ iff an even number of complemented arcs is encountered along the path defined by $\psi_1, \psi_2, \dots, \psi_n$.



As an example, let us consider the BDD aside, where solid arrows denote then-arcs, while dashed (dotted) arcs are used for regular (complemented) else-arcs. It represents a binary function $F(a, b, c)$ of three binary variables a , b and c . It is, for example, $F(0, 1, 1) = 1$ and $F(1, 1, 1) = 0$. The represented function may be written as $F = b \wedge (a \vee c) \wedge (\neg a \vee \neg c)$. In a set-oriented interpretation, this BDD represents the *one-set* of F , i.e. the set having F as characteristic function. In our case, it stands for the set $\{\langle 0, 1, 1 \rangle, \langle 1, 1, 0 \rangle\}$ where F evaluates to 1.

The BDDs we utilize are *ordered* and *reduced*: The same variable ordering is followed along each path, and no two nodes representing the same set exist, so that each function has only one *canonic* representation. Furthermore, the version with complemented arcs is such that the set $\bar{S}$ is denoted by the same node as S (referred to with a complemented arc).

The BDD way of representing sets is regarded as *symbolic* in that it avoids the explicit enumeration of sets' elements in favor of a more abstract, diagram-based way of computing characteristic functions. Such representations may be exponentially more succinct than explicit ones (see [Wegener, 2000]), and all the operations on the sets/functions they represent (union/disjunction, intersection/conjunction, etc.) can be performed by manipulating the involved BDDs [Bryant, 1986]. With a small abuse of notation, we treat BDDs as if they were the sets they represent. For example, $x \in \mathcal{E}$ is an element in the subset of $\mathfrak{B}^n$ individuated by $\mathcal{E}$.

When we manage collections of BDDs, canonicity spans over their set of nodes as a whole. This allows the sharing of structural information among diagrams. A BDD in such a set

¹In [Büning and Zhao, 2004] propositional formulas and QBFs with free variables are used. *Implicitness* is not an issue for the authors as they focus on characterizing classes of models/formulas.

of interconnected diagrams—called a *forest*—is identified by a (complemented) arc pointing to its root node.

Definition 3.1 (QBF sat-certificate, validity) A *sat-certificate* for a QBF F with $\text{var}_{\exists}(F) = \{e_1, \dots, e_m\}$, $\text{var}_{\forall}(F) = \{u_1, \dots, u_n\}$, and $\delta_i = \delta(e_i)$ is a forest of BDDs containing two roots $\langle \mathcal{E}_i^+, \mathcal{E}_i^- \rangle$ for each i in $[1, m]$. Both $\mathcal{E}_i^+$ and $\mathcal{E}_i^-$ are defined over $\text{var}_{\forall}(F, e_i) = \{u_1, \dots, u_{\delta_i}\}$. The certificate

$$\mathcal{C}(F) = [\langle \mathcal{E}_1^+, \mathcal{E}_1^- \rangle, \langle \mathcal{E}_2^+, \mathcal{E}_2^- \rangle, \dots, \langle \mathcal{E}_m^+, \mathcal{E}_m^- \rangle]$$

is consistent when $\forall i \in [1, m]$ it is $\mathcal{E}_i^+ \cap \mathcal{E}_i^- = \emptyset$. It is valid for F when for any $\langle \psi_1, \dots, \psi_n \rangle \in \mathfrak{B}^n$ the formula $\tilde{F}_{[u_1=\psi_1, \dots, u_n=\psi_n]}$ is satisfied by $\{e_i = s^{(i)}(\psi_1, \dots, \psi_{\delta_i}), i \in [1, m]\}$, where the functions $s^{(i)} : \mathfrak{B}^{\delta_i} \rightarrow \mathfrak{B}$ are defined as

$$s^{(i)}(\psi_1, \dots, \psi_{\delta_i}) = \begin{cases} 1 & \text{if } \langle \psi_1, \dots, \psi_{\delta_i} \rangle \in \mathcal{E}_i^+ \\ 0 & \text{if } \langle \psi_1, \dots, \psi_{\delta_i} \rangle \in \mathcal{E}_i^- \\ \text{undef. otherwise} \end{cases}$$

In essence, a sat-certificate is a compact but explicit representation of the dependencies that have to exist between existential (dependent) and universal (independent) variables in order to satisfy the matrix whichever the universal hypothesis.

Lemma 3.1 If $\mathcal{C}(F)$ is valid for a QBF F , then F is satisfiable. Every satisfiable QBF has at least one valid certificate.

Proof sketch. A QBF is satisfiable iff it has at least one model, i.e. iff we find at least one tree-like structure (like the one introduced in Section 2), such that for every assignment $U = [u_1 = \psi_1, \dots, u_n = \psi_n]$ to the universal variables the set of existential literals collected along the branch individuated by U satisfies $\tilde{F} * U$. Given a consistent certificate $\mathcal{C}$ for F , we insert the literal e_i into the label of the node reached following the $u_1 = \psi_1, \dots, u_{\delta_i} = \psi_{\delta_i}$ path iff $\langle \psi_1, \dots, \psi_{\delta_i} \rangle \in \mathcal{E}_i^+$ (and, dually, $\neg e_i$ appears in the label iff $\langle \psi_1, \dots, \psi_{\delta_i} \rangle \in \mathcal{E}_i^-$). By construction, if the certificate is valid according to the notion of validity given in Definition 3.1, the tree-like structure obtained is a model. $\square$

A valid sat-certificate for (1) is depicted in Figure 1.

4 Certificate verification

The first thing we wish to do with a consistent certificate $\mathcal{C}$ for F is to verify its validity. We check that by choosing the truth values of the existential variables according to what the certificate suggests, we always satisfy the matrix.

Function checkValidity(QBF F , certificate $\mathcal{C}$)

```

Let  $\text{var}_{\exists}(F)$  be  $\{e_1, \dots, e_m\}$ ;
Let  $\text{var}_{\forall}(F)$  be  $\{u_1, \dots, u_n\}$ ;
Let  $\mathcal{C}$  be  $[\langle \mathcal{E}_1^+, \mathcal{E}_1^- \rangle, \langle \mathcal{E}_2^+, \mathcal{E}_2^- \rangle, \dots, \langle \mathcal{E}_m^+, \mathcal{E}_m^- \rangle]$ ;
forall  $\Gamma \in F$  do
  Let  $\Gamma$  be  $\psi_1 \otimes u_{i_1} \vee \dots \vee \psi_h \otimes u_{i_h} \vee$ 
     $\phi_1 \otimes e_{j_1} \vee \dots \vee \phi_k \otimes e_{j_k}$ ;
   $\mathcal{E} \leftarrow (\cup_{\phi_i=0} \mathcal{E}_{j_i}^+) \cup (\cup_{\phi_i=1} \mathcal{E}_{j_i}^-)$ ;
  if  $\bar{\mathcal{E}}_{[u_{i_1}=\bar{\psi}_1, \dots, u_{i_h}=\bar{\psi}_h]} \neq \emptyset$  then return FALSE;
return TRUE;
```

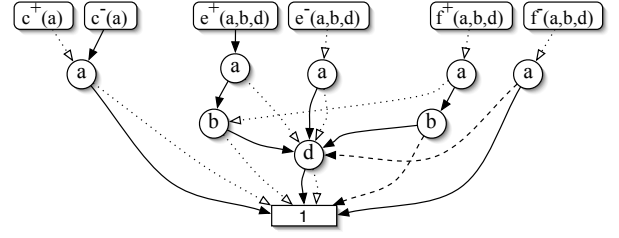


Figure 1: A BDD-based sat-certificate for the QBF (1).

An easy but impractical way of checking a certificate would be to check that $\mathcal{M}$ produces a satisfying assignment under all the possible universal hypotheses. Fortunately, the symbolic nature of the certificate helps us to perform a much more efficient, clause by clause, BDD-based verification. Let us use the exclusive or “ $\otimes$ ” to construct literals out of variables ($\varphi \otimes v$ means v when $\varphi = 0$, and $\neg v$ when $\varphi = 1$).

Lemma 4.1 The algorithm checkValidity answers TRUE on $\langle F, \mathcal{C} \rangle$ if and only if $\mathcal{C}$ is a valid certificate for F .

Let us consider, for example, the clause $\neg u_1 \vee e_1 \vee u_2 \vee \neg e_2 \vee e_3$ under the prefix $\forall u_1 \exists e_1 \forall u_2 \exists e_2 \exists e_3$. The only relevant universal hypotheses for this clause are those assigning both $u_1=1$ and $u_2=0$: All the others immediately satisfy the clause via one of its universal literals. So, it remains to verify that under the assignment $[u_1=1, u_2=0]$ at least one of the three remaining literals in the clause is true, i.e. that every universal hypothesis containing $u_1 = 1, u_2 = 0$ falls within the one-set of at least one out of $\mathcal{E}_1^+$ (for e_1), $\mathcal{E}_2^-$ (for $\neg e_2$), and $\mathcal{E}_3^+$ (for e_3). This is a two-step check: First, we collect the universal hypotheses $\mathcal{E} = \mathcal{E}_1^+ \cup \mathcal{E}_2^- \cup \mathcal{E}_3^+$ under which the clause is satisfied by some existential literal. Then, we check that all the hypotheses $\bar{\mathcal{E}}$ (in which no existential literal satisfies the clause) assign either $u_1 = 1$, or $u_2 = 0$, or both, so that the clause is satisfied by a universal literal.

The meaning of a successful verification is twofold: We are ensured that the formula is sat, and that the certificate encodes a model. Conversely, the verification fails when either the certificate is invalid or the formula is unsat (we cannot tell right away which circumstance has occurred). Validity check is a coNP-complete problem [Büning and Zhao, 2004].

5 Certificate Extraction

For certificate extraction to be symbolic in the same sense as our certificate is, we want it to work on a BDD-based representation of the problem. We describe one such representation (Section 5.1), and show how it relates to certificates (Section 5.2). Then, a two-step procedure is discussed to (a) evaluate the instance (Section 5.3), and (b) construct a certificate on the basis of the steps taken in the evaluation (Section 5.4).

5.1 Symbolic Formulas via Skolemization

The Skolem theorem shows how to transform any given *First Order Logic* (FOL) statement F into a *skolemized* formula $Sk(F)$ that has two properties: (1) $Sk(F)$ contains no existential quantifier, and (2) $Sk(F)$ is satisfiable iff F is satisfiable. Existential quantifiers are eliminated by replacing the

variables they bind with *Skolem functions* whose definition domains are appositely chosen to preserve satisfiability. In the *outer* form of skolemization, the function introduced for $e \in \text{var}_\exists(F)$ depends on the universal variables $\text{var}_\forall(F, e)$ that have e in their scope (for prenex formulas: all the universal variables to the left of e in the prefix).

Skolemization-based solvers replace the original formula F with the satisfiability-equivalent instance $Sk(F)$. Such instance is no longer propositional. Nevertheless, we are able to capture its semantics without exceeding the expressive power of propositional logic, by explicitly managing the truth values of the (interpretation of) skolem terms in each point of their definition domains, as shown in [Benedetti, 2005].

The duty we pay is a (possibly) exponential blowup in the size of the problem. BDDs come out to be precious in keeping this space explosion problem under control: What we actually manage is a *symbolic formula*, i.e. a compact BDD-based representation of a propositional instance representing the *definability* of the set of skolem terms introduced in $Sk(F)$.

Let us denote by $\Psi|_k$ the k -bit long prefix of $\Psi \in \mathfrak{B}^n, n \geq k$. The notion is extended to sets: $\mathcal{I}|_k = \{\Psi|_k \mid \Psi \in \mathcal{I}\}$.

Definition 5.1 (Symbolic formula) A symbolic formula $\mathcal{F}$ is a BDD-based representation of a CNF instance. It consists of a symbolic prefix $[e_1]_{\delta_1} \dots [e_m]_{\delta_m}$ on the variables $\text{var}(\mathcal{F}) = \{e_1, \dots, e_m\}$, with $0 \leq \delta_1 \leq \dots \leq \delta_m$, followed by a symbolic matrix $\tilde{\mathcal{F}}$, i.e. a conjunction of symbolic clauses. A symbolic clause $\Gamma_{\mathcal{I}}$ is made up by a consistent set $\Gamma = [\varphi_1 \otimes e_{i_1}, \dots, \varphi_h \otimes e_{i_h}]$ of literals on $\text{var}(\mathcal{F})$, and a (BDD represented) subset $\mathcal{I}$ of $\mathfrak{B}^{\delta(\Gamma)}$, $\delta(\Gamma) \doteq \max_{l \in \Gamma} \delta(l)$. The CNF represented by $\mathcal{F}$ is called *propositional expansion* of $\mathcal{F}$. It has variables $\{s_\Phi^{(i)}, i \in [1, m], \Phi \in \mathfrak{B}^{\delta(e_i)}\}$, and is defined as $\text{Prop}(\mathcal{F}) \doteq \bigwedge_{\Gamma \in \mathcal{F}} \text{Prop}(\Gamma_{\mathcal{I}})$, where

$$\text{Prop}(\Gamma_{\mathcal{I}}) \doteq \bigwedge_{\Psi \in \mathcal{I}} \varphi_1 \otimes s_{\Psi|_{\delta_1}}^{(1)} \vee \dots \vee \varphi_m \otimes s_{\Psi|_{\delta_m}}^{(m)}$$

Symbolic formulas inherit the semantics of their propositional expansions (which we also call *ground* counterparts). Noticeably, a consistent set $\mathcal{M} = \{[l_1]_{\mathcal{I}_1}, \dots, [l_m]_{\mathcal{I}_m}\}$ of symbolic literals satisfies $\mathcal{F}$ ($\mathcal{M} \models \mathcal{F}$) iff its expansion $\text{Prop}(\mathcal{M}) = \bigcup_{[l]_{\mathcal{I}} \in \mathcal{M}} \text{Prop}([l]_{\mathcal{I}})$ satisfies $\text{Prop}(\mathcal{F})$.

Definition 5.2 (Symbolic skolemization) The symbolic skolemization $\text{SymbSk}(F)$ of a QBF F with $\text{var}_\exists(F) = \{e_1, \dots, e_m\}$ is a symbolic formula with prefix $[e_1]_{\delta(e_1)} \dots [e_m]_{\delta(e_m)}$, having one symbolic clause $[l_1, \dots, l_h]_{\mathcal{I}}$ for each clause $\Lambda \in F$, where $\{l_1, \dots, l_h\}$ are the existential literals in Λ , $\{\varphi_1 \otimes e_{i_1}, \dots, \varphi_h \otimes e_{i_h}\}$ are the universal literals in Λ , and $\mathcal{I} = \{\langle \psi_i, \dots, \psi_k \rangle \in \mathfrak{B}^k \mid \forall j. \psi_{i_j} \neq \varphi_j\}$, $k = \delta(\Lambda)$.

As an example, the symbolic skolemization of (1) is given in Figure 2. In essence, a symbolic skolemization $\mathcal{F} = \text{SymbSk}(F)$ is a compact representations for a purely existential instance $\text{Prop}(\mathcal{F})$ having the following key property.

Theorem 5.1 For any QBF F , $\text{Prop}(\text{SymbSk}(F)) \stackrel{\text{sat}}{=} F$.

Proof sketch. By applying outer skolemization as described in [Benedetti, 2005] we turn the QBF instance F into a sat-equivalent purely universal formula: We substitute every existential variable v dominated by $\{u_1, u_2, \dots, u_n\} \subseteq$

$\text{var}_\forall(F)$ with a *Skolem function* $s^v(u_1, u_2, \dots, u_n)$. For example, the instance (1) with matrix $\tilde{N}$, is sat-equivalent to

$$\forall a \forall b \forall d. \tilde{N}[s^c(a, b)/c, s^e(a, b, d)/e, s^f(a, b, d)/f] \quad (2)$$

Then, we propositionally encode the *definability* of the Skolem terms introduced, leveraging a noteworthy feature: They all map $\mathfrak{B}^n$ onto $\mathfrak{B}$ (for some $n \geq 0$), hence they are fully specified by 2^n boolean parameters, and have a direct CNF representation. For example, if we denote by $\{s_{\alpha\beta}^c, \langle \alpha, \beta \rangle \in \mathfrak{B}^2\}$ the four boolean parameters representing the truth value of s^c over $\langle x, y \rangle$, we obtain for $s^c(a, b)$ the following *propositional skolemization* $Sk(s^c) \equiv s^c(a, b)$:

$$(a \vee b \vee s_{00}^c) \wedge (a \vee \neg b \vee s_{01}^c) \wedge (\neg a \vee b \vee s_{10}^c) \wedge (\neg a \vee \neg b \vee s_{11}^c)$$

By replacing each $v \in \text{var}_\exists(F)$ with $Sk(s^v)$ we obtain $Sk(F)$

$$\exists s^c \exists s^e \exists s^f \forall a \forall b \forall d. \tilde{N}[Sk(s^c)/c, Sk(s^d), Sk(s^e)/e]$$

where $\exists s^c$ stands for $\exists s_{00}^c \exists s_{01}^c \exists s_{10}^c \exists s_{11}^c$, and similarly for the other functions. This formula is easily turned—clause by clause—into a CNF. By distributing the connectives, removing clauses with complementary literals, and eliminating literals over universal quantifiers, we obtain $2^{\delta(\Gamma)-m}$ clauses out of a QBF clause Γ with m universal literals. For example, we obtain $2^{3-1}=4$ clauses from the clause $a \vee c \vee f$ in (1):

$$(s_{00}^c \vee s_{000}^f) \wedge (s_{00}^c \vee s_{001}^f) \wedge (s_{01}^c \vee s_{010}^f) \wedge (s_{01}^c \vee s_{011}^f)$$

All the clauses coming out of a given QBF clause mention the same skolem functions. What makes one differ from another are the subscripted indexes. This allows us to write them compactly by representing function names apart from indexes. For example, the four clauses above may be succinctly represented as $\Gamma_{\mathcal{I}} = [c, f]_{\{000, 001, 010, 011\}}$, where $\mathcal{I} \subseteq \mathfrak{B}^{\delta(\Gamma)}$ contains one element per clause, and the i -th component of each $\Psi \in \mathcal{I}$ refers to the universal variable u_i : Once $\Psi \in \mathcal{I}$ is selected, each literal l obtains its own subscript by *projecting* Ψ onto the subspace related to the first $\delta(l)$ components, written $\Psi|_{\delta(l)}$. For example, given $\Psi = 010$ it is $\Psi|_{\delta(c)} = \Psi|_2 = 01$ and $\Psi|_{\delta(f)} = \Psi|_3 = 010$, hence $[c, f]_{\{010\}} = s_{01}^c \vee s_{010}^f$. This property allows us to recover the ground meaning of a factored clause through the *Prop* function. The factored representation becomes *symbolic* as soon as we represent and manipulate sets of indexes via BDDs over $\text{var}_\forall(F)$, thus obtaining the (linear-size) representation $\text{SymbSk}(F)$. $\square$

For example, the propositional expansion of the formula in Figure 2 yields a CNF instance equivalent to (1).

5.2 From symbolic models to certificates

The connection between a symbolic model and a certificate is so close that the former is smoothly turned into the latter. Let $i(\mathcal{M}, l) \doteq \bigcup_{[l]_{\mathcal{I}} \in \mathcal{M}} \mathcal{I}$ be the indexes on the literal l mentioned in a symbolic model $\mathcal{M}$.

Theorem 5.2 If $\mathcal{M} \models \text{SymbSk}(F)$, then the certificate

$$\mathcal{C}(\mathcal{M}) \doteq [\langle \mathcal{C}_{e_1}^+, \mathcal{C}_{e_1}^- \rangle, \langle \mathcal{C}_{e_2}^+, \mathcal{C}_{e_2}^- \rangle, \dots, \langle \mathcal{C}_{e_m}^+, \mathcal{C}_{e_m}^- \rangle]$$

with $\mathcal{C}_e^+ = i(\mathcal{M}, e)$, and $\mathcal{C}_e^- = i(\mathcal{M}, \neg e)$ is valid for F .

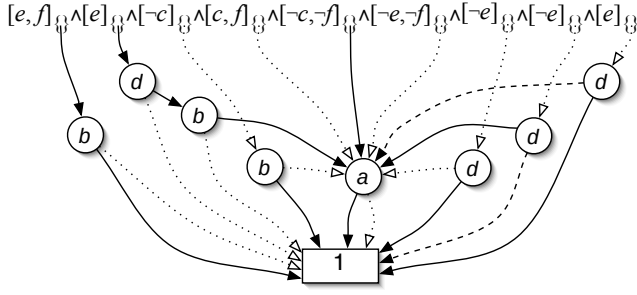


Figure 2: Symbolic propositional skolemization for (1).

Proof. The propositional variable s_{Ψ}^i in $Prop(\mathcal{F})$, $\mathcal{F} = SymbSk(F)$ represents, for each $i \in [1, m]$ and each $\Psi = \langle \psi_1, \dots, \psi_k \rangle \in \mathfrak{B}^k$, $k = \delta(e_i)$, the value the Skolem function $s^{(i)}(u_1, \dots, u_k)$ (introduced to skolemize $e_i \in var_{\exists}(F)$) assumes under the universal hypothesis $[u_1 = \psi_1, \dots, u_k = \psi_k]$. By construction, $\mathcal{F}$ embodies the definability of the skolem terms, i.e. all the mutual constraints among (interpretation of) terms that have to be obeyed to always satisfy the matrix. A model $\mathcal{M}$ for $\mathcal{F}$ is a way to comply with all these constraints at once: $s^{(i)}(\psi_1, \dots, \psi_k)$ has to evaluate to 1 if $s_{\Psi}^{(i)} \in Prop(\mathcal{M})$, to 0 if $\neg s_{\Psi}^{(i)} \in Prop(\mathcal{M})$, and is unconstrained otherwise. By comparing the resulting functions

$$s^e(\psi_1, \dots, \psi_k) = \begin{cases} 1 & \text{if } \langle \psi_1, \dots, \psi_k \rangle \in \mathcal{C}_e^+ \\ 0 & \text{if } \langle \psi_1, \dots, \psi_k \rangle \in \mathcal{C}_e^- \\ \text{undefined} & \text{otherwise} \end{cases}$$

with Definition 3.1 (valid certificates) the thesis follows. $\square$

5.3 Evaluating symbolic formulas

We devote our attention to *solution-based* procedures [Biere, 2004; Pan and Vardi, 2004; Benedetti, 2005], as they may be lifted to extract models *symbolically*. We consider a solver that manipulates symbolic clauses through rules designed to achieve at once on $\Gamma_{\mathcal{I}}$ the same result ground rules would obtain if applied separately to each clause in $Prop(\Gamma_{\mathcal{I}})$.

- **Resolution:** Given two *resolving* clauses containing the same variable e in opposite polarities, we construct a necessary consequence—called *resolvent* clause—made up by all the literals from both originating clauses apart from e and $\neg e$. For the symbolic version to produce at most one symbolic resolvent per step, we only resolve on variables with the same universal depth as all the clauses in which they appear. The resolvent of $\Gamma_{\mathcal{I}}$ and $\Gamma'_{\mathcal{I}}$ on e , with $e \in \Gamma$, $\neg e \in \Gamma'$, $\delta(\Gamma) = \delta(\Gamma') = \delta(e)$, is².

$$(\Gamma \setminus \{e\} \cup \Gamma' \setminus \{\neg e\})_{\mathcal{I} \cap \mathcal{I}'} \quad (3)$$

If either $\mathcal{I} \cap \mathcal{I}' = \emptyset$ or Γ and Γ' share further couples of complementary literals other than e and $\neg e$, the resolvent represents an empty set of clauses and is removed.

- **Subsumption:** $\Gamma_{\mathcal{I}}$ is subsumed by $\Gamma'_{\mathcal{I}'}$ when all the ground clauses in $\Gamma_{\mathcal{I}}$ are subsumed by some clause in

²To simplify the notation, we omit the projection of the index sets of derived clauses onto the proper subspaces: Whenever $\Gamma_{\mathcal{I}}$ is derived, the clause actually inserted into the formula is $\Gamma_{\mathcal{I}|\delta(\Gamma)}$.

$\Gamma'_{\mathcal{I}'}$, i.e. when $\Gamma' \subseteq \Gamma$ and $\mathcal{I} \subseteq \mathcal{I}'$. Under this condition, $\Gamma_{\mathcal{I}}$ can be removed. In *partial* subsumption ($\Gamma' \subseteq \Gamma$ but $\mathcal{I} \not\subseteq \mathcal{I}'$) the subsumed clause is replaced by $\Gamma_{\mathcal{I} \cap \mathcal{I}'}$.

- **Substitution:** We write $[l/a]_{\mathcal{J}}$, $\delta(l) \leq \delta(a)$, to denote the substitution of $l_{\Psi|\delta(\phi)}$ for a_{Ψ} for each $\Psi \in \mathcal{J}$:

$$\Gamma_{\mathcal{I}}[l/a]_{\mathcal{J}} = (\Gamma[l/a])_{\mathcal{I} \cap \mathcal{J}} \wedge \Gamma_{\mathcal{I} \cap \bar{\mathcal{J}}} \quad (4)$$

The rules above establish primitive manipulation capabilities for the solver. It arranges these components into the following higher-level, satisfiability-preserving transformations.

Assignment. When $[l]_{\mathcal{J}}$ is realized to be a consequence of $\mathcal{F}$ (i.e.: for all $\Psi \in \mathcal{J}$, it is $Prop(\mathcal{F}) \vdash l_{\Psi}$), the formula is simplified to $\mathcal{F} * [l]_{\mathcal{J}}$ by *assigning* $[l]_{\mathcal{J}}$, i.e. resolving against $[\neg l]_{\mathcal{J}}$ and subsuming against $[l]_{\mathcal{J}}$.

Equivalence reasoning. If the solver discovers a proof that $[a] \xrightarrow{\mathcal{J}} [l]$ (i.e.: $\forall \Psi \in \mathcal{J}, Prop(\mathcal{F}) \vdash a_{\Psi|\delta(a)} \leftrightarrow l_{\Psi}$), it is entitled to simplify $\mathcal{F}$ by applying the substitution $[l/a]_{\mathcal{J}}$.

Redundancy removal. Subsumption is applied to eliminate subsumed clauses. This may heavily reduce the burden on the solver, while logical equivalence is preserved.

Variable elimination. The clauses containing a variable d are replaced by the set of resolvents of every clause containing d against each clause containing $\neg d$ (Figure 3). Symbolic elimination rules out one symbolic variable, hence all the related ground variables at once.

As far as we are concerned, a solution-based evaluation procedure is an algorithm that produces a (memory and time affordable) sequence of instantiations of the above rules, and guarantees to end up with the empty formula on **sat** instances, and with a contradiction (an empty clause) on **unsat** formulas. Each step is a back-chaining reduction from a problem $\mathcal{F}'$ to a problem $\mathcal{F}$. Hence, a model for the original instance is *not* directly extracted. Rather, satisfiability equivalence guarantees that at each step, should $\mathcal{F}$ have a model, then a model for $\mathcal{F}'$ could be (easily) derived.

This apparent drawback suddenly turns into an advantage. It indeed allows to *decouple* evaluation from model reconstruction, with almost no overhead for the former and a clear semantics for the latter. The two meshes of the chain are connected through an *inference log*, produced by the solver, and subsequently read by a *model reconstructor*.

Definition 5.3 (Inference log and trace) An inference log is a list of entries, each one describing an instantiation of one of the above *sat*-preserving transformations. It contains:

- $\langle \text{assign}, l, \mathcal{J} \rangle$, for an assignment $[l]_{\mathcal{J}}$.
- $\langle \text{subst}, a, l, \mathcal{J}, \mathcal{G} \rangle$, for a substitution $[l/a]_{\mathcal{J}}$, where $\mathcal{G}$ is the set of clauses containing $\{a, \neg l\}$ or $\{\neg a, l\}$.
- $\langle \text{elim}, d, \mathcal{G}^+, \mathcal{G}^- \rangle$, for the elimination of d , where $\mathcal{G}^+$ and $\mathcal{G}^-$ are the sets of clauses containing d and $\neg d$.

Let us denote with $\mathcal{I}(\mathcal{F}, \text{op})$ the formula obtained by applying to $\mathcal{F}$ the transformation described by the entry **op**. An inference log $\mathcal{L} = [\text{op}_1, \text{op}_2, \dots, \text{op}_t]$ induces an inference trace $[\mathcal{F}_0, \mathcal{F}_1, \dots, \mathcal{F}_t]$ where $\mathcal{F}_i = \mathcal{I}(\mathcal{F}_{i-1}, \text{op}_i)$, with $\mathcal{F}_0 = \mathcal{F}$. A log such that $\mathcal{F}_t$ is empty is called **sat-log** for $\mathcal{F}$.

Clearly, a **sat-log** for $\mathcal{F}$ exists iff $\mathcal{F}$ is **sat**. For example, the reader may verify that Figure (4) depicts a **sat-log** for (1).

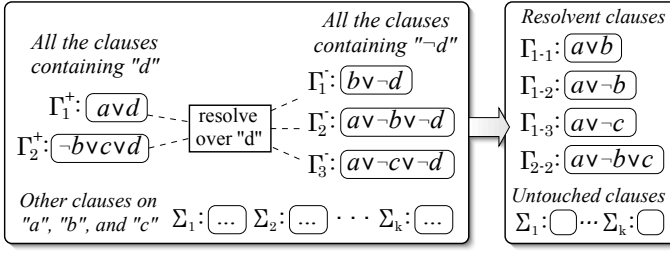


Figure 3: Moving from F' to F by eliminating the variable d .

5.4 Inductive model reconstruction

Once a sat-log is known, the reconstructor comes into play. It trusts the solver about the log being a sat-log, and parses it backward, reasoning by induction on the number of entries:

Base case. At the end of the inference trace we find the empty formula $\mathcal{F}_t$, satisfied by an empty model $\mathcal{M}_t$.

Inductive case. Given a model $\mathcal{M}_i$ for $\mathcal{F}_i$, the reconstructor computes a model $\mathcal{M}_{i-1} = \mathcal{R}(\mathcal{M}_i, \text{op}_i)$ for $\mathcal{F}_{i-1}$ by reasoning on how op_i turned $\mathcal{F}_{i-1}$ into $\mathcal{F}_i$.

This leads to a model $\mathcal{M}_0$ for $\mathcal{F}$, hence to a certificate for F , once the function $\mathcal{R}$ has been properly defined. Let us denote by $i(\mathcal{F}) \doteq \bigcup_{\mathcal{I} \in \mathcal{F}} \mathcal{I}$ the set of indexes mentioned in $\mathcal{F}$.

Theorem 5.3 *When applied to any sat-log for $\mathcal{F}$, the inductive model reconstruction procedure defined by the following function $\mathcal{R}$ computes a model $\mathcal{M}_0$ for $\mathcal{F}_0 = \mathcal{F}$.*

op	$\mathcal{R}(\mathcal{M}, \text{op})$
$\langle \text{assign}, l, \mathcal{I} \rangle$	$\mathcal{M} \cup \{[l]_{\mathcal{I}}\}$
$\langle \text{subst}, e, l, \mathcal{I}, \mathcal{G} \rangle$	$\text{ext}^s(\mathcal{M} \cup \{[e]_{\mathcal{I} \cap i(\mathcal{M}, l)}, [\neg e]_{\mathcal{I} \cap i(\mathcal{M}, \neg l)}\})$
$\langle \text{elim}, e, \mathcal{G}^+, \mathcal{G}^- \rangle$	$\mathcal{M}^e \cup \{[e]_{i(\mathcal{G}^+ * \mathcal{M}^e)}, [\neg e]_{i(\mathcal{G}^- * \mathcal{M}^e)}\}$

where $\text{ext}^s(\mathcal{M}) \doteq \mathcal{M} \cup [e]_{i(\mathcal{G} * \mathcal{M})}$, and $\mathcal{M}^e \doteq \text{ext}^e(\mathcal{M})$, with

$$\text{ext}^e(\mathcal{M}) = \begin{cases} \text{ext}(\mathcal{M} \cup [v]_{\mathcal{I}}) & \text{if } \exists (\mathcal{I}_L, \mathcal{I}_R) \in \mathcal{G}^+ * \mathcal{M} \times \mathcal{G}^- * \mathcal{M} \\ & \text{with } e \neq v, v \in \mathcal{I} \cap \mathcal{I}_L, \mathcal{I} \cap \mathcal{I}_R \neq \emptyset \\ \mathcal{M}, & \text{otherwise} \end{cases}$$

Proof. For each op we show that $\mathcal{R}(\mathcal{M}, \text{op}) \models \mathcal{F}'$, working under the inductive hypothesis of knowing a model $\mathcal{M}$ for $\mathcal{F} = \mathcal{I}(\mathcal{F}', \text{op})$ (i.e. $\mathcal{F} * \mathcal{M}$ is the empty formula).

Assignment. $\mathcal{F} * \mathcal{M} = (\mathcal{F}' * [l]_{\mathcal{I}}) * \mathcal{M} = \mathcal{F}' * (\{[l]_{\mathcal{I}}\} \cup \mathcal{M})$ is empty, hence $\{[l]_{\mathcal{I}}\} \cup \mathcal{M}$ satisfies $\mathcal{F}'$.

Equivalence. Let $\mathcal{F}'$ be $\mathcal{F}'_0 \cup \mathcal{F}'_1 \cup \mathcal{F}'_{2+} \cup \mathcal{F}'_{2-}$, where no clause in $\mathcal{F}'_0$ mentions e , all the clauses in $\mathcal{F}'_1$ mention e but not $l, \neg l$; all the clauses in $\mathcal{F}'_{2+}$ contain both e and l (or $\neg e$ and $\neg l$), and the clauses in $\mathcal{F}'_{2-}$ contain e and $\neg l$ (or $\neg e$ and l). $\mathcal{M}$ satisfies $\mathcal{F} = \mathcal{F}'[l/e]_{\mathcal{I}} = \mathcal{F}'_0 \cup (\mathcal{F}'_1 \cup \mathcal{F}'_{2+})[l/e]_{\mathcal{I}}$, hence it satisfies $\mathcal{F}'_0 = \mathcal{F}_0$ with no modification. It also satisfies $\mathcal{F}'_1 \cup \mathcal{F}'_{2+}$ provided we mirror on e the assignments collected so far over l , by adding $[e]_{\mathcal{I} \cap i(\mathcal{M}, l)}$ and $[\neg e]_{\mathcal{I} \cap i(\mathcal{M}, \neg l)}$. In general, the resulting model $\mathcal{M}^+ = \mathcal{M} \cup \{[e]_{\mathcal{I} \cap i(\mathcal{M}, l)}, [\neg e]_{\mathcal{I} \cap i(\mathcal{M}, \neg l)}\}$ satisfies all the clauses in $\mathcal{F}'$ but some in $\mathcal{F}'_{2-}$: By construction, such clauses failed to pass to $\mathcal{F}$ because $\mathcal{F}'_{2-}[l/e]_{\mathcal{I}} = \mathcal{G}[l/e]_{\mathcal{I}}$ is empty (clauses satisfied by complementary $e/\neg l$ literals). To satisfy the (possibly) nonempty remaining set $\mathcal{G} * \mathcal{M}^+$ of

#1	$\langle \text{substitute}, f, \neg e, \{010, 011\}, \{[e, f]_{\{010, 011, 110, 111\}}, [\neg e, \neg f]_{\{000, 001, 010, 011\}}\} \rangle$
#2	$\langle \text{assign}, \neg e, \{001, 011, 100, 110\} \rangle$
#3	$\langle \text{resolve}, f, \{[f]_{\{110\}}, [c, f]_{\{000, 001\}}, [e, f]_{\{111\}}\}, \{[\neg c, \neg f]_{\{100, 101, 110, 111\}}, [\neg e, \neg f]_{\{000\}}\} \rangle$
#4	$\langle \text{assign}, e, \{111\} \rangle$
#5	$\langle \text{resolve}, c, \{[c]_{\{00, 01\}}, [\neg c]_{\{10, 11\}}\} \rangle$

Figure 4: A sat-log that solves the QBF (1).

clauses, we extend $\mathcal{M}^+$ by applying the ext^s function: It adds an arbitrary truth assignment to all (and only) the indexes for e mentioned in $\mathcal{G} * \mathcal{M}^+$, i.e. it adds $[e]_{i(\mathcal{G} * \mathcal{M}^+)}$.

Variable elimination. We focus on the insightful ground-case proof (see Figure 3). The extension to the symbolic case is a matter of notation. Suppose that the model M for F satisfies at least one clause of each $\langle \Gamma_i^+, \Gamma_j^- \rangle$ couple, $i \in [1, n], j \in [1, m]$. This implies that it either satisfies the whole set $G_d^+ = \bigwedge_i \Gamma_i^+$, or the whole $G_d^- = \bigwedge_j \Gamma_j^-$, or both. Hence, we are free to choose a literal on d in such a way to satisfy F' : $M \cup \{a\}$ satisfies F' when $G_d^+ * M$ is non-empty, $M \cup \{\neg a\}$ when $G_d^- * M$ is non-empty, while M itself suffices when both sets are empty. Now, we show that every model M for F either satisfies at least one clause in each $\langle \Gamma_i^+, \Gamma_j^- \rangle$ couple, or can be extended to a model $\text{ext}^e(M)$ that fulfills this property. The former case happens (at least) when no resolvent clause has been satisfied by complementary literals during variable elimination. In this case, F contains exactly $m * n$ resolvents, one for each couple in $G_d^+ \times G_d^-$. Every model of a resolvent is valid for at least one of its resolving clauses, hence the thesis. Now, suppose that the couple $\langle \Gamma_i^+, \Gamma_j^- \rangle$ is not satisfied by M . It follows that M doesn't model the resolvent of Γ_i^+ and Γ_j^- , hence such resolvent failed to pass to F' : It was satisfied by complementary literals on some still unassigned variable $v \neq d$. A literal on v can be arbitrarily added to the model under construction to satisfy either Γ_i^+ or Γ_j^- (no conflict arises: v would have not appeared as unassigned if involved in any past—w.r.t. the solver's standpoint—inference step). This extension of M to $\text{ext}^e(M)$ is repeated until no unsatisfied couple is left. $\square$

Examples: $M' = \{a, \neg b, c\}$ is a model for $F' = (\neg a \vee c) \wedge (a \vee b \vee \neg c) \wedge (\neg a \vee \neg b)$ because $M = \{a, \neg b\}$ is a model for $F = F' * c = (a \vee b) \wedge (\neg a \vee \neg b)$. $M' = \{\neg a, b, c\}$ is a model for $F' = (\neg a \vee c) \wedge (a \vee b \vee \neg c) \wedge (\neg a \vee \neg b)$ because $M = \{b, c\}$ is a model for $F'[\neg c/a] = c \wedge (b \vee \neg c) \wedge (c \vee \neg b)$. In Figure 3, the assignment $M_1 = \{a, b, c\}$ satisfies F and also F' : d is left unassigned. The model $M_2 = \{a, \neg b\}$ satisfies G_d^+ but not G_d^- , hence $M' = M_2 \cup \{\neg d\}$ is constructed to satisfy F' . The couple $\langle \Gamma_2^+, \Gamma_1^- \rangle$ is untouched by $M_3 = \{a\}$, so no truth value for d helps. But the resolvent of Γ_2^+ and Γ_1^- was satisfied by complementary literals on b , so we construct $\text{ext}^e(M_3) = M_3 \cup \{b\}$, then satisfy F' with $M' = \text{ext}^e(M_3) \cup \{\neg d\} = \{a, b, \neg d\}$. As a complete example, the reader may verify that by performing inductive model reconstruction according to the rules given in Theorem 5.3 over the sat-log in Figure 4 for the $\text{SymbSk}(F)$ in Figure 2, a model is extracted which through Theorem 5.2 produces the sat-certificate in Figure 1.

6 Discussion and Conclusions

We presented a solution to an open question on QBFs, namely the problem of representing, verifying and extracting their sat-certificates. The importance of such solution is twofold: It gives means of conveying solver-independent evidence of satisfiability, and enables the extraction of precious information from certified formulas. The former feature can be used, for example, to effectively test the decisions of QBF solvers. The latter contribution is valuable to applications, in that a certificate is needed to exemplify a definite scenario in which QBF-encoded problems reveal their satisfiability. For example, a sat-answer suffices to know that at least one winning strategy exists in a QBF-encoded two-player game, but it takes a certificate to exhibit an actual strategy.

Stand-alone certificates convey no self-contained semantics, as the meaning of each variable is a piece of information held by the “encoders”. To allow semantics’ owners to interrogate their own QBF models, we have implemented a solver/verifier suite [Benedetti, 2004b; 2004a] to produce, verify, dump to file (in open formats), and query certificates.

This implementation is helping to shed light on further open questions on QBFs. For example: *Is QBF certification impractical?* Let us define the certification of F “impractical” when it is affordable to solve F while it is not feasible to memorize and/or verify any certificate $C(F)$. Our approach suggests that impracticality is not an issue. For instance, the certificates in Table 1 are well within the manipulation capabilities of current machines, while the formulas they certify are hard for present solvers³. Another crucial question we are enabled to address is the following: *Given a certificate C for F , how much in practice the trio $\langle F, C, \text{verify} \rangle$ improves on the couple $\langle F, \text{decide} \rangle$ as a means to prove that F is sat?* Metrics such as the *time · space* product will be used to precisely compare the two strategies. Our results already suggest that the improvement is quite large, as exemplified in Table 1.

We observed a surprising phenomenon: The time taken to reconstruct a model may overcome the time needed to solve the instance. This is rather unusual when compared, for example, with search-based SAT reasoning, where a model is extracted with no overhead on the satisfiability decision. Conversely, we have not yet observed an expected phenomenon: `checkValidity` would operate in polynomial time should we employ a constant-time BDD oracle. The non-polynomiality of verification stems from the size of the forest of BDDs, which should grow exponentially for some parametrically scalable family of instances. This unfavorable phenomenon *doesn’t* show up in Table 1: Certificates scale up polynomially with instance size. These effects and upper bounds on the certificate size will be further investigated.

We showed how to extract certificates using a particular class of QBF solvers (the skolemization-based ones). However, the BDD-based representation we employ is solver-independent. It comes out that not only the mere representation but also the technique for constructing certificates can be lifted to work with other families of QBF-solvers. The

³The “adder” family belongs to a set of benchmarks related to equivalence checking of partial implementations of circuits, and is regarded as extremely challenging [Le Berre *et al.*, 2004]

instance	$\forall$	$\exists$	T_s	T_r	T_v	$ \mathcal{L} $	$ \mathcal{C} $
adder-2	27	37	0.1	0.1	0.1	38	$3.7 \cdot 10^2$
adder-4	106	174	0.2	0.1	0.1	165	$7.1 \cdot 10^3$
adder-6	237	411	0.6	2.3	0.1	384	$5.6 \cdot 10^4$
adder-8	420	748	4.6	35.6	0.4	695	$1.1 \cdot 10^5$
adder-10	755	1185	40.2	537.2	4.6	1098	$4.1 \cdot 10^6$

Table 1: A family of QBF encodings with $\forall\exists\forall\exists$ alternation. We report: the number of existential ($\exists$) and universal ($\forall$) variables, the time taken to solve/reconstruct/verify (T_s, T_r, T_v), the size of the log ($|\mathcal{L}|$, number of steps) and of the certificate ($|\mathcal{C}|$, number of nodes).

key ingredient stays the same: Inductive model reconstruction detached from instance evaluation, with an inference log in between. What changes is the kind of information recorded in the log (and the way it is to be interpreted).

We are extending our technique towards solvers based on (1) non-symbolic q-resolution, (2) SAT reasoning, and (3) symbolic/non-symbolic DPLL-like branching reasoning. The ultimate goal is to build a QBF model reconstructor able to extract certificates by interpreting *generic* QBF inference logs⁴, whichever the evaluation strategy adopted to solve the instance (including hybrid strategies such as the one leveraged by the QBF solver sKizzo [Benedetti, 2004b]).

Acknowledgments

We thank Amedeo Cesta and Gigina Aiello for their comments on how to improve the organization of the paper, Marco Cadoli for helpful discussions on technical issues, and the anonymous referees for their precious remarks.

References

- [Benedetti, 2004a] M. Benedetti. sKizzo: a QBF Decision Procedure based on Propositional Skolemization and Symbolic Reasoning. Technical Report 04-11-03, ITC-irst, 2004.
- [Benedetti, 2004b] M. Benedetti. sKizzo’s web site, sra.itc.it/people/benedetti/sKizzo. 2004.
- [Benedetti, 2005] M. Benedetti. Evaluating QBFs via Symbolic Skolemization. In *Proc. of the 11th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR’04)*, number 3452 in LNCS. Springer, 2005.
- [Biere, 2004] A. Biere. Resolve and Expand. In *Proc. of SAT’04*, 2004.
- [Bryant, 1986] R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transaction on Computing*, C-35(8):677–691, 1986.
- [Büning and Zhao, 2004] H. K. Büning and X. Zhao. On Models for Quantified Boolean Formulas. In *Proc. of SAT’04*, 2004.
- [Gent and Rowley, 2003] I. Gent and A. Rowley. Encoding Connect-4 using Quantified Boolean Formulae. Technical Report 68-2003, APES Research Group, 2003.
- [Le Berre *et al.*, 2004] D. Le Berre, M. Narizzano, L. Simon, and A. Tacchella. Second QBF solvers evaluation, available on-line at www.qbflib.org. 2004.
- [Pan and Vardi, 2004] G. Pan and M.Y. Vardi. Symbolic Decision Procedures for QBF. In *Proc. of the 10th Conf. on Princip. and Practice of Constraint Programming*, 2004.
- [Rintanen, 1999] J. Rintanen. Construction Conditional Plans by a Theorem-prover. *Journal of A. I. Research*, pages 323–352, 1999.
- [Rintanen, 2001] J. Rintanen. Partial implicit unfolding in the davis-putnam procedure for quantified boolean formulae. In *Proceedings of the International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR’01)*, 2001.
- [Stockmeyer and Meyer, 1973] L. J. Stockmeyer and A. R. Meyer. Word Problems Requiring Exponential Time. In *In 5th Annual ACM Symposium on the Theory of Computing*, 1973.
- [Wegener, 2000] Ingo Wegener. *Branching Programs and Binary Decision Diagrams*. Monographs on Discrete Mathematics and Applications. SIAM, 2000.

⁴A standard log format for each inference rule is required.

Optimal and Suboptimal Singleton Arc Consistency Algorithms

Christian Bessière

LIRMM (CNRS / University of Montpellier)
161 rue Ada, Montpellier, France
bessiere@lirmm.fr

Romuald Debruyne

École des Mines de Nantes / LINA
4 rue Alfred Kastler, Nantes, France
romuald.debruyne@emn.fr

Abstract

Singleton arc consistency (SAC) enhances the pruning capability of arc consistency by ensuring that the network cannot become arc inconsistent after the assignment of a value to a variable. Algorithms have already been proposed to enforce SAC, but they are far from optimal time complexity. We give a lower bound to the time complexity of enforcing SAC, and we propose an algorithm that achieves this complexity, thus being optimal. However, it can be costly in space on large problems. We then propose another SAC algorithm that trades time optimality for a better space complexity. Nevertheless, this last algorithm has a better worst-case time complexity than previously published SAC algorithms. An experimental study shows the good performance of the new algorithms.

1 Introduction

Ensuring that a given local consistency does not lead to a failure when we enforce it after having assigned a variable to a value is a common idea in constraint reasoning. It has been applied (sometimes under the name 'shaving') in constraint problems with numerical domains by limiting the assignments to bounds in the domains and ensuring that bounds consistency does not fail [Lhomme, 1993]. In SAT, it has been used as a way to compute more accurate heuristics for DPLL [Freeman, 1995; Li and Ambulagan, 1997]. Finally, in constraint satisfaction problems (CSPs), it has been introduced under the name Singleton Arc Consistency (SAC) in [Debruyne and Bessière, 1997b] and further studied, both theoretically and experimentally in [Prosser *et al.*, 2000; Debruyne and Bessière, 2001].

Some nice properties give to SAC a real advantage over the other local consistencies enhancing the ubiquitous arc consistency. Its definition is much simpler than restricted path consistency [Berlandier, 1995], max-restricted-path consistency [Debruyne and Bessière, 1997a], or other exotic local consistencies. Its operational semantics can be understood by a non-expert of the field. Enforcing it only removes values in domains, and thus does not change the structure of the problem, as opposed to path consistency [Montanari, 1974], k -

consistency [Freuder, 1978], etc. Finally, implementing it can be done simply on top of any AC algorithm.

Algorithms enforcing SAC were already proposed in the past (SAC1 [Debruyne and Bessière, 1997b], and SAC2 [Barták and Erben, 2004]) but they are far from optimal time complexity. Moreover, singleton arc consistency lacks an analysis of its complexity. In this paper, we study the complexity of enforcing SAC (in Section 3) and propose an algorithm, *SAC-Opt*, enforcing SAC with optimal worst-case time complexity (Section 4). However, optimal time complexity is reached at the cost of a high space complexity that prevents the use of this algorithm on large problems. In Section 5, we propose *SAC-SDS*, a SAC algorithm with better worst-case space complexity but no longer optimal in time. Nevertheless, its time complexity remains better than the SAC algorithms proposed in the past. The experiments presented in Section 6 show the good performance of both *SAC-Opt* and *SAC-SDS* compared to previous SAC algorithms.

2 Preliminaries

A *constraint network* P consists of a finite set of n variables $X = \{i, j, \dots\}$, a set of domains $D = \{D(i), D(j), \dots\}$, where the domain $D(i)$ is the finite set of at most d values that variable i can take, and a set of e constraints $C = \{c_1, \dots, c_e\}$. Each constraint c_k is defined by the ordered set $var(c_k)$ of the variables it involves, and the set $sol(c_k)$ of combinations of values satisfying it. A *solution* to a constraint network is an assignment of a value from its domain to each variable such that every constraint in the network is satisfied. When $var(c) = (i, j)$, we use c_{ij} to denote $sol(c)$ and we say that c is *binary*.

A value a in the domain of a variable i (denoted by (i, a)) is consistent with a constraint c_k iff there exists a *support* for (i, a) on c_k , i.e., an instantiation of the other variables in $var(c_k)$ to values in their domain such that together with (i, a) they satisfy c_k . Otherwise, (i, a) is said *arc inconsistent*. A constraint network $P = (X, D, C)$ is *arc consistent* iff D does not contain any arc inconsistent value. $AC(P)$ denotes the network where all arc inconsistent values have been removed from P . If $AC(P)$ has empty domain, we say that P is arc inconsistent.

Definition 1 (Singleton arc consistency) A constraint network $P = (X, D, C)$ is *singleton arc consistent* iff $\forall i \in$

$X, \forall a \in D(i)$, the network $P|_{i=a} = (X, D|_{i=a}, C)$ obtained by replacing $D(i)$ by the singleton $\{a\}$ is not arc inconsistent. If $P|_{i=a}$ is arc inconsistent, we say that (i, a) is SAC inconsistent.

3 Complexity of SAC

Both SAC1 [Debruyne and Bessière, 1997b] and SAC2 [Barták and Erben, 2004] have an $O(en^2d^4)$ time complexity on binary constraints. But it is not optimal.

Theorem 1 *The best time complexity we can expect from an algorithm enforcing SAC on binary constraints is in $O(end^3)$.*

Proof. According to [McGregor, 1979], we know that checking whether a value (i, a) is such that AC does not fail in $P|_{i=a}$ is equivalent to verifying if in each domain $D(j)$ there is at least one value b such that $((i, a), (j, b))$ is “path consistent on (i, a) ”. (A pair of values $((i, a), (j, b))$ is *path consistent on (i, a)* iff it is not deleted when enforcing path consistency only on pairs involving (i, a) .) For each variable j , we have to find a value $b \in D(j)$ such that $((i, a), (j, b))$ is path consistent on (i, a) . In the worst case, all the values b in $D(j)$ have to be considered. So, the path consistency of each pair $((i, a), (j, b))$ may need to be checked against every third variable k by looking for a value (k, c) compatible with both (i, a) and (j, b) . However, it is useless to consider variables k not linked to j since in this partial form of path consistency only pairs involving (i, a) can be removed. By using an optimal time path consistency technique (storing supports), we are guaranteed to consider at most once each value (k, c) when seeking compatible value on k for a pair $((i, a), (j, b))$. The worst-case complexity of proving that path consistency on (i, a) does not fail is thus $\sum_{b \in D(j)} \sum_{c_{kj} \in C} \sum_{c \in D(k)} = ed^2$. Furthermore, enforcing path consistency on (i, a) is completely independent from enforcing path consistency on another value (j, b) . Indeed, a pair $((i, a), (j, b))$ can be path consistent on (i, a) while becoming path inconsistent after the deletion of a pair $((k, c), (j, b))$ thus being path inconsistent on (j, b) . Therefore, the worst case time complexity of any SAC algorithm is at least in $O(\sum_{(i,a) \in D} ed^2) = O(end^3)$. $\square$

4 Optimal Time Algorithm for SAC

SAC1 has no data structure storing which values may become SAC inconsistent when a given value is removed. Hence, it must check again the SAC consistency of all remaining values. SAC2 has data structures that use the fact that if AC does not lead to a wipe out in $P|_{i=a}$ then the SAC consistency of (i, a) holds as long as all the values in $AC(P|_{i=a})$ are in the domain. After the removal of a value (j, b) , SAC2 checks again the SAC consistency of all the values (i, a) such that (j, b) was in $AC(P|_{i=a})$. This leads to a better average time complexity than SAC1 but the data structures of SAC2 are not sufficient to reach optimality since SAC2 may waste time re-enforcing AC in $P|_{i=a}$ several times from scratch.

Algorithm 1, called SAC-Opt, is an algorithm that enforces SAC in $O(end^3)$, the lowest time complexity which can be expected (see Theorem 1).

The idea behind such an optimal algorithm is that we don’t want to do and redo (potentially nd times) arc consistency

Algorithm 1: The optimal SAC algorithm

```

function SAC-Opt(inout  $P$ : Problem): Boolean;
  /* init phase */;
  1 if AC( $P$ ) then PendingList  $\leftarrow \emptyset$  else return false;
  2 foreach  $(i, a) \in D$  do
  3    $P_{ia} \leftarrow P$  /* copy only domains and data structures */;
  4    $D_{ia}(i) \leftarrow \{a\}$ ;
  5   if  $\neg \text{propagAC}(P_{ia}, D(i) \setminus \{a\})$  then
  6      $D \leftarrow D \setminus \{(i, a)\}$ ;
  7     if  $\text{propagAC}(P, \{(i, a)\})$  then
  8       foreach  $P_{jb}$  such that  $(i, a) \in D_{jb}$  do
  9          $Q_{jb} \leftarrow Q_{jb} \cup \{(i, a)\}$ ;
 10        PendingList  $\leftarrow$  PendingList  $\cup \{(j, b)\}$ ;
 11      else return false;

  /* propag phase */;
 12 while PendingList  $\neq \emptyset$  do
 13   pop  $(i, a)$  from PendingList;
 14   if  $\text{propagAC}(P_{ia}, Q_{ia})$  then  $Q_{ia} \leftarrow \emptyset$ ;
 15   else
 16      $D \leftarrow D \setminus \{(i, a)\}$ ;
 17     if  $D(i) = \emptyset$  then return false;
 18     foreach  $(j, b) \in D$  such that  $(i, a) \in D_{jb}$  do
 19        $D_{jb}(i) \leftarrow D_{jb}(i) \setminus \{a\}$ ;  $Q_{jb} \leftarrow Q_{jb} \cup \{(i, a)\}$ ;
 20       PendingList  $\leftarrow$  PendingList  $\cup \{(j, b)\}$ ;
 21 return true;

```

from scratch in each subproblem $P|_{j=b}$ each time a value (i, a) is found SAC inconsistent. (Which represents n^2d^2 potential arc consistency calls.) To avoid such costly repetitions of arc consistency calls, we duplicate the problem nd times, one for each value (i, a) , so that we can benefit from the incrementality of arc consistency on each of them. All generic AC algorithms are incremental, i.e., their complexity on a problem P is the same for a single call or for up to nd calls, where two consecutive calls differ only by the deletion of some values from P .

In the following, P_{ia} is the subproblem in which SAC-Opt stores the current domain (noted D_{ia}) and the data structure corresponding to the AC enforcement in $P|_{i=a}$. $\text{propagAC}(P, Q)$ denotes the function that incrementally propagates the removal of set Q of values in problem P when an initial AC call has already been executed, initialising the data structure required by the AC algorithm in use. It returns false iff P is arc inconsistent.

SAC-Opt is composed of two main steps. After making the problem arc consistent (line 1), the loop in line 2 takes each value a in each domain $D(i)$ and creates the problem P_{ia} as a copy of the current P . Then, in line 5 the removal of all the values different from a for i is propagated. If a subproblem P_{ia} is arc inconsistent, (i, a) is pruned from the ‘master’ problem P and its removal is propagated in P (line 7). The advantage of propagating immediately in the master problem the SAC inconsistent values found in line 5 is twofold. First, the subproblem P_{kc} of a value (k, c) removed in line 7 before its selection in line 2 will never be created. Second, the subproblems P_{jb} created after the removal of (k, c) will benefit from this propagation since they are created by duplication of P (line 3). For each already created subproblem P_{jb} with $a \in D_{jb}(i)$, (i, a) is put in Q_{jb} and P_{jb} is put in PendingList

for future propagation (lines 9–10).

Once the initialisation phase has finished, we know that *PendingList* contains all values (i, a) for which some SAC inconsistent value removals (contained in Q_{ia}) have not yet been propagated in P_{ia} . The loop in line 12 propagates these removals (line 14). When propagation fails, this means that (i, a) itself is SAC inconsistent. It is removed from the domain D of the master problem in line 16 and each subproblem P_{jb} containing (i, a) is updated together with its propagation list Q_{jb} for future propagation (lines 18–20).

When *PendingList* is empty, all removals have been propagated in the subproblems. So, all the values in D are SAC.

Theorem 2 *SAC-Opt is a correct SAC algorithm with $O(end^3)$ optimal time complexity and $O(end^2)$ space complexity on binary constraints.*

Proof. Soundness. A value a is removed from $D(i)$ either in line 6 or in line 16 because P_{ia} was found arc inconsistent. If P_{ia} was found arc inconsistent in the initialisation phase (line 5), SAC inconsistency of (i, a) follows immediately. Let us proceed by induction for the remaining case. Suppose all values already removed in line 16 were SAC inconsistent. If P_{ia} is found arc inconsistent in line 14, this means that P_{ia} cannot be made arc consistent without containing some SAC inconsistent values. So, (i, a) is SAC inconsistent and SAC-Opt is sound.

Completeness. Thanks to the way *PendingList* and Q_{ia} are updated in lines 8–10 and 18–20, we know that at the end of the algorithm, all P_{ia} are arc consistent. Thus, for any value (i, a) remaining in P at the end of SAC-Opt, $P_{i=a}$ is not arc inconsistent and (i, a) is therefore SAC consistent.

Complexity. The complexity analysis is given for networks of binary constraints since the optimality proof was given for networks of binary constraints only. (But SAC-Opt works on constraints of any arity.) Since any AC algorithm can be used to implement our SAC algorithm, the space and time complexities will obviously depend on this choice. Let us first discuss the case where an optimal time algorithm such as AC-6 [Bessière, 1994] or AC2001 [Bessière and Régin, 2001; Zhang and Yap, 2001] is used. Line 3 tells us that the space complexity will be nd times the complexity of the AC algorithm, namely $O(end^2)$. Regarding time complexity, the first loop copies the data structures and propagates arc consistency on each subproblem (line 5), two tasks which are respectively in $nd \cdot ed$ and $nd \cdot ed^2$. In the *while* loop (line 12), each subproblem can be called nd times for arc consistency, and there are nd subproblems. Now, AC-6 and AC2001 are both incremental, which means that the complexity of nd restrictions on the same problem is ed^2 and not $nd \cdot ed^2$. Thus the total cost of arc consistency propagation on the nd subproblems is $nd \cdot ed^2$. We have to add the cost of updating the lists in lines 8 and 18. In the worst case, each value is removed one by one, and thus, nd values are put in nd Q lists, leading to $n^2 d^2$ updates of *PendingList*. If $n < e$, the total time complexity is $O(end^3)$, which is optimal. $\square$

Remarks. We chose to present the Q_{ia} lists as lists of values to be propagated because the AC algorithm used is not specified here, and value removal is the most accurate information we can have. When using a fine-grained AC algorithm,

such as AC-6 the lists will be used as they are. If we use a coarse-grained algorithm such as AC-3 [Mackworth, 1977] or AC2001, only the variables from which the domain has changed are needed. The modification is direct. We should bear in mind that if AC-3 is used, we decrease space complexity to $O(n^2 d^2)$, but time complexity increases to $O(end^4)$ since AC-3 is non optimal.

5 Losing Time Optimality to Save Space

SAC-Opt cannot be used on large constraint networks because of its $O(end^2)$ space complexity. Moreover, it seems difficult to reach optimal time complexity with smaller space requirements. A SAC algorithm has to maintain AC on nd subproblems $P_{i=a}$, and to guarantee $O(ed^2)$ total time on these subproblems we need to use an optimal time AC algorithm. Since there does not exist any optimal AC algorithm requiring less than $O(ed)$ space, we obtain $nd \cdot ed$ space for the whole SAC process.

We propose to relax time optimality to reach a satisfactory trade-off between space and time. To avoid discussing in too general terms, we instantiate this idea on a AC2001-based SAC algorithm for binary constraints, but the same idea could be implemented with other optimal AC algorithms such as AC-6, and with constraints of any arity. The algorithm SAC-SDS (*Sharing Data Structures*) tries to use as much as possible the incrementality of AC to avoid redundant work, without duplicating on each subproblem $P_{i=a}$ the data structures required by optimal AC algorithms. This algorithm requires less space than SAC-Opt but is not optimal in time.

As in SAC-Opt, for each value (i, a) SAC-SDS stores the local domain D_{ia} of the subproblem $P_{i=a}$ and its propagation list Q_{ia} . Note that in our AC2001-like presentation, Q_{ia} is a list of variables j for which the domain $D_{ia}(j)$ has changed (in SAC-Opt it is a list of values (j, b) removed from D_{ia}). As in SAC-Opt, thanks to these local domains D_{ia} , we know which values (i, a) may no longer be SAC after the removal of a value (j, b) : those for which (j, b) was in D_{ia} . These local domains are also used to avoid starting each new AC propagation phase from scratch, since D_{ia} is exactly the domain from which we should start when enforcing again AC on $P_{i=a}$ after a value removal. By comparison, SAC1 and SAC2 restart from scratch for each new propagation.

But the main idea in SAC-SDS is that as opposed to SAC-Opt, it does not duplicate to all subproblems P_{ia} the data structures of the optimal AC algorithm used. The data structure is maintained only in the master problem P . In the case of AC2001, the *Last* structure which maintains in *Last*(i, a, j) the smallest value in $D(j)$ compatible with (i, a) on c_{ij} is built and updated only in P . Nevertheless, it is used by all subproblems P_{ia} to avoid repeating constraint checks already done in P .

SAC-SDS (Algorithm 2) works as follows: After some initialisations (lines 1–4), SAC-SDS repeatedly pops a value from *PendingList* and propagates AC in P_{ia} (lines 6–9). Note that ' $D_{ia}=nil$ ' means that this is the first enforcement of AC in P_{ia} , so D_{ia} must be initialised (line 8).¹ If P_{ia} is arc in-

¹We included this initialisation step in the main loop to avoid repeating similar lines of code, but a better implementation will put

Algorithm 2: The SAC-SDS algorithm

```

function SAC-SDS-2001(inout  $P$ : problem): Boolean;
1 if AC2001( $P$ ) then PendingList  $\leftarrow \emptyset$  else return false;
2 foreach  $(i, a) \in D$  do
3    $D_{ia} \leftarrow nil$ ;  $Q_{ia} \leftarrow \{i\}$ ;
4   PendingList  $\leftarrow$  PendingList  $\cup \{(i, a)\}$ ;
5 while PendingList  $\neq \emptyset$  do
6   pop  $(i, a)$  from PendingList;
7   if  $a \in D(i)$  then
8     if  $D_{ia} = nil$  then  $D_{ia} \leftarrow (D \setminus D(i)) \cup \{(i, a)\}$ ;
9     if propagSubAC( $D_{ia}, Q_{ia}$ ) then  $Q_{ia} \leftarrow \emptyset$ ;
10    else
11       $D(i) \leftarrow D(i) \setminus \{a\}$ ; Deleted  $\leftarrow \{(i, a)\}$ ;
12      if propagMainAC( $D, \{i\}, Deleted$ ) then
13        updateSubProblems(Deleted)
14      else return false;
15 return true;

function PropagMain/Sub AC(inout  $D$ : domain; in  $Q$ : set ;
    inout Deleted: set): Boolean;
16 while  $Q \neq \emptyset$  do
17   pop  $j$  from  $Q$ ;
18   foreach  $i \in X$  such that  $\exists c_{ij} \in C$  do
19     foreach  $a \in D(i)$  such that  $Last(i, a, j) \notin D(j)$  do
20       if  $\exists b \in D(j), b > Last(i, a, j) \wedge c_{ij}(a, b)$  then
21          $Last(i, a, j) \leftarrow b$ 
22       else
23          $D(i) \leftarrow D(i) \setminus \{a\}$ ;  $Q \leftarrow Q \cup \{i\}$ ;
24         Deleted  $\leftarrow Deleted \cup \{(i, a)\}$ ;
25       if  $D(i) = \emptyset$  then return false;
26 return true;
/*  $\boxed{\dots}$ : in propagMainAC but not in propagSubAC */;

procedure updateSubProblems(in Deleted: set);
27 foreach  $(j, b) \in D \mid D_{jb} \cap Deleted \neq \emptyset$  do
28    $Q_{jb} \leftarrow Q_{jb} \cup \{i \in X \mid D_{jb}(i) \cap Deleted \neq \emptyset\}$ ;
29    $D_{jb} \leftarrow D_{jb} \setminus Deleted$ ;
30   PendingList  $\leftarrow$  PendingList  $\cup \{(j, b)\}$ ;

```

consistent, (i, a) is SAC inconsistent. It is therefore removed from D (line 11) and this deletion is propagated in the master problem P using propagMainAC (line 12). Any value (i, a) removed from P is put in the set Deleted (lines 11 and 24). This set is used by updateSubProblems (line 13) to pass the values removed from P on to the subproblems, and to update the lists Q_{jb} and PendingList for further propagation in modified subproblems.

At this point we have to explain the difference between propagMainAC, which propagates deletions in the master problem P , and propagSubAC, which propagates deletions in subproblems P_{ia} . The parts that are in boxes belong to propagMainAC and not to propagSubAC. Function propagSubAC, used to propagate arc consistency in the subproblems, follows the same principle as the AC algorithm. The only difference comes from the fact that the data structures of the AC algorithm are not modified. When using AC2001, this means that $Last$ is not updated in line 21. Indeed, this data structure is useful to achieve AC in the sub-

it in a former loop as in SAC-Opt.

problems faster (lines 19–20) since we know that there is no support for (i, a) on c_{ij} lower than $Last(i, a, j)$ in P , and so in any subproblem as well. But this data structure is shared by all subproblems. Thus it must not be modified by propagSubAC. Otherwise, $Last(i, a, j)$ would not be guaranteed to be the smallest support in the other subproblems and in P . When achieving AC in P , propagMainAC does update the $Last$ data structure. The only difference with AC2001 is line 24 where it needs to store in Deleted the values removed from D during the propagation. Those removals performed in P can directly be transferred in the subproblems without the effort of proving their inconsistency in each subproblem. This is done via function updateSubProblems, which removes values in Deleted from all the subproblems. It also updates the local propagation lists Q_{ia} and PendingList for future propagation in the subproblems.

Theorem 3 SAC-SDS is a correct SAC algorithm with $O(end^4)$ time complexity and $O(n^2d^2)$ space complexity on binary constraints.

Proof. Soundness. Note first that the structure $Last$ is updated only when achieving AC in P so that any support of (i, a) in $D(j)$ is greater than or equal to $Last(i, a, j)$. The domains of the subproblems being subdomains of D , any support of a value (i, a) on c_{ij} in a subproblem is also greater than or equal to $Last(i, a, j)$. This explains that propagSubAC can benefit from the structure $Last$ without losing any support (lines 19–20). Once this observation is made, soundness comes from the same reason as in SAC-Opt.

Completeness. Completeness comes from the fact that any deletion is propagated. After initialisation (lines 2-4), PendingList = D and so, the main loop of SAC-SDS processes any subproblem $P|_{i=a}$ at least once. Each time a value (i, a) is found SAC inconsistent in P because $P|_{i=a}$ is arc inconsistent (line 9) or because the deletion of some SAC-inconsistent value makes it arc inconsistent in P (line 23 of propagMainAC called in line 12), (i, a) is removed from the subproblems (line 29), and PendingList and the local propagation lists are updated for future propagation (lines 28 and 30). At the end of the main loop, PendingList is empty, so all the removals have been propagated and for any value $(i, a) \in D$, D_{ia} is a non empty arc consistent subdomain of $P|_{i=a}$.

Complexity. The data structure $Last$ requires a space in $O(ed)$. Each of the nd domains D_{ia} can contain nd values and there is at most n variables in the nd local propagation lists Q_{ia} . Since $e < n^2$, the space complexity of SAC-SDS-2001 is in $O(n^2d^2)$. So, considering space requirements, SAC-SDS-2001 is similar to SAC2.

Regarding time complexity, SAC-SDS-2001 first duplicates the domains and propagates arc consistency on each subproblem (lines 8 and 9), two tasks which are respectively in $nd \cdot nd$ and $nd \cdot ed^2$. Each value removal is propagated to all $P|_{i=a}$ problems via an update of PendingList, Q_{ia} and D_{ia} (lines 27–30). This requires $nd \cdot nd$ operations. Each subproblem can in the worst case be called nd times for arc consistency, and there are nd subproblems. The domains of each subproblem are stored so that the AC propagation is launched with the domains in the state in which they were at the end of the

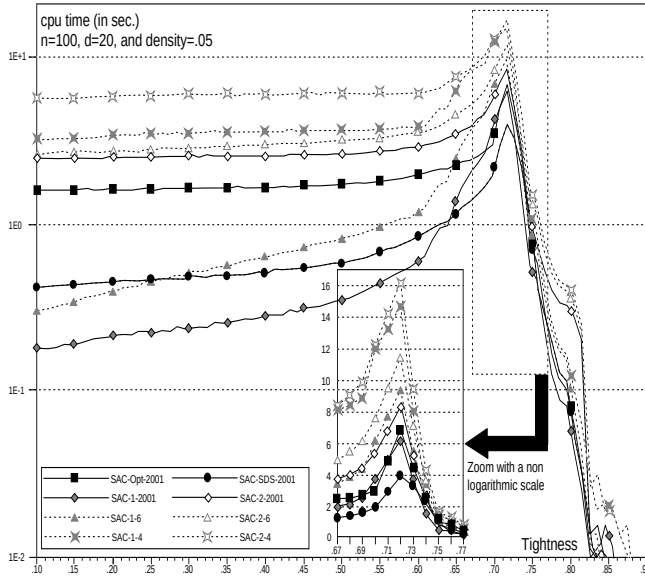


Figure 1: cpu time with $n = 100$, $d = 20$, and density = .05.

previous AC propagation in the subproblem. Thus, in spite of the several AC propagations on a subproblem, a value will be removed at most once and, thanks to incrementality of arc consistency, the propagation of these nd value removals is in $O(ed^3)$. (Note that we cannot reach the optimal ed^2 complexity for arc consistency on these subproblems since we do not duplicate the data structures necessary for AC optimality.) Therefore, the total cost of arc consistency propagations is $nd \cdot ed^3$. The total time complexity is $O(end^4)$. $\square$

As with SAC2, SAC-SDS performs a better propagation than SAC1 since after the removal of a value (i, a) from D , SAC-SDS checks the arc consistency of the subproblems $P|_{j=b}$ only if they have (i, a) in their domains (and not all the subproblems as SAC1). But this is not sufficient to have a better worst-case time complexity than SAC1. SAC2 has indeed the same $O(en^2d^4)$ time complexity as SAC1. SAC-SDS improves this complexity because it stores the *current* domain of each subproblem, and so, does not propagate in the subproblems each time from scratch.² Furthermore, we can expect a better average time complexity since the shared structure *Last* reduces the number of constraint checks required, even if it does not permit to obtain optimal worst-case time complexity. This potential gain can only be assessed by an experimental comparison of the different SAC versions. Finally, in SAC1 and SAC2 each AC enforcement in a subproblem must be done on a new copy of D built at runtime (potentially $nd \cdot nd$ times) while such duplication is performed only once for each value in SAC-SDS (by creating subdomains D_{ia}).

6 Experimental Results

We compared the performance of the SAC algorithms on random uniform constraint networks of binary constraints generated by [Frost *et al.*, 1996], which produces instances accord-

²This is independent on the AC algorithm used and so, enforcing AC with AC-3 preserves this $O(end^4)$ time complexity.

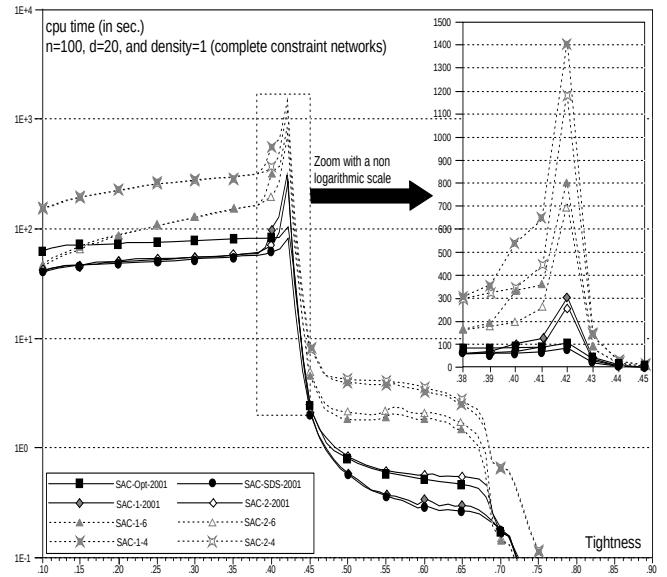


Figure 2: cpu time with $n = 100$, $d = 20$, and density = 1.

ing to Model B [Prosser, 1996]. All algorithms have been implemented in C++ and ran on a Pentium IV-1600 MHz with 512 Mb of memory under Windows XP. SAC1 and SAC2 have been tested using several AC algorithms. In the following, we note SAC-1- X , SAC-2- X and SAC-Opt- X the versions of SAC1, SAC2 and SAC-Opt based on AC- X . Note that for SAC2 the implementation of the propagation list has been done according to the recommendations made in [Barták and Erben, 2003]. For each combination of parameters tested, we generated 50 instances for which we report mean cpu times.

6.1 Experiments on sparse constraint networks

Fig. 1 presents performance on constraint networks having 100 variables, 20 values in each domain, and a density of .05. These constraint networks are relatively sparse since the variables have five neighbours on average.

For a tightness lower than .55, all the values are SAC. On these under-constrained networks, the SAC algorithms check the arc consistency of each subproblem at most once. Storing support lists (as in SAC2) or local subdomains (as in SAC-Opt and SAC-SDS) does not pay-off. A brute-force algorithm such as SAC1 is sufficient. SAC-1-2001 shows the best performance.

On problems having tighter constraints, some SAC inconsistent values are removed and at tightness .72 we can see a peak of complexity. However, as mentioned in [Barták and Erben, 2003], the improved propagation of SAC2 is useless on sparse constraint networks and SAC2- X (with $X \in \{4, 6, 2001\}$) is always more expensive than SAC1- X on the generated problems. Around the peak of complexity, SAC-SDS-2001 is the clear winner. SAC-Opt-2001 and SAC1-2001 are around 1.7 times slower, and all the others are between 2.1 and 10 times slower.

6.2 Experiments on dense constraint networks

Fig. 2 presents performance on constraint networks having 100 variables, 20 values in each domain and a complete graph of binary constraints.

SAC1 and SAC2 show very close performance. When all values are SAC consistent (tightness lower than .37) the additional data structure of SAC2 is useless since there is no propagation. However the cost of building this data structure is not important compared to the overall time and the time required by SAC2 is almost the same as SAC1. Around the peak of complexity, SAC2-X (with $X \in \{4, 6, 2001\}$) requires a little less time than SAC1-X. SAC2 has to repeatedly recheck the arc consistency of less subproblems than SAC1 but the cost of testing a subproblem remains the same. On very tight constraints, SAC1 requires less time than SAC2 since the inconsistency of the problem is found with almost no propagation and building the data structure of SAC2 is useless.

Conversely to what was expected in [Barták and Erben, 2004], using AC-4 in SAC1 (or in SAC2) instead of AC-6 or AC2001 is not worthwhile. The intuition was that since the data structure of AC-4 does not have to be updated, the cost of its creation would be low compared to the profit we can expect. However, SAC1-4 and SAC2-4 are far more costly than the versions based on AC-6 or AC2001.

The best results are obtained with SAC-Opt-2001 and SAC-SDS-2001 which are between 2.6 and 17 times faster than the others at the peak. These two algorithms have a better propagation between subproblems than SAC1 but they also avoid some redundant work and so reduce the work performed on each subproblem.

7 Summary and Conclusion

We have presented SAC-Opt, the first optimal time SAC algorithm. However, the high space complexity of this algorithm prevents its use on large constraint networks. Hence, we have proposed SAC-SDS, a SAC algorithm that is not optimal in time but that requires less space than SAC-Opt. Experiments show the good performance of these new SAC algorithms compared to previous versions available in the literature. This opens again the issue of using SAC as an alternative to AC for pruning values in a constraint network, or at least in 'promising' parts of the network. SAC could also be used to compute variable selection heuristics, as this had been done with success in SAT [Li and Ambulagan, 1997].

References

- [Barták and Erben, 2003] R. Barták and R. Erben. Singleton arc consistency revised. In *ITI Series 2003-153*, Prague, 2003.
- [Barták and Erben, 2004] R. Barták and R. Erben. A new algorithm for singleton arc consistency. In *Proceedings FLAIRS'04*, Miami Beach FL, 2004. AAAI Press.
- [Berlandier, 1995] P. Berlandier. Improving domain filtering using restricted path consistency. In *Proceedings IEEE-CAIA'95*, 1995.
- [Bessière and Régin, 2001] C. Bessière and J.C. Régin. Refining the basic constraint propagation algorithm. In *Proceedings IJCAI'01*, pages 309–315, Seattle WA, 2001.
- [Bessière, 1994] C. Bessière. Arc-consistency and arc-consistency again. *Artificial Intelligence*, 65:179–190, 1994.
- [Debruyne and Bessière, 1997a] R. Debruyne and C. Bessière. From restricted path consistency to max-restricted path consistency. In *Proceedings CP'97*, pages 312–326, Linz, Austria, 1997.
- [Debruyne and Bessière, 1997b] R. Debruyne and C. Bessière. Some practicable filtering techniques for the constraint satisfaction problem. In *Proceedings IJCAI'97*, pages 412–417, Nagoya, Japan, 1997.
- [Debruyne and Bessière, 2001] R. Debruyne and C. Bessière. Domain filtering consistencies. *Journal of Artificial Intelligence Research*, 14:205–230, 2001.
- [Freeman, 1995] J.W. Freeman. *Improvements to propositional satisfiability search algorithms*. PhD thesis, University of Pennsylvania, Philadelphia PA, 1995.
- [Freuder, 1978] E.C. Freuder. Synthesizing constraint expressions. *Communications of the ACM*, 21(11):958–966, Nov 1978.
- [Frost et al., 1996] D. Frost, C. Bessière, R. Dechter, and J.C. Régin. Random uniform csp generators. URL: <http://www.lirmm.fr/~bessiere/generator.html>, 1996.
- [Lhomme, 1993] O. Lhomme. Consistency techniques for numeric csps. In *Proceedings IJCAI'93*, pages 232–238, Chambéry, France, 1993.
- [Li and Ambulagan, 1997] C.M. Li and Ambulagan. Heuristics based on unit propagation for satisfiability problems. In *Proceedings IJCAI'97*, pages 366–371, Nagoya, Japan, 1997.
- [Mackworth, 1977] A.K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8:99–118, 1977.
- [McGregor, 1979] J.J. McGregor. Relational consistency algorithms and their application in finding subgraph and graph isomorphism. *Information Science*, 19:229–250, 1979.
- [Montanari, 1974] U. Montanari. Networks of constraints: Fundamental properties and applications to picture processing. *Information Science*, 7:95–132, 1974.
- [Prosser et al., 2000] P. Prosser, K. Stergiou, and T. Walsh. Singleton consistencies. In *Proceedings CP'00*, pages 353–368, Singapore, 2000.
- [Prosser, 1996] P. Prosser. An empirical study of phase transition in binary constraint satisfaction problems. *Artificial Intelligence*, 81:81–109, 1996.
- [Zhang and Yap, 2001] Y. Zhang and R.H.C. Yap. Making AC-3 an optimal algorithm. In *Proceedings IJCAI'01*, pages 316–321, Seattle WA, 2001.

The Range and Roots Constraints: Specifying Counting and Occurrence Problems*

Christian Bessiere	Emmanuel Hebrard	Brahim Hnich	Zeynep Kiziltan	Toby Walsh
LIRMM-CNRS	NICTA and UNSW	4C, UCC	DEIS	NICTA and UNSW
Montpellier, France	Sydney, Australia	Ireland	Univ. di Bologna, Italy	Sydney, Australia
bessiere@lirmm.fr	e.hebrard@nicta.com.au	brahim@4c.ucc.ie	zkiziltan@deis.unibo.it	tw@cse.unsw.edu.au

Abstract

We propose a simple declarative language for specifying a wide range of counting and occurrence constraints. This specification language is executable since it immediately provides a polynomial propagation algorithm. To illustrate the capabilities of this language, we specify a dozen global constraints taken from the literature. We observe one of three outcomes: we achieve generalized arc-consistency; we do not achieve generalized arc-consistency, but achieving generalized arc-consistency is NP-hard; we do not achieve generalized arc-consistency, but specialized propagation algorithms can do so in polynomial time. Experiments demonstrate that this specification language is both efficient and effective in practice.

1 Introduction

Global constraints are central to the success of constraint programming. Global constraints allow users to specify patterns that occur in many problems, and to exploit efficient and effective propagation algorithms for pruning the search space. Two common types of global constraints are counting and occurrence constraints. Occurrence constraints place restrictions on the occurrences of particular values. For instance, we may wish to ensure that no value used by one set of variables occurs in a second set. Counting constraints, on the other hand, restrict the number of values or variables meeting some condition. For example, we may want to limit the number of distinct values assigned to a set of variables. Many different counting and occurrences constraints have been proposed to help model a wide range of problems, especially those involving resources (see, for example, [Régis, 1994; Beldiceanu and Contejean, 1994; Régis, 1996; Beldiceanu, 2001; Beldiceanu *et al.*, 2004b]).

We will show that many such constraints can be specified using a simple declarative language. This specification language is executable. It decomposes these constraints into some simple primitives for which there are polynomial propagation algorithms. In some cases, we show that this decomposition does not hinder propagation. In other cases, achieving

generalized arc-consistency is NP-hard and decomposition is one method to obtain a polynomial propagation algorithm. In the remaining cases, we show that the decomposition hinders propagation, and there exists some specialized and polynomial propagation algorithm. As not all global constraints are implemented and integrated into every solver, our language may still be attractive in this case. We provide a generic means for propagating counting and occurrence constraints in the absence of a specialized algorithm. Our experiments demonstrate that this is an efficient and effective way to reason about counting and occurrence constraints in practice.

2 Formal Background

A constraint satisfaction problem consists of a set of variables, each with a finite domain of values, and a set of constraints specifying allowed combinations of values for subsets of variables. We use capital letters for variables (e.g. X , Y and S), and lower case for values (e.g. d and d_i). We write $\mathcal{D}(X)$ for the domain of a variable X . A solution is an assignment of values to the variables satisfying the constraints. A variable is ground when it is assigned a value. We consider both integer and set variables. A constraint can be defined on only integer, or only set variables, or on both. A set variable S can be represented by its lower bound $lb(S)$ which contains the definite elements and an upper bound $ub(S)$ which contains the definite and potential elements. Each set variable is equivalent to a sequence of 0/1 variables representing the characteristic function. A variable in such a sequence is 1 iff the corresponding element is in the set.

Given a constraint C , a *bound support* on C is a tuple that assigns to each integer variable a value between its minimum and maximum, and to each set variable a set between its lower and upper bounds which satisfies C . A *hybrid support* on C is a tuple that assigns to each integer variable a value in its domain, and to each set variable a set between its lower and upper bounds which satisfies C . If C involves only integer variables, a hybrid support is a *support*. A constraint C is *bound consistent* (BC) iff for each integer variable X_i , its minimum and maximum values belong to a bound support, and for each set variable S_j , the values in $ub(S_j)$ belong to S_j in at least one bound support and the values in $lb(S_j)$ belong to S_j in all bound supports. A constraint C is *hybrid consistent* (HC) iff for each integer variable X_i , every value in $\mathcal{D}(X_i)$ belongs to an hybrid support, and for each set vari-

*Brahim Hnich is currently supported by Science Foundation Ireland under Grant No. 00/PI.1/C075.

able S_j , the values in $ub(S_j)$ belong to S_j in at least one hybrid support, and the values in $lb(S_j)$ belong to S_j in all hybrid supports. A constraint C involving only integer variables is *generalized arc consistent* (GAC) iff for each variable X_i , every value in $D(X_i)$ belongs to a support.

Enforcing hybrid consistency on a constraint C is equivalent to enforcing GAC if the set variables are represented by their characteristic function. If all variables in C are integer variables, hybrid consistency reduces to generalized arc-consistency, and if all variables in C are set variables, hybrid consistency reduces to bound consistency.

We will compare local consistency properties applied to (sets of) logically equivalent constraints, c_1 and c_2 . As in [Debruyne and Bessière, 1997], a local consistency property Φ on c_1 is as strong as Ψ on c_2 iff, given any domains, if Φ holds on c_1 then Ψ holds on c_2 ; Φ on c_1 is stronger than Ψ on c_2 iff Φ on c_1 is as strong as Ψ on c_2 but not vice versa; Φ on c_1 is equivalent to Ψ on c_2 iff Φ on c_1 is as strong as Ψ on c_2 and vice versa; Φ on c_1 is incomparable to Ψ on c_2 iff Φ on c_1 is not as strong as Ψ on c_2 and vice versa.

A total function $\mathcal{F}$ from a set $\mathcal{S}$ into a set $\mathcal{T}$ is denoted by $\mathcal{F} : \mathcal{S} \rightarrow \mathcal{T}$ where $\mathcal{S}$ is the domain of $\mathcal{F}$ and $\mathcal{T}$ is the range of $\mathcal{F}$. The set of all elements in the domain of $\mathcal{F}$ that have the same image $j \in \mathcal{T}$ is $\mathcal{F}^{-1}(j) = \{i : \mathcal{F}(i) = j\}$. The image of a set $S \subseteq \mathcal{S}$ under $\mathcal{F}$ is $\mathcal{F}(S) = \bigcup_{i \in S} \mathcal{F}(i)$, whilst the domain of a set $T \subseteq \mathcal{T}$ under $\mathcal{F}$ is $\mathcal{F}^{-1}(T) = \bigcup_{j \in T} \mathcal{F}^{-1}(j)$. Throughout, we will view a set of variables, X_1 to X_n as a function $\mathcal{X} : \{1, \dots, n\} \rightarrow \bigcup_{i=1}^n \mathcal{D}(X_i)$. That is, $\mathcal{X}(i)$ is the value of X_i .

3 Specification Language

We propose a simple declarative language in which we can specify many different counting and occurrence constraints. This specification language is executable. It permits us to decompose each global constraint into more primitive constraints, each of which has an associated polynomial propagation algorithm. In some cases, this decomposition does not hinder propagation. In other cases, enforcing local consistency on the global constraint is intractable, and decomposition is one method to obtain a polynomial propagation algorithm. The specification language consists of simple non-global constraints over integer variables (like $X \leq m$), simple non-global constraints over set variables (like $S_1 \subseteq S_2$ or $|S| = k$) and two special global constraints acting on sequences of variables: ROOTS and RANGE.

Given a function $\mathcal{X}$ representing a set of variables, X_1 to X_n , the RANGE constraint holds iff a set variable, T is the range of this function restricted to the indices belonging to a second set variable, S .

$$\text{RANGE}([X_1, \dots, X_n], S, T) \text{ iff } \mathcal{X}(S) = T$$

The ROOTS constraint holds iff S is the set of indices which map to an element in T .

$$\text{ROOTS}([X_1, \dots, X_n], S, T) \text{ iff } S = \mathcal{X}^{-1}(T)$$

ROOTS and RANGE are not exact inverses. A RANGE constraint can hold, but the corresponding ROOTS constraint may not, and vice versa. For instance, $\text{RANGE}([1, 1], \{1\}, \{1\})$

holds but not $\text{ROOTS}([1, 1], \{1\}, \{1\})$ since $\mathcal{X}^{-1}(1) = \{1, 2\}$, and $\text{ROOTS}([1, 1, 1], \{1, 2, 3\}, \{1, 2\})$ holds but not $\text{RANGE}([1, 1, 1], \{1, 2, 3\}, \{1, 2\})$ as no X_i is assigned to 2.

In an associated report, we discuss how to propagate these two global constraints. Enforcing HC on the RANGE constraint is polynomial and give an $O(nd + n \cdot |lb(T)|^{3/2})$ algorithm where d is the maximum domain size of the X_i . Enforcing BC on the ROOTS constraint is $O(nd)$. Unfortunately, enforcing HC on the ROOTS constraint is NP-hard. Nevertheless, all the cases needed here are polynomial. For example, if all X_i are ground, or T is ground then enforcing HC on $\text{ROOTS}([X_1, \dots, X_n], S, T)$ is $O(nd)$.

4 Range Constraints

In an associated report, we present a catalog containing over 70 global constraints from [Beldiceanu, 2000] specified with this simple language. We have room here to present just a few of the more important constraints. In this and the subsequent three sections, we list some global constraints which can be specified using RANGE constraints, using ROOTS constraints, and using both RANGE and ROOTS constraints.

4.1 All Different

The ALLDIFFERENT constraint forces a sequence of variables to take different values from each other. Such a constraint is useful in a wide range of problems (e.g. allocation of activities to different slots in a timetabling problem). It can be propagated efficiently [Régim, 1994]. It can also be decomposed with a single RANGE constraint:

$$\begin{aligned} &\text{ALLDIFFERENT}([X_1, \dots, X_n]) \text{ iff} \\ &\text{RANGE}([X_1, \dots, X_n], \{1, \dots, n\}, T) \wedge |T| = n \end{aligned}$$

A special but nevertheless important case of this constraint is the PERMUTATION constraint. This is an ALLDIFFERENT constraint where we additionally know $\mathcal{R}$, the set of values to be taken. That is, the sequence of variables is a permutation of the values in $\mathcal{R}$ where $|\mathcal{R}| = n$. This also can be decomposed using a single RANGE constraint:

$$\begin{aligned} &\text{PERMUTATION}([X_1, \dots, X_n], \mathcal{R}) \text{ iff} \\ &\text{RANGE}([X_1, \dots, X_n], \{1, \dots, n\}, \mathcal{R}) \end{aligned}$$

Such a decomposition of the PERMUTATION constraint obviously does not hinder propagation. However, decomposition of ALLDIFFERENT into a RANGE constraint does. In addition, HC on the RANGE decomposition is incomparable to AC on the decomposition of ALLDIFFERENT which uses a clique of binary inequality constraints. Thus, we may be able to obtain more pruning by using both decompositions.

Theorem 1 *GAC on PERMUTATION is equivalent to HC on the decomposition with RANGE. GAC on ALLDIFFERENT is stronger than HC on the decomposition with RANGE. AC on the decomposition of ALLDIFFERENT into binary inequalities is incomparable to HC on the decomposition with RANGE.*

Proof: Consider $X_1, X_2 \in \{1, 2\}$, $X_3 \in \{1, 2, 3, 4\}$, and $\{1, 2\} \subseteq T \subseteq \{1, 2, 3, 4\}$. Then

$\text{RANGE}([X_1, X_2, X_3], \{1, 2, 3\}, T)$ and $|T| = 3$ are both HC, but $\text{ALLDIFFERENT}([X_1, X_2, X_3])$ is not GAC. Consider $X_1, X_2 \in \{1, 2\}$, $X_3 \in \{1, 2, 3\}$, and $T = \{1, 2, 3\}$. Then $X_1 \neq X_2$, $X_1 \neq X_3$ and $X_2 \neq X_3$ are AC but $\text{RANGE}([X_1, X_2, X_3], \{1, 2, 3\}, T)$ is not HC. Consider $X_1, X_2 \in \{1, 2, 3, 4\}$, $X_3 \in \{2\}$, and $\{2\} \subseteq T \subseteq \{1, 2, 3, 4\}$. Then $\text{RANGE}([X_1, X_2, X_3], \{1, 2, 3\}, T)$ and $|T| = 3$ are HC. But $X_1 \neq X_3$ and $X_2 \neq X_3$ are not AC. The other result holds immediately. $\square$

4.2 Number of Values

The NVALUE constraint is useful in a wide range of problems involving resources since it counts the number of distinct values used by a sequence of variables [Pachet and Roy, 1999]. $\text{NVALUE}([X_1, \dots, X_n], N)$ holds iff $N = |\{X_i \mid 1 \leq i \leq n\}|$. The ALLDIFFERENT constraint is a special case of the NVALUE constraint in which $N = n$. Unfortunately, it is NP-hard in general to enforce GAC on a NVALUE constraint [Bessiere *et al.*, 2004]. However, there is an $O(n \log(n))$ algorithm to enforce a level of consistency similar to BC [Beldiceanu, 2001]. An alternative and even simpler way to implement this constraint is with a RANGE constraint:

$$\begin{aligned} &\text{NVALUE}([X_1, \dots, X_n], N) \text{ iff} \\ &\text{RANGE}([X_1, \dots, X_n], \{1, \dots, n\}, T) \wedge |T| = N \end{aligned}$$

HC on this decomposition is incomparable to BC on the NVALUE constraint.

Theorem 2 *BC on NVALUE is incomparable to HC on the decomposition.*

Proof: Consider $X_1, X_2 \in \{1, 2\}$, $X_3 \in \{1, 2, 3, 4\}$, $N \in \{3\}$ and $\{1, 2\} \subseteq T \subseteq \{1, 2, 3, 4\}$. Then $\text{RANGE}([X_1, X_2, X_3], \{1, 2, 3\}, T)$ and $|T| = N$ are both HC. However, enforcing BC on $\text{NVALUE}([X_1, X_2, X_3], N)$ prunes 1 and 2 from X_3 .

Consider $X_1, X_2, X_3 \in \{1, 3\}$ and $N \in \{3\}$. Then $\text{NVALUE}([X_1, X_2, X_3], N)$ is BC. However, enforcing HC on $\text{RANGE}([X_1, X_2, X_3], \{1, 2, 3\}, T)$ makes $\{1, 3\} \subseteq T \subseteq \{1, 3\}$ which will cause $|T| = 3$ to fail. $\square$

4.3 Uses

In [Beldiceanu *et al.*, 2004b], propagation algorithms achieving GAC and BC are proposed for the USED BY constraint. $\text{USED BY}([X_1, \dots, X_n], [Y_1, \dots, Y_m])$ holds iff the *multiset* of values assigned to $Y_1, \dots, Y_m$ is a subset of the *multiset* of values assigned to $X_1, \dots, X_n$. We now introduce a variant of the USED BY constraint called the USES constraint. $\text{USES}([X_1, \dots, X_n], [Y_1, \dots, Y_m])$ holds iff the *set* of values assigned to $Y_1, \dots, Y_m$ is a subset of the *set* of values assigned to $X_1, \dots, X_n$. That is, USED BY takes into account the number of times a value is used while USES does not. Unlike the USED BY constraint, enforcing GAC on USES is NP-hard.

Theorem 3 *Enforcing GAC on USES is NP-hard.*

Proof: We reduce 3-SAT to the problem of deciding if a USES constraint has a solution. Finding support is therefore NP-hard. Consider a formula φ with n Boolean variables and m clauses. For each Boolean variable i , we introduce a variable $X_i \in \{i, -i\}$. For each clause $c_j = x \vee \neg y \vee z$, we

introduce $Y_j \in \{x, -y, z\}$. Then φ has a model iff the USES constraint has a satisfying assignment, and i is true iff $X_i = i$. $\square$

One way to propagate a USES constraint is to decompose it using RANGE constraints:

$$\begin{aligned} &\text{USES}([X_1, \dots, X_n], [Y_1, \dots, Y_m]) \text{ iff} \\ &\text{RANGE}([X_1, \dots, X_n], \{1, \dots, n\}, T) \wedge \\ &\text{RANGE}([Y_1, \dots, Y_m], \{1, \dots, m\}, T') \wedge T' \subseteq T \end{aligned}$$

Enforcing HC on this decomposition is polynomial. Not surprisingly, this hinders propagation (otherwise we would have a polynomial algorithm for a NP-hard problem).

Theorem 4 *GAC on USES is stronger than HC on the decomposition.*

Proof: Consider $X_1 \in \{1, 2, 3, 4\}$, $X_2 \in \{1, 2, 3, 5\}$, $X_3, X_4 \in \{4, 5, 6\}$, $Y_1 \in \{1, 2\}$, $Y_2 \in \{1, 3\}$, and $Y_3 \in \{2, 3\}$. The decomposition is HC while GAC on USES prunes 4 from the domain of X_1 and 5 from the domain of X_2 . $\square$

Thus, decomposition is a simple method to obtain a polynomial propagation algorithm.

5 Roots Constraints

RANGE constraints are often useful to specify constraints on the values used by a sequence of variables. ROOTS constraint, on the other hand, are useful to specify constraints on the variables taking particular values.

5.1 Global Cardinality

The global cardinality constraint introduced in [Régim, 1996] constrains the number of times values are used. We consider a generalization in which the number of occurrences of a value may itself be an integer variable. That is, $\text{GCC}([X_1, \dots, X_n], [d_1, \dots, d_m], [O_1, \dots, O_m])$ holds iff $|\{i \mid X_i = d_j\}| = O_j$ for all j . Such a GCC constraint can be decomposed into a set of ROOTS constraints:

$$\begin{aligned} &\text{GCC}([X_1, \dots, X_n], [d_1, \dots, d_m], [O_1, \dots, O_m]) \text{ iff} \\ &\forall i. \text{ROOTS}([X_1, \dots, X_n], S_i, \{d_i\}) \wedge |S_i| = O_i \end{aligned}$$

Enforcing HC on these ROOTS constraints is polynomial since the sets $\{d_i\}$ are ground. Enforcing GAC on a generalized GCC constraint is NP-hard, but we can enforce GAC on the X_i and BC on the O_j in polynomial time using a specialized algorithm [Quimper *et al.*, 2004]. This is more than is achieved by the decomposition.

Theorem 5 *GAC on the X_i and BC on the O_j of a GCC constraint is stronger than HC on the decomposition using ROOTS constraints.*

Proof: Sets being represented by their bounds, HC on the decomposition cannot prune more on the O_j than BC does on the GCC. To show strictness, consider $X_1, X_2 \in \{1, 2\}$, $X_3 \in \{1, 2, 3\}$, $d_i = i$ and $O_1, O_2, O_3 \in \{0, 1\}$. The decomposition is HC (with $\{1\} \subseteq S_1, S_2 \subseteq \{1, 2, 3\}$ and $\{1\} \subseteq S_3 \subseteq \{3\}$). However, enforcing GAC on the X_i and BC on the O_j of the GCC constraint will prune 1 and 2 from X_3 and 0 from O_1, O_2 and O_3 . $\square$

This example illustrates that, whilst many global constraints can be expressed in terms of ROOTS and RANGE, there are some global constraints like GCC for which it is worth developing specialized propagation algorithms. Nevertheless, ROOTS and RANGE provide a means of propagation for such constraints in the absence of specialised algorithms.

5.2 Among

The AMONG constraint was introduced in CHIP to help model resource allocation problems like car sequencing [Beldiceanu and Contejean, 1994]. It counts the number of variables using values from a given set. $\text{AMONG}([X_1, \dots, X_n], [d_1, \dots, d_m], N)$ holds iff $N = |\{i \mid X_i \in \{d_1, \dots, d_m\}\}|$.

One decomposition using the global cardinality constraint GCC introduced in [R  gin, 1996] is as follows:

$$\text{AMONG}([X_1, \dots, X_n], [d_1, \dots, d_m], N) \text{ iff} \\ \text{GCC}([X_1, \dots, X_n], [d_1, \dots, d_m], [O_1, \dots, O_m]) \wedge \sum O_i = N$$

Unfortunately, this decomposition hinders propagation.

Theorem 6 *GAC on AMONG is stronger than GAC on the decomposition using GCC.*

Proof: Consider $X_1, X_2 \in \{1, 2\}$, $X_3 \in \{1, 2, 3\}$, $d_1 = 1$, $d_2 = 2$, and $N \in \{1, 2\}$. The decomposition is GAC, but enforcing GAC on $\text{AMONG}([X_1, X_2, X_3], [d_1, d_2], N)$ prunes 1 and 2 from X_3 as well as 1 from N . $\square$

An alternative way to propagate the AMONG constraint is to decompose it using a ROOTS constraint:

$$\text{AMONG}([X_1, \dots, X_n], [d_1, \dots, d_m], N) \text{ iff} \\ \text{ROOTS}([X_1, \dots, X_n], S, \{d_1, \dots, d_m\}) \wedge |S| = N$$

It is polynomial to enforce HC on this case of the ROOTS constraint since the target set is ground. This decomposition also does not hinder propagation. It is therefore a potentially attractive method to implement the AMONG constraint.

Theorem 7 *GAC on AMONG is equivalent to HC on the decomposition using ROOTS.*

Proof: (Sketch) Suppose the decomposition into $\text{ROOTS}([X_1, \dots, X_n], S, \{d_1, \dots, d_m\})$ and $|S| = N$ is HC. The variables X_i divide into three categories: those whose domain only contains elements from $\{d_1, \dots, d_m\}$ (at most $\min(N)$ such vars); those whose domain do not contain any such elements (at most $n - \max(N)$ such vars); those whose domain contains both elements from this set and from outside. Consider any value for a variable X_i in the first such category. To construct support for this value, we assign the remaining variables in the first category with values from $\{d_1, \dots, d_m\}$. If the total number of assigned values is less than $\min(N)$, we assign a sufficient number of variables from the second category with values from $\{d_1, \dots, d_m\}$ to bring up the count to $\min(N)$. We then assign all the remaining unassigned X_j with values outside $\{d_1, \dots, d_m\}$. Finally, we assign $\min(N)$ to N . Support can be constructed for variables in the other two categories in a similar way, as well as for any value of N between $\min(N)$ and $\max(N)$. $\square$

The ATMOST and ATLEAST constraints are closely related. The ATMOST constraint puts an upper bound on

the number of variables using a particular value, whilst the ATLEAST puts a lower bound. Both can be decomposed using a ROOTS constraint without hindering propagation.

6 Range and Roots Constraints

Some global constraints need both ROOTS and RANGE constraints in their specifications.

6.1 Assign and NValues

In bin packing and knapsack problems, we may wish to assign both a value and a bin to each item, and place constraints on the values appearing in each bin. For instance, in the steel mill slab design problem (prob038 in CSPLib), we assign colors and slabs to orders so that there are a limited number of colors on each slab. $\text{ASSIGN\&NVALUES}([X_1, \dots, X_n], [Y_1, \dots, Y_n], N)$ holds iff $|\{Y_i \mid X_i = j\}| \leq N$ for each j [Beldiceanu, 2000]. It can be decomposed into a set of ROOTS and RANGE constraints:

$$\text{ASSIGN\&NVALUES}([X_1, \dots, X_n], [Y_1, \dots, Y_n], N) \text{ iff} \\ \forall j. \text{ROOTS}([X_1, \dots, X_n], S_j, \{j\}) \wedge \\ \text{RANGE}([Y_1, \dots, Y_n], S_j, T_j) \wedge |T_j| \leq N$$

Decomposition hinders propagation as it considers the values separately. However, since this constraint generalizes NVALUE, enforcing GAC is NP-hard. Decomposition is thus one method to obtain a polynomial propagation algorithm.

6.2 Common

A generalization of the AMONG and ALLDIFFERENT constraints introduced in [Beldiceanu, 2000] is the COMMON constraint. $\text{COMMON}(N, M, [X_1, \dots, X_n], [Y_1, \dots, Y_m])$ ensures $N = |\{i \mid \exists j, X_i = Y_j\}|$ and $M = |\{j \mid \exists i, X_i = Y_j\}|$. That is, N variables in X_i take values in common with Y_j and M variables in Y_j takes values in common with X_i . We prove that we cannot expect to enforce GAC on such a constraint as it is NP-hard to do so in general.

Theorem 8 *Enforcing GAC on COMMON is NP-hard.*

Proof: We again use a transformation from 3-SAT. Consider a formula φ with n Boolean variables and m clauses. For each Boolean variable i , we introduce a variable $X_i \in \{i, -i\}$. For each clause $c_j = x \vee \neg y \vee z$, we introduce $Y_j \in \{x, -y, z\}$. We let $N \in \{0, \dots, n\}$ and $M = m$. φ has a model iff the COMMON constraint has a solution in which the X_i take the literals true in this model. $\square$

One way to propagate a COMMON constraint is to decompose it into RANGE and ROOTS constraints:

$$\text{COMMON}(N, M, [X_1, \dots, X_n], [Y_1, \dots, Y_m]) \text{ iff} \\ \text{RANGE}([Y_1, \dots, Y_m], \{1, \dots, m\}, T) \wedge \\ \text{ROOTS}([X_1, \dots, X_n], S, T) \wedge |S| = N \wedge \\ \text{RANGE}([X_1, \dots, X_n], \{1, \dots, n\}, V) \wedge \\ \text{ROOTS}([Y_1, \dots, Y_m], U, V) \wedge |U| = M$$

Enforcing HC on this decomposition is polynomial. Decomposition thus offers a simple and promising method to propagate a COMMON constraint. Not surprisingly, the decomposition hinders propagation.

Theorem 9 *GAC on COMMON is stronger than HC on the decomposition.*

Proof: Consider $N = M = 0$, $X_1, Y_1 \in \{1, 2\}$, $X_2, Y_2 \in \{1, 3\}$, $Y_3 \in \{2, 3\}$. Hybrid consistency on the decomposition enforces $\{\} \subseteq T$, $V \subseteq \{1, 2, 3\}$, and $S = U = \{\}$ but no pruning on the X_i and Y_j . However, enforcing GAC on $\text{COMMON}(N, M, [X_1, X_2], [Y_1, Y_2, Y_3])$ prunes 2 from X_1 , 3 from X_2 and 1 from both Y_1 and Y_2 . $\square$

6.3 Symmetric All Different

In certain domains, we may need to find symmetric solutions. For example, in sports scheduling problems, if one team is assigned to play another then the second team should also be assigned to play the first. $\text{SYMMALLDIFF}([X_1, \dots, X_n])$ ensures $X_i = j$ iff $X_j = i$ [Régin, 1999]. It can be decomposed into a set of ROOTS and RANGE constraints:

$$\begin{aligned} & \text{SYMMALLDIFF}([X_1, \dots, X_n]) \text{ iff} \\ & \text{RANGE}([X_1, \dots, X_n], \{1, \dots, n\}, \{1, \dots, n\}) \wedge \\ & \forall i. \text{ROOTS}([X_1, \dots, X_n], S_i, \{i\}) \wedge X_i \in S_i \wedge |S_i| = 1 \end{aligned}$$

It is polynomial to enforce HC on these cases of the ROOTS constraint. However, as with the ALLDIFFERENT constraint, it is more effective to use a specialized propagation algorithm like that in [Régin, 1999].

Theorem 10 *GAC on SYMMALLDIFF is stronger than HC on the decomposition.*

Proof: Consider $X_1 \in \{2, 3\}$, $X_2 \in \{1, 3\}$, $X_3 \in \{1, 2\}$, $\{\} \subseteq S_1 \subseteq \{2, 3\}$, $\{\} \subseteq S_2 \subseteq \{1, 3\}$, and $\{\} \subseteq S_3 \subseteq \{1, 2\}$. Then the decomposition is HC. However, enforcing GAC on $\text{SYMMALLDIFF}([X_1, X_2, X_3])$ will detect unsatisfiability. $\square$

To our knowledge, this constraint has not been integrated into any constraint solver. Thus, this decomposition provides a means of propagation for the SYMMALLDIFF constraint.

7 Beyond Counting and Occurrence

The RANGE and ROOTS constraints are useful for specifying a wide range of counting and occurrence constraints. Nevertheless, their expressive power permits their use to specify many other constraints.

The ELEMENT constraint introduced in [Van Hentenryck and Carillon, 1988] indexes into an array with a variable. More precisely, $\text{ELEMENT}(I, [X_1, \dots, X_n], J)$ holds iff $X_I = J$. We can use such a constraint to look up the price of a component included in a configuration problem. The ELEMENT constraint can be decomposed into a RANGE constraint without hindering propagation:

$$\begin{aligned} & \text{ELEMENT}(I, [X_1, \dots, X_n], J) \text{ iff } |S| = |T| = 1 \wedge \\ & I \in S \wedge J \in T \wedge \text{RANGE}([X_1, \dots, X_n], S, T) \end{aligned}$$

We may wish to channel between a variable and the sequence of 0/1 variables representing the possible values taken by the variable. The $\text{DOMAIN}(X, [X_1, \dots, X_m])$ constraint introduced in [Refalo, 2000] ensures $X = i$ iff $X_i = 1$. This can be decomposed into a ROOTS constraint:

$$\begin{aligned} & \text{DOMAIN}(X, [X_1, \dots, X_m]) \text{ iff} \\ & \text{ROOTS}([X_1, \dots, X_m], S, \{1\}) \wedge |S| = 1 \wedge X \in S \end{aligned}$$

Enforcing HC on this decomposition is polynomial ($\{1\}$ is ground) and it is equivalent to enforcing GAC on the DOMAIN constraint.

The CONTIGUITY constraint ensures that, in a sequence of 0/1 variables, those taking the value 1 appear contiguously. This is a discrete form of convexity. The constraint was introduced in [Maher, 2002] to model a hardware configuration problem. It can be decomposed into a ROOTS constraint:

$$\begin{aligned} & \text{CONTIGUITY}([X_1, \dots, X_n]) \text{ iff} \\ & \text{ROOTS}([X_1, \dots, X_n], S, \{1\}) \wedge \\ & X = \max(S) \wedge Y = \min(S) \wedge |S| = X - Y + 1 \end{aligned}$$

Again it is polynomial to enforce HC on this case of the ROOTS constraint. Unfortunately, decomposition hinders propagation. Consider $X_1, X_3 \in \{0, 1\}$, $X_2, X_4 \in \{1\}$. HC on the decomposition will enforce $\{2, 4\} \subseteq S \subseteq \{1, 2, 3, 4\}$, $X \in \{4\}$, $Y \in \{1, 2\}$ and $|S|$ to be in $\{3, 4\}$ but no pruning will happen on the X_i . However, enforcing GAC on $\text{CONTIGUITY}([X_1, \dots, X_n])$ will prune 0 from X_3 .

Whilst ROOTS and RANGE can specify concepts quite distant from counting and occurrences like convexity and discrete derivatives, it seems that we may need other algorithmic ideas to propagate them effectively.

8 Experiments

The last three sections demonstrate that many global constraints can be specified in terms of ROOTS and RANGE constraints. We also argued that these specifications are executable and in certain cases provide efficient and effective propagation algorithms. In this section, we support this argument by means of some experiments.

Size	2GCC+SUM			GCC+ROOTS			ROOTS		
	f	t	#s	f	t	#s	f	t	#s
10	6	0.02	9/10	6	0.01	9/10	335	0.41	10/10
15	6475	1.56	29/52	6468	1.38	29/52	23	0.10	52/52
20	2852	1.37	20/35	2852	0.97	20/35	1093	3.54	32/35
25	71	0.22	13/20	71	0.09	13/20	2776	13.62	19/20
30	12472	11.87	6/10	12472	6.80	7/10	4279	17.23	9/10
35	16	0.44	20/56	14	0.15	21/56	5107	36.29	49/56

Table 1: Mystery Shopper. Instance are run with a 5 minutes time limit. #fails (f) and cpu time (t) are averaged on the #instances solved (#s) by each method.

We used a model for the Mystery Shopper problem [Cheng et al., 1998] due to Helmut Simonis that appears in CSPLib. This model contains a GCC constraint and a number of AMONG constraints. ILOG's Solver contains the GCC but not the AMONG constraint. We implement AMONG either with a GCC and a SUM constraint or directly with ROOTS. This gives us two models (2GCC+SUM and GCC+ROOTS, resp.). We also consider the model (ROOTS) which is a variation of GCC+ROOTS where we decompose the GCC constraint using ROOTS. In order to generate a range of instances, we used the following protocol. The duration in weeks as well as the number of varieties of shoppers, visits per salelady and number of areas are all set to 4 (as in the original problem). The number of saleladies takes its value in $\{10, 15, 20, 25, 30, 35\}$, and the

number of shoppers is the next multiple of 4 that is not consecutive. All the possible ways of partitioning the salesladies into 4 areas (that do not trivially violate the constraints) are solved. As expected, the decomposition of the AMONG constraint using ROOTS is more efficient than the decomposition with a GCC and a SUM constraint (see Table 1). Surprisingly, however, the ROOTS model is the most efficient strategy (see the '#s' column). This is because we were able to branch on the 'extra' set variables, which proved to be the most effective method. Finally, the inference using GCC (2GCC+SUM) is faster (better ratio t/f) than using its decomposition (ROOTS). This suggests that it will be worth investing time and effort into optimizing this new ROOTS constraint.

9 Related Work

Beldiceanu has specified a wide range of global constraints using simple properties of suitably constructed graphs [Beldiceanu, 2000]. There are several important differences between this approach and the framework presented here. For instance, our specification language is executable. Given a specification, we immediately have a polynomial propagation algorithm. This is not the case with Beldiceanu's framework. On the other hand, Beldiceanu's framework is more general since the ROOTS and RANGE constraints can easily be specified in terms of some simple graph properties.

A number of researchers have specified global constraints over sequences of variables using automaton. Beldiceanu *et al.* use an extended polynomial sized automaton, and show how to extract automatically a GAC propagation algorithm [Beldiceanu *et al.*, 2004a]. Pesant uses a deterministic finite automaton, and develops a GAC propagation algorithm for membership of the corresponding regular language [Pesant, 2004]. These approaches are complementary to the framework presented here. For example, automaton can easily specify constraints like the CONTIGUITY constraint and thus provide us with a GAC propagation algorithm. This does not seem possible with just ROOTS and RANGE constraints. On the other hand, deterministic or extended automaton cannot specify a simple PERMUTATION constraint.

10 Conclusions

We have proposed a simple declarative language for specifying a wide range of counting and occurrence constraints. The language uses just two global primitives: the RANGE constraint which computes the range of values used by a set of variables, and the ROOTS constraint which computes the variables in a set mapping onto particular values. This specification language is executable. It immediately provides a polynomial propagation algorithm for any constraint that can be specified. In some cases, this propagation algorithm achieves GAC (e.g. the PERMUTATION and AMONG constraints). In other cases, this propagation algorithm may not make the constraint GAC, but achieving GAC is NP-hard (e.g. the NVALUE and COMMON constraints). Decomposition is then one method to obtain a polynomial algorithm. In the remaining cases, the propagation algorithm may not make the constraint GAC, and specialized propagation algorithms can do so in polynomial time (e.g. the SYMALLDIFF constraint).

Our method can still be attractive in this last case as it provides a generic means of propagation for counting and occurrence constraints in the absence of a specialized algorithm. Experiments demonstrate that such propagation algorithms can be both efficient and effective in practice.

References

- [Beldiceanu and Contejean, 1994] N. Beldiceanu and E. Contejean. Introducing global constraints in CHIP. *Mathl. Comput. Modelling*, 20:97–123, no. 12 1994.
- [Beldiceanu *et al.*, 2004a] N. Beldiceanu, M. Carlsson, and T. Petit. Deriving filtering algorithms from constraint checkers. In *Proc. of CP'2004*.
- [Beldiceanu *et al.*, 2004b] N. Beldiceanu, I. Katriel, and S. Thiel. Filtering algorithms for the *same* and *usedby* constraints. MPI Technical Report MPI-I-2004-1-001, 2004.
- [Beldiceanu, 2000] N. Beldiceanu. Global constraints as graph properties on a structured network of elementary constraints of the same type. SICS TR T2000/01.
- [Beldiceanu, 2001] N. Beldiceanu. Pruning for the minimum constraint family and for the number of distinct values constraint family. In *Proc. of CP'2001*.
- [Bessiere *et al.*, 2004] C. Bessiere, E. Hebrard, B. Hnich, and T. Walsh. The complexity of global constraints. In *Proc. of AAAI'2004*.
- [Cheng *et al.*, 1998] B.M.W. Cheng, K.M.F. Choi, J.H.M. Lee, and J.C.K. Wu. Increasing Constraint Propagation by Redundant Modeling: an Experience Report. *Constraints*, 1998.
- [Debruyne and Bessière, 1997] R. Debruyne and C. Bessière. Some practicable filtering techniques for the constraint satisfaction problem. In *Proc. of IJCAI'97*.
- [Maher, 2002] M. Maher. Analysis of a global contiguity constraint. In *Proc. of the CP'2002 Workshop on Rule Based Constraint Reasoning and Programming*, 2002.
- [Pachet and Roy, 1999] F. Pachet and P. Roy. Automatic generation of music programs. In *Proc. of CP'99*.
- [Pesant, 2004] G. Pesant. A regular language membership constraint for finite sequences of variables. In *Proc. of CP'2004*.
- [Quimper *et al.*, 2004] C.-G. Quimper, A. Lopez-Ortiz, P. van Beek, and A. Golynski. Improved algorithms for the global cardinality constraint. In *Proc. of CP'2004*.
- [Refalo, 2000] P. Refalo. Linear formulation of constraint programming models and hybrid solvers. In *Proc. of CP'2000*.
- [Régim, 1994] J.-C. Régim. A filtering algorithm for constraints of difference in CSPs. In *Proc. of AAAI'94*.
- [Régim, 1996] J.-C. Régim. Generalized arc consistency for global cardinality constraints. In *Proc. of AAAI'96*.
- [Régim, 1999] J.-C. Régim. The symmetric alldiff constraint. In *Proc. of IJCAI'99*. Morgan Kaufmann, 1999.
- [Van Hentenryck and Carillon, 1988] P. Van Hentenryck and J.-P. Carillon. Generality versus specificity: An experience with AI and OR techniques. In *Proc. of AAAI'88*.

A Model for Generating Random Quantified Boolean Formulas

Hubie Chen

Departament de Tecnologia
Universitat Pompeu Fabra
Barcelona, Spain
hubie.chen@upf.edu

Yannet Interian

Center for Applied Mathematics
Cornell University
Ithaca, New York, USA
interian@cam.cornell.edu

Abstract

The *quantified boolean formula (QBF)* problem is a powerful generalization of the boolean satisfiability (SAT) problem where variables can be both universally and existentially quantified. Inspired by the fruitfulness of the established model for generating random SAT instances, we define and study a general model for generating random QBF instances. We exhibit experimental results showing that our model bears certain desirable similarities to the random SAT model, as well as a number of theoretical results concerning our model.

1 Introduction

A phenomenal success story of artificial intelligence research over the past decade has been progress on the boolean satisfiability (SAT) problem. The technology of solvers for SAT has been transformed by techniques such as conflict-driven learning, highly efficient propagation mechanisms, and randomization. Today, state-of-the-art solvers can cope with problem instances having sizes that are orders of magnitude beyond what one would have optimistically predicted a decade ago.

Inspired and encouraged by the tremendous advances in boolean satisfiability, many researchers have recently turned to studying a powerful generalization of boolean satisfiability where both universal and existential quantification of variables is permitted; this generalization is often called the *quantified boolean formula (QBF)* problem. Whereas the SAT problem provides a framework wherein one can model search problems within the complexity class NP, the QBF problem permits the modelling of problems having higher complexity—from the complexity class PSPACE—including problems from the areas of verification, planning, knowledge representation, game playing, logic, and combinatorics. Recall that SAT is known to be NP-complete, and QBF is known to be PSPACE-complete.

A significant tool for SAT research has been the model for generating random SAT instances studied by [Mitchell *et al.*, 1992], in which it is possible to reliably generate instances that are robustly difficult. This model has become a canonical benchmark for solver evaluation, has been used heavily in conducting and thinking about SAT-related research, and is widely regarded as a suitable basis for investigating

“average” or “typical” case complexity. In addition, theoretical issues concerning this model have stimulated fruitful and highly non-trivial interaction among the areas of artificial intelligence, theoretical computer science, probability theory, and statistical physics. One particular question that has garnered significant attention is the “satisfiability threshold conjecture” which roughly states that, as the density of constraints is increased, formulas abruptly change from being satisfiable to being unsatisfiable at a critical threshold point.

The utility and widespread embrace of the random SAT model suggests that research on QBF could similarly benefit from random QBF models. This paper defines and studies a general model for generating random instances of the QBF problem. We exhibit experimental results showing that our model bears certain desirable similarities to the random SAT model; in particular, we show that our model exhibits threshold and easy-hard-easy phenomena as in the SAT model. However, we also experimentally demonstrate that the much larger parameter space of our model offers a richer framework for exploring and controlling complexity than the SAT model. In addition, we present a number of theoretical results concerning our model, namely, we give techniques for establishing lower and upper bounds on the purported threshold in our model. This places our work in contrast to previous work on random QBF models (discussed below), which contained very little rigorous theoretical analysis. Our theoretical results demonstrate nice connections to both previously studied problems as well as less studied but natural problems; we believe that these results evidence that our model is mathematically tangible and holds promise for being analyzed on the level of sophistication that work on the SAT model has reached.

Formulation of QBF. In this paper, we use the formulation of QBF where consecutive variables having the same quantifier are combined into a set. The following are the thus the general forms of QBF formulas having one, two, and three *quantifier blocks*, respectively:

$$\begin{aligned} &\exists X_1 \phi \\ &\forall Y_1 \exists X_1 \phi \\ &\exists X_2 \forall Y_1 \exists X_1 \phi \end{aligned}$$

By a *quantifier block*, we mean an expression QV where Q is a quantifier ($\exists$ or $\forall$) and V is a set of boolean variables. In

general, an instance of QBF contains a sequence of quantifier blocks which alternate between universal and existential type, followed by a set of clauses, denoted here by ϕ .¹ The semantics of these formulas are defined as is standard in logic; for instance, a formula $\forall Y_1 \exists X_1 \phi$ is true if for every assignment to the variables Y_1 , there exists an assignment to the variables X_1 such that ϕ is true. An intuitive way to view a QBF formula is as a game between two players: a “universal player”, who attempts to falsify ϕ , and an “existential player”, who attempts to satisfy ϕ . Players set their variables in the order given by the quantifier prefix. For instance, a formula $\exists X_2 \forall Y_1 \exists X_1 \phi$ can be viewed as a three-round game in which the existential player first sets the variables X_1 , the universal player then sets the variables Y_1 , and the existential player finishes by setting the variables X_1 . In general, the formula is true if the existential player can always win the game by satisfying ϕ .

Description of model. Let us review the random SAT model, which our model generalizes. This model takes three parameters: the clause length k , the number of variables n , and the number of clauses C . A random formula with these parameters, denoted here by $k\text{-}\mathcal{F}(n, C)$, is generated one clause at a time; each clause is generated by selecting, uniformly at random, k distinct variables, and then negating each with probability one half.

Now we describe our random QBF model. In the random SAT model, every clause has the same length. In our model, this also holds, but in addition, the number of literals in a clause from each quantifier block is a constant. The first parameter to our model is thus a tuple specifying, for each quantifier block, the number of literals from that block that are in a clause. For instance, the tuple $(2, 3)$ specifies that the generated formulas are of the form $\forall Y_1 \exists X_1 \phi$, where each clause in ϕ has 5 literals, 2 from Y_1 and 3 from X_1 . We will call such formulas $(2, 3)$ -QBF formulas. As another example, the tuple $(4, 2, 3)$ specifies that the generated formulas are of the form $\exists X_2 \forall Y_1 \exists X_1 \phi$, where each clause in ϕ has 9 literals, 4 from X_2 , 2 from Y_1 , and 3 from X_1 . Analogously, we call such formulas $(4, 2, 3)$ -QBF formulas.

The second parameter to our model is a tuple of the same length as the first one, specifying the number of variables in each quantifier block. The third parameter to our model is simply the number of clauses C . As an example, suppose that the first tuple is $(2, 3)$, the second tuple is $(44, 37)$, and the number of clauses is 290. The generated formulas are of the form $\forall Y_1 \exists X_1 \phi$, where Y_1 has 44 variables, X_1 has 37 variables, each clause in ϕ has 2 literals from Y_1 and 3 literals from X_1 , and there is a total of 290 clauses in ϕ . Each clause is generated by selecting uniformly at random, for each quantifier block, the designated number of distinct variables, and then negating each variable with probability one half. We denote such a random formula by $(2, 3)\text{-}\mathcal{F}((44, 37), 290)$. In general, to denote random formulas from our model, we use

¹Note that we use the standard assumption that the innermost quantifier block is existential, for if it is universal, the block can be efficiently eliminated. We also assume, as usual, that different quantifier blocks have disjoint variable sets.

the notation

$$(j_l, k_l, \dots, j_1, k_1)\text{-}\mathcal{F}((m_l, n_l, \dots, m_1, n_1), C)$$

or

$$(k_l, j_{l-1}, \dots, j_1, k_1)\text{-}\mathcal{F}((n_l, m_{l-1}, \dots, m_1, n_1), C)$$

depending on whether or not the number of quantifier blocks is even or odd, respectively.

Notice that in a random QBF from this model, if any quantifier block is picked and the clauses are restricted to literals over the quantifier block, the result is a random formula over the variables of the quantifier block in the sense of the random SAT model. It can in fact be verified that the model has a certain “recursive” property: generating a random QBF and then randomly instantiating the variables of the outermost quantifier block results in another random QBF from the model.

Previous work. Random QBF models have been previously studied by [Cadoli *et al.*, 1999] and [Gent and Walsh, 1999]. [Cadoli *et al.*, 1999] have experimentally studied the QBF generalization of the SAT model where clauses of fixed length k are generated by randomly choosing k distinct variables from the entire set of variables, and then negating each with probability one half. In this generalization, denoted *FCL*, the generation of clauses does not differentiate among variables from different quantifier blocks. It was pointed out in [Gent and Walsh, 1999] that this model is “flawed” in that it generates trivially false formulas when the number of clauses is on the order of the square root of n , the number of variables. These trivially false formulas arise from pairs of clauses where one clause contains a single existential literal, the other contains a single existential literal that is the complement of the first, and no universal variable is repeated in the pair.

In attempts to remedy this issue, [Gent and Walsh, 1999] proposed two models, *model A* and *model B*, and [Cadoli *et al.*, 1999] proposed another model, *FCL2*. Model A and *FCL2* are essentially modifications of the original *FCL* model that disallow the types of clauses that lead to the trivial falsity; for instance, in Model A, each clause is generated independently as before, but clauses containing one existential literal are simply discarded and replaced. As pointed out in [Gent and Walsh, 1999], instances generated by Model A lack uniformity; in our view, the definitions of Model A and *FCL2* have an *ad hoc* flavor that we believe make them less amenable to mathematical analysis.

The study of model B in [Gent and Walsh, 1999] is concerned mostly with QBF formulas with two quantifier blocks; results of two experiments are reported. The parameterizations of Model B studied can be obtained within our model, and indeed our model can be viewed as a generalization of these parameterizations. However, no systematic definition of the parameter space of Model B is given. We believe that one of the contributions of this paper is the precise articulation of a general random QBF model that may be instantiated in various ways, and from which it is possible to methodically embark on experimental and theoretical explorations of the parameter space.

2 Experimental results

Our first set of experimental results concerns $(2, 3)$ -QBF formulas, all of which have the same number of variables, 80. We varied both the number of clauses, as well as the ratio of universal to existential variables, denoted by ρ (rho). Figure 1 shows the computational cost of deciding these formulas.

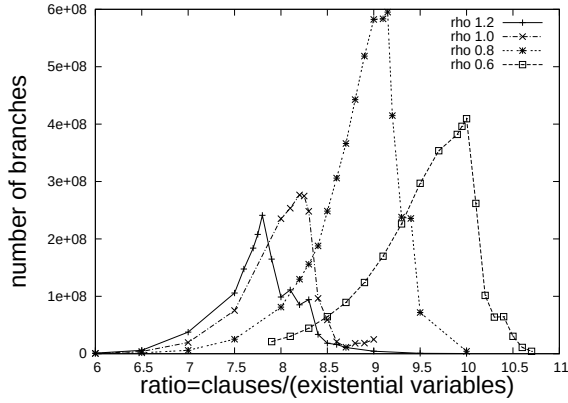


Figure 1: Computational cost of random $(2, 3)$ -QBF formulas with 80 variables.

Each curve corresponds to a different value of ρ . For instance, using the introduced notation, the curve with $\rho = 0.6$ corresponds to the hardness of formulas of the form $(2, 3)\text{-}\mathcal{F}((30, 50), C)$ for varying numbers of clauses C . The hardness is plotted as a function of the ratio of clauses to *existential* variables; in the case of $\rho = 0.6$ and 80 variables, this ratio is equal to $C/50$. Note that all of the experiments were performed using the solver QuBE-BJ [Giunchiglia *et al.*, 2001], and each point of our plots represents the median value.

Figure 2 concerns exactly the same formulas studied by Figure 1, but shows the probability of truth instead of the computational cost.

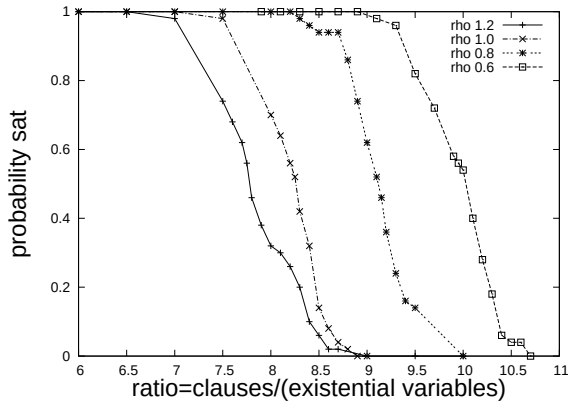


Figure 2: Probability of truth of random $(2, 3)$ -QBF formulas with 80 variables.

Examining each value of ρ individually, we see that as the number of clauses is increased, the formulas change from being almost certainly true to almost certainly false. Moreover,

this transition seems to take place relatively abruptly, suggesting that our model exhibits a threshold phenomenon as is conjectured in the random SAT model. Also, recall that in the random SAT model, computational difficulty exhibits an easy-hard-easy pattern, where the computational difficulty of formulas peaks around the point where half of the formulas are satisfiable. For each value of ρ in these two figures, and in all of the other experiments we performed, our model also exhibits this phenomena.

Now, let us consider the curves of Figure 1 together. The picture suggested is an extremely intriguing one: starting from the left, the peaks of each of the four curves appear to increase as ρ decreases, attain a maximum, and then decrease as ρ further decreases. That is, the peaks of the easy-hard-easy patterns appear to themselves exhibit an easy-hard-easy pattern! One lesson begged by this figure is that if one wishes to generate the computationally hardest quantified formulas for a given number of variables, it is of crucial importance to select the appropriate relative numbers of variables in each quantifier block. It is also clear from this figure that, even after we have committed ourselves to studying $(2, 3)$ -QBF formulas and fixed the number of variables, there are still two parameters, ρ and the clause density, that can be used to control complexity. We envision that the large parameter space of our model will be useful in the evaluation of QBF solvers. For instance, by tuning the clause density appropriately, one can generate an ensemble of $(2, 3)$ -QBF formulas with 80 variables which all have the same hardness (for one solver), but different ρ values; observing the behavior of another solver on such an ensemble could give insights into the relative behaviors of the solvers.

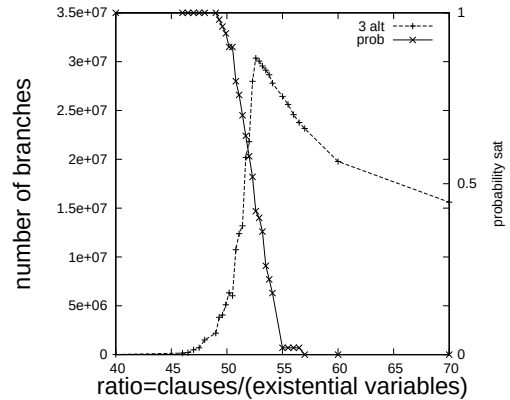


Figure 3: Formulas of type $(2, 2, 3)\text{-}\mathcal{F}((15, 15, 15), C)$. The x-axis is the ratio of clauses to 15, the number of existential variables in the innermost block X_1 .

Figure 3 shows results on $(2, 2, 3)$ -QBF formulas in which each quantifier block has 15 variables, for a total of 45 variables. When there is an odd number of quantifier blocks, a solver can conclude truth of a formula as soon as it finds an assignment to the outermost quantifier block such that the rest of the formula is true. Accordingly, for an odd number of quantifier blocks, the hardness drops down more slowly on the right of the peak (the “false” region) than on the left (the

“true” region). This slowness in dropping down on the right is quite pronounced in Figure 3. Note that a dual phenomenon takes place for an even number of quantifier blocks; see for example Figure 1.

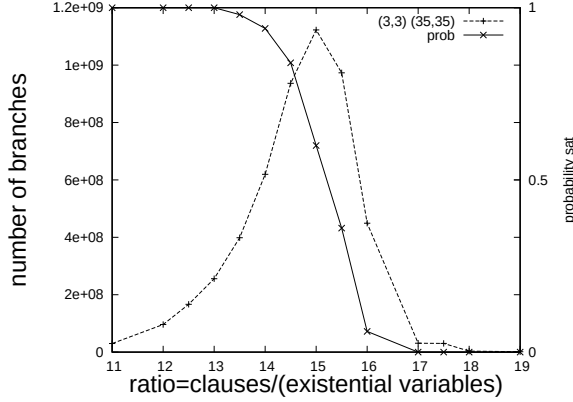


Figure 4: Formulas of type $(3, 3)\text{-}\mathcal{F}((35, 35), C)$.

Figure 4 shows results on $(3, 3)$ -QBF formulas with $\rho = 1$, that is, with the same number of existential and universal variables, 35 each, for a total of 70 variables. Interestingly, although the number of variables is less than in Figure 1, the hardest formulas (of Figure 4) are considerably harder than those of Figure 1, emphasizing the importance of the first parameter of our model, the tuple giving, for each quantifier block, the number of literals per clause.

Figure 5 shows results for $(1, 4)$ -QBF formulas with 80 variables. We again see that differences in ρ can dramatically affect complexity.²

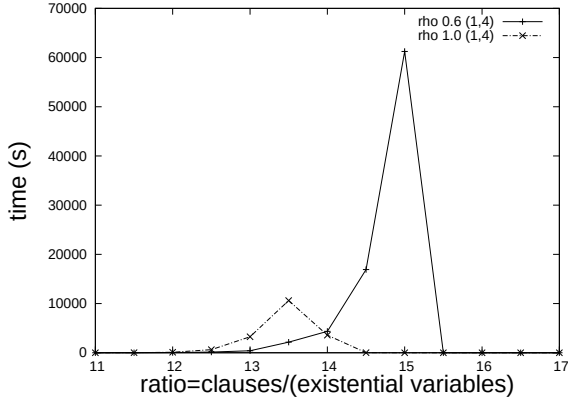


Figure 5: Formulas of type $(1, 4)\text{-}\mathcal{F}((30, 50), C)$ and $(1, 4)\text{-}\mathcal{F}((40, 40), C)$.

²Note that in Figure 5, we use execution time instead of number of branches as a measure of hardness. This is because in some cases, the number of branches was so high that it was not correctly reported by the solver (in particular, overflow occurred).

3 Theoretical results

In this section, we present a number of theoretical results which concern techniques for proving lower and upper bounds on the purported threshold in our model. In the random SAT model, theoretical work generally studies asymptotic properties of the model, in particular, properties that hold as the number of variables approaches infinity. In this section, we will similarly be concerned with asymptotic properties of our model, and we assume that in parameterizations

$$(j_l, k_l, \dots, j_1, k_1)\text{-}\mathcal{F}((m_l, n_l, \dots, m_1, n_1), C)$$

and

$$(k_l, j_{l-1}, \dots, j_1, k_1)\text{-}\mathcal{F}((n_l, m_{l-1}, \dots, m_1, n_1), C)$$

of our model, the k_i and j_i are constant, but the n_i , m_i , and C are all functions of an underlying argument n . We will discuss properties that hold *almost surely* by which we mean with probability tending to one as n approaches infinity.

Our first result concerns a way of obtaining threshold upper bounds, that is, “almost surely false” results. To illustrate the idea, let us focus on (j, k) -QBF formulas with two quantifier blocks, which are of the form $\forall Y_1 \exists X_1 \phi$. Consider the game view (mentioned in the introduction) of such a formula: the universal player wishes to set Y_1 so that the resulting formula cannot be satisfied by the existential player. A reasonable tactic for the universal player is to attempt to set the variables Y_1 so that the number of clauses in the resulting formula is maximized. Suppose that the universal player chooses a setting to Y_1 in such a way that she does not “look at” the existential literals of the formula—that is, the choice of assignment to Y_1 depends only on the universal literals in the formula. With this form of choice, in a sense that can be made precise, the resulting formula (over X_1) can be verified to be a random SAT formula. If the universal player can guarantee that the resulting random SAT formula has sufficiently many clauses, then almost sure falsity follows by making use of SAT threshold upper bounds.

Let us be a bit more precise. We show how to obtain almost sure falsity results for formulas of the form $(j, k)\text{-}\mathcal{F}((m, n), D)$. Assume that $k\text{-}\mathcal{F}(n, C)$ formulas are almost surely false, that is, C/n is an upper bound for the k -SAT threshold. If we restrict such a formula to the universal literals (the Y_1 literals), we obtain a $j\text{-}\mathcal{F}(m, D)$ formula. Assume further that $D \geq C$ is sufficiently high so that (almost surely) a $j\text{-}\mathcal{F}(m, D)$ has an assignment leaving C clauses unsatisfied. (In notation, we assume that $\min\text{SAT}(j\text{-}\mathcal{F}(m, D)) \leq D - C$ almost surely, where $\min\text{SAT}(F)$ denotes the minimum, over all assignments, of the number of clauses satisfied in the formula F .) Together, these two assumptions, in light of the above discussion, imply that $(j, k)\text{-}\mathcal{F}((m, n), D)$ formulas are almost surely false. The following proposition is a formal statement of this claim, but in a more general form.

Proposition 3.1. *Suppose that instances of*

$$(k_l, j_{l-1}, \dots, j_1, k_1)\text{-}\mathcal{F}((n_l, m_{l-1}, \dots, m_1, n_1), C)$$

are almost surely false, and that D is sufficiently high so that almost surely, it holds that

$$\min\text{SAT}(j_p, k_p, \dots, k_{l+1}, j_l)\text{-}\mathcal{F}((m_p, n_p, \dots, n_{l+1}, m_l), D)$$

is less than or equal to $D - C$. Then, instances of

$$(j_p, k_p, \dots, j_1, k_1) \text{-}\mathcal{F}((m_p, n_p, \dots, m_1, n_1), D)$$

are almost surely false.

Note that one can formulate a dual of Proposition 3.1 that allows one to infer “almost surely true” results from previous “almost surely true” results, and that concerns maxSAT instead of minSAT.

We next observe that, given a parameterization (of the model), if the parameterization is almost surely true after the elimination of a “prefix”, then the original parameterization is almost surely true. This permits, for instance, the derivation of “almost surely true” results for formulas beginning with a universal quantifier from such results for formulas beginning with an existential quantifier.

Proposition 3.2. *Let*

$$(j_l, k_l, \dots, j_1, k_1) \text{-}\mathcal{F}((m_l, n_l, \dots, m_1, n_1), C)$$

be a parameterization of the model. If for some $i < l$ it holds that instances of

$$(j_i, k_i, \dots, j_1, k_1) \text{-}\mathcal{F}((m_i, n_i, \dots, m_1, n_1), C)$$

or

$$(k_{i+1}, j_i, \dots, j_1, k_1) \text{-}\mathcal{F}((n_{i+1}, m_i, \dots, m_1, n_1), C)$$

are almost surely true, then instances of

$$(j_l, k_l, \dots, j_1, k_1) \text{-}\mathcal{F}((m_l, n_l, \dots, m_1, n_1), C)$$

are almost surely true.

Proposition 3.2 follows from the general fact that a quantified formula Φ is true if eliminating one or more quantifier blocks from the outside of Φ , along with their associated literals, gives a true formula.

Example 3.3. We demonstrate how to make use of the two propositions given so far by explicitly computing some threshold bounds for our model. We will consider $(2, 3)\text{-}\mathcal{F}((\rho n, n), cn)$ formulas, that is, formulas of the type considered in Figures 1 and 2.

We begin by looking at the case where $\rho = 1$, that is, $(2, 3)\text{-}\mathcal{F}((n, n), cn)$ formulas. We can use Proposition 3.1 specialized to two quantifier blocks (that is, with $p = l = 1$) to obtain an upper bound of $c = 14.5$ for such formulas, as discussed prior to the statement of that proposition. How is this done? The first requisite ingredient is a value c_u such that $3\text{-}\mathcal{F}(n, c_u n)$ formulas are almost surely false, that is, a SAT threshold upper bound. By [Dubois et al., 2000], we can take $c_u = 4.506$. The second ingredient needed is an upper bound for $\text{minSAT}(2\text{-}\mathcal{F}(n, cn))$. By Theorem 3.5, which is presented below, we have that $\text{minSAT}(2\text{-}\mathcal{F}(n, cn)) \leq (0.657)cn$ for $c = 14.5$. Using the proposition with $C = c_u n$ and $D = 14.5n$, we see that $(0.657)cn \leq (14.5n) - (c_u n)$ and hence we have $\text{minSAT}(2\text{-}\mathcal{F}(n, cn)) \leq D - C$ as required. Therefore, instances of $(2, 3)\text{-}\mathcal{F}((n, n), cn)$ are almost surely false for $c = 14.5$ (and hence for all $c \geq 14.5$).

Now let c_l be a lower bound on the 3-SAT threshold, so that instances of $3\text{-}\mathcal{F}(n, c_l n)$ are almost surely true. We can take $c_l = 3.52$ by [Kaporis et al., 2003]. By Proposition 3.2,

instances of $(2, 3)\text{-}\mathcal{F}((n, n), c_l n)$ are almost surely true. The idea is that $(2, 3)\text{-}\mathcal{F}((n, n), c_l n)$ formulas are almost surely true because after “eliminating” the first quantifier block in such formulas, one obtains $3\text{-}\mathcal{F}(n, c_l n)$ formulas, which are almost surely true by our choice of c_l . To summarize the results observed so far, we have that $(2, 3)\text{-}\mathcal{F}((n, n), cn)$ instances are almost surely true when $c \leq c_l = 3.52$, and almost surely false when $c \geq 14.5$.

Now, let us consider $(2, 3)\text{-}\mathcal{F}((\rho n, n), cn)$ formulas. Regarding lower bounds, as before, we have by Proposition 3.2 that instances of $(2, 3)\text{-}\mathcal{F}((\rho n, n), c_l n)$ are almost surely true. Turning to upper bounds, we observe that, by a standard probabilistic argument, every 2-SAT formula has an assignment satisfying (at most) $3/4$ of the clauses. Thus, we have $\text{minSAT}(2\text{-}\mathcal{F}(\rho n, 4c_u n)) \leq 3c_u n$ for any value of ρ . Using Proposition 3.1 as before with $C = c_u n$ and $D = 4c_u n$, we obtain that instances of $(2, 3)\text{-}\mathcal{F}((\rho n, n), cn)$ are almost surely false for $c \geq 4c_u$, that is, $4c_u$ is a threshold upper bound on formulas $(2, 3)\text{-}\mathcal{F}((\rho n, n), cn)$, for any value of ρ .

Summarizing, we have obtained that for any value of ρ , formulas $(2, 3)\text{-}\mathcal{F}((\rho n, n), cn)$ are almost surely false when $c \leq c_l$, and almost surely true when $c \geq 4c_u$. That is, such formulas change from being almost surely true to almost surely false between c_l and $4c_u$. If we set c_l and c_u to 4.26, which is an approximate value of the 3-SAT threshold, we expect the threshold for such formulas to lie between $c = 4.26$ and $c = 4 \cdot 4.26 = 17.04$, where c denotes the ratio of clauses to existential variables; this expectation is consistent with Figure 2.

In the usual random SAT model, whenever the clause length is fixed, it is known that at a sufficiently low constant clause-to-variable ratio, formulas are almost surely satisfiable, and that at a sufficiently high constant clause-to-variable ratio, formulas are almost surely unsatisfiable. These results, of course, validate the hypothesis that there is a threshold phenomenon in this model. We can show analogous results in our model.

Proposition 3.4. Fix $(j_l, k_l, \dots, j_1, k_1)$ to be a tuple with $k_l \geq 2$, and for $i \in \{1, \dots, l\}$, let $n_i(n)$ be a linear function of n , that is, $n_i(n) = \sigma_i n$ for a constant $\sigma_i > 0$. There exist positive constants a_l and a_u such that instances of

$$(j_l, k_l, \dots, j_1, k_1) \text{-}\mathcal{F}((m_l, n_l, \dots, m_1, n_1), cn)$$

are almost surely true when $c \leq a_l$, and instances of

$$(j_l, k_l, \dots, j_1, k_1) \text{-}\mathcal{F}((m_l, n_l, \dots, m_1, n_1), cn)$$

are almost surely false when $c \geq a_u$. (This proposition is meant to address formulas with both even and odd numbers of quantifier blocks, that is, we permit $j_l = 0$.)

The “almost surely true” part follows from Proposition 3.2 along with known lower bounds on the SAT threshold, while the “almost surely false” part follows from the fact that shifting existential quantifiers inward preserves truth of quantified formulas, the idea of Proposition 3.1, and a standard first moment argument.

We now study threshold upper bounds for our model in greater detail. Proposition 3.1 establishes a connection between minSAT and upper bounds in our model. Although

minSAT has not, as far as we know, been studied explicitly in previous work, we certainly believe that it is a natural concept of independent interest. We have the following result concerning it.

Theorem 3.5. *For every $k \geq 2$, there exists a function g_k such that for all $c > 0$, it holds almost surely that $\text{minSAT}(k\text{-}\mathcal{F}(n, cn)) \leq g_k(c)cn + o(n)$.*

We plot obtainable g_2 and g_3 in Figure 6, comparing them against the values $3/4$ and $7/8$ obtainable by a standard probabilistic argument.

We prove this theorem by analyzing a simple algorithm similar to one used in the paper [Coppersmith *et al.*, 2003] to study maxSAT. The algorithm is analyzed using the differential equation approach of [Wormald, 1995] that has been used in many random SAT threshold lower bound arguments.

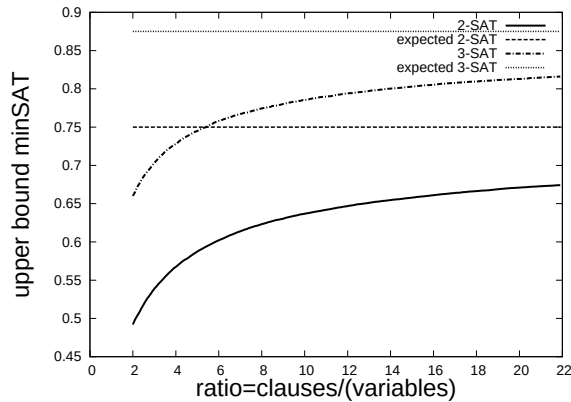


Figure 6: Upper bounds for $\text{minSAT}(k\text{-}\mathcal{F}(n, cn))$ from Theorem 3.5.

Now we look further at threshold lower bounds in our model, focusing on formulas with quantifier prefix $\forall\exists$. One can observe that if all “small” subformulas of a random SAT formula are satisfiable, at clause-to-variable ratio c , then formulas at clause-to-variable ratio c are true.

Proposition 3.6. *Fix a tuple (j, k) and constants $c, \rho > 0$. There exists a positive constant $x < 1$ (depending on j, c , and ρ) such that if all subformulas of size xcn in an instance of $k\text{-}\mathcal{F}(n, cn)$ are satisfiable almost surely, then instances of $(j, k)\text{-}\mathcal{F}((\rho n, n), cn)$ are true almost surely.*

Proposition 3.6 follows from the fact that a sufficiently small $y > 0$ can be picked so that the probability of

$$\text{minSAT}(j\text{-}\mathcal{F}(\rho n, cn)) \leq ycn$$

converges to zero. This can be proved by using a union bound and Chernoff bounds on a first moment. The desired x can then be obtained as $(1 - y)$.

This “small subformula” property has not been previously studied to our knowledge, but as with minSAT, we believe that it is a natural concept of possible independent interest. We can show that this property holds for random SAT formulas, for subformulas having size equal to a constant fraction of the number of clauses.

Theorem 3.7. *For all constants $k \geq 2$ and $c > 0$, there exists a constant $x > 0$ depending on k and c such that all size xcn subformulas of $k\text{-}\mathcal{F}(n, cn)$ are satisfiable almost surely.*

The proof of Theorem 3.7 makes use of [Chvatal and Szemerédi, 1988, Lemma 1].

Proposition 3.6 and Theorem 3.7 can be used together to obtain threshold lower bounds for $\forall\exists$ formulas. In particular, notice that in Proposition 3.6, relative to fixed c and ρ , increasing j allows one to decrease x . Moreover, the lower the x required in Theorem 3.7, the higher the c may be. Thus, fixing k and ρ , increasing j allows us to obtain higher and higher lower bounds on the threshold of $(j, k)\text{-}\mathcal{F}((\rho n, n), cn)$.

Acknowledgements. The authors wish to thank Víctor Dalmau and Bart Selman for helpful comments, and Massimo Narizzano for his help with QUBE.

References

- [Cadoli *et al.*, 1999] Marco Cadoli, Marco Schaerf, Andrea Giovanardi, and Massimo Giovanardi. An algorithm to evaluate quantified boolean formulae and its experimental evaluation. Technical report, Dipartimento di Informatica e Sistemistica, Universita di Roma “La Sapienza”, 1999.
- [Chvatal and Szemerédi, 1988] V. Chvatal and E. Szemerédi. Many hard examples for resolution. *Journal of the Association for Computing Machinery*, 35:759–768, 1988.
- [Coppersmith *et al.*, 2003] D. Coppersmith, D. Gamarnik, M. Hajiaghay, and G. Sorkin. Random max 2-sat and max cut, 2003.
- [Dubois *et al.*, 2000] Olivier Dubois, Yacine Boufkhad, and Jacques Mandler. Typical random 3-sat formulae and the satisfiability threshold. In *SODA*, pages 126–127, 2000. Full version in Electronic Colloquium on Computational Complexity (ECCC 2003).
- [Gent and Walsh, 1999] Ian P. Gent and Toby Walsh. Beyond NP: the QSAT phase transition. *Proceedings of AAAI 1999*, 1999.
- [Giunchiglia *et al.*, 2001] Enrico Giunchiglia, Massimo Narizzano, and Armando Tacchella. QUBE: A system for deciding quantified boolean formulas satisfiability. *Proceedings of the International Joint Conference on Automated Reasoning*, 2001.
- [Kaporis *et al.*, 2003] Alexis C. Kaporis, Lefteris M. Kirois, and Efthimios Lalas. Selecting complementary pairs of literals. *Electronic Notes in Discrete Mathematics*, 16, 2003.
- [Mitchell *et al.*, 1992] D. Mitchell, B. Selman, and H. Levesque. Hard and easy distributions of sat problems. In *Proc. 10-th National Conf. on Artificial Intelligence*, pages 459–465, 1992.
- [Wormald, 1995] N. C. Wormald. Differential equations for random processes and random graphs. *Annals of Applied Probability*, 5(4):1217–1235, 1995.

A Unified Theory of Structural Tractability for Constraint Satisfaction and Spread Cut Decomposition

David Cohen

Dept of Computer Science
Royal Holloway
University of London
Egham, Surrey, UK
D.Cohen@rhul.ac.uk

Peter Jeavons

Computing Laboratory
Oxford University
Wolfson Building, Parks Road
Oxford, UK
Peter.Jeavons@comlab.ox.ac.uk

Marc Gyssens

Department WNI
Limburgs Universitair Centrum
Universitaire Campus
B-3590 Diepenbeek, Belgium
marc.gyssens@luc.ac.be

Abstract

In this paper we introduce a generic form of structural decomposition for the constraint satisfaction problem, which we call a guarded decomposition. We show that many existing decomposition methods can be characterized in terms of finding guarded decompositions satisfying certain specified additional conditions.

Using the guarded decomposition framework we are also able to define a new form of decomposition, which we call a spread cut. We show that discovery of width k spread-cut decompositions is tractable for each k , and that the spread cut decomposition strongly generalize all existing decompositions except hypertrees. Finally we exhibit a family of hypergraphs H_n , for $n = 1, 2, 3 \dots$, where the width of the best hypertree decomposition of each H_n is at least $3n$, but the width of the best spread-cut decomposition is at most $2n$.

1 Introduction

A constraint satisfaction problem consists of a set of variables that must be assigned values subject to certain constraints. These constraints restrict the simultaneous assignments to certain specified subsets of the variables. Many real-world problems can be represented very naturally in this framework [Dechter, 2003; Marriott and Stuckey, 1998]

Similar problems arise in the field of relational databases, where they are referred to as conjunctive query evaluation problems [Kolaitis and Vardi, 2000]. Many of the techniques developed in this paper can also be applied directly to the conjunctive query evaluation problem, but we shall not pursue this application here.

The decision problem for the general constraint satisfaction problem is NP-hard [Mackworth, 1977]. This motivates the search for more restricted subproblems which are *tractable*, that is, solvable in polynomial-time.

This paper considers subproblems of the constraint satisfaction problem which can be shown to be tractable using so-called *structural* methods or *decomposition* methods. These methods rely exclusively on using the structural properties of instances, in other words the way in which the constraints overlap each other.

A variety of such decomposition methods have been developed and applied in both the constraint satisfaction community, and the database community. Examples include methods based on the use of treewidth [Freuder, 1985], cycle cut-sets [Dechter and Pearl, 1988], tree-clustering [Dechter and Pearl, 1989], hinges [Gyssens *et al.*, 1994], cycle hypercut-sets and hypertrees [Gottlob *et al.*, 2000]. All of these methods rely on reducing a given problem instance to an equivalent instance with a simpler structure, which can then be solved efficiently.

In this paper we present a generic, abstract form of decomposition, which we call a *guarded decomposition*. We show that all of the earlier decomposition methods can be viewed as special cases of guarded decomposition, each characterized by some simple additional conditions.

One existing decomposition method, based on the use of hypertrees [Gottlob *et al.*, 2000; 2002], is of particular importance as it has been shown to be strictly more general than many other decomposition methods and that, for each k , it is tractable to discover hypertree decompositions of width at most k . The general framework presented here allows us to identify a new decomposition method, based on the use of a structure that we call a *spread cut*. We show that spread-cut decompositions generalize all of the methods previously shown to be generalized by hypertree decompositions, and still it is tractable, for each k to discover spread cut decompositions of width k . Furthermore, we exhibit a family of examples for which the spread-cut decomposition method obtains better decompositions than the hypertree decomposition method.

2 Constraint Satisfaction, Hypergraphs, and Tractability

Definition 2.1 A CSP instance is a triple $P = \langle V, D, C \rangle$ where:

- V is a finite set of **variables**;
- D is a set called the **domain** of P ;
- C is a set of **constraints**. Each constraint $c \in C$ is a pair $c = \langle \chi, \rho \rangle$ where $\chi \subseteq V$ is a set of variables, called the **scope** of c , and $\rho \subseteq D^\chi$ is a set of functions from the scope to the domain, called the **relation** of c .

A **solution** to the CSP instance $P = \langle V, D, C \rangle$ is a function from V to D whose restriction¹ to the scope of any constraint $c \in C$ is one of the functions in the relation of c .

Example 2.2 Consider the CSP instance $P_{AG} = \langle V, D, C \rangle$ where $V = \{1, 2, \dots, 10\}$, $D = \{0, 1\}$, and $C = \{c_1, c_2, \dots, c_8\}$.

This instance has ten variables which must each be assigned the value 0 or 1, subject to 8 constraints. The constraints in C are defined as follows:

$$\begin{aligned} c_1 &= \langle \{1, 2\}, D^{\{1,2\}} \rangle, & c_2 &= \langle \{2, 3, 9\}, D^{\{2,3,9\}} \rangle, \\ c_3 &= \langle \{3, 4, 10\}, D^{\{3,4,10\}} \rangle, & c_4 &= \langle \{4, 5\}, D^{\{4,5\}} \rangle, \\ c_5 &= \langle \{5, 6, 9\}, D^{\{5,6,9\}} \rangle, & c_6 &= \langle \{6, 7, 10\}, D^{\{6,7,10\}} \rangle, \\ c_7 &= \langle \{7, 8, 9\}, D^{\{7,8,9\}} \rangle, & & \\ c_8 &= \langle \{1, 8, 10\}, \{f \mid f(1) = f(8) = 0 \rightarrow f(10) = 1\} \rangle. \end{aligned}$$

Note that each constraint except c_8 allows every assignment to the variables of its scope. The constraint c_8 does not allow the three variables of its scope all to take the value 0 simultaneously, but does allow all other assignments.

A straightforward calculation shows that this instance has exactly $7 * 2^5 = 224$ solutions. $\square$

In order to study the *structural* properties of CSP instances, that is, the way in which the constraint scopes overlap each other, we need the standard notion of a *hypergraph*.

Definition 2.3 A **hypergraph** is a pair $H = \langle V, E \rangle$, where V is an arbitrary set, called the **vertices** of H , and E is a set of subsets of V , called the **hyperedges** of H .

Definition 2.4 For any CSP instance $P = \langle V, D, C \rangle$, the **structure** of P , denoted $\sigma(P)$, is the hypergraph $\langle V, \{\chi \mid \langle \chi, \rho \rangle \in C\} \rangle$.

For any hypergraph H , the class of all CSP instances with structure H is denoted $\Psi(H)$.

Example 2.5 Recall the CSP instance P_{AG} defined in Example 2.2. The structure of P_{AG} is the hypergraph H_{AG} illustrated in Figure 1.

The set of vertices of H_{AG} is the set $\{1, 2, \dots, 10\}$, and the eight hyperedges of H_{AG} are the following subsets of these vertices: $e_1 = \{1, 2\}$, $e_2 = \{2, 3, 9\}$, $e_3 = \{3, 4, 10\}$, $e_4 = \{4, 5\}$, $e_5 = \{5, 6, 9\}$, $e_6 = \{6, 7, 10\}$, $e_7 = \{7, 8, 9\}$, $e_8 = \{1, 8, 10\}$. $\square$

As indicated in the Introduction, the decision problem for the general constraint satisfaction problem is NP-hard [Mackworth, 1977], so there has been considerable interest in finding more restricted problem classes whose instances can be recognized and solved in polynomial time.

Definition 2.6 A class I of CSP instances is called **tractable** if there is a polynomial time algorithm to decide membership of I , and a polynomial time algorithm which solves all instances in I .

A class I of CSP instances is called **structural** if $I = \bigcup_{H \in \mathcal{H}} \Psi(H)$ for some class of hypergraphs $\mathcal{H}$.

¹To simplify the presentation we assume, throughout this paper, that every variable of a CSP instance is constrained; that is, every variable occurs in the scope of some constraint.

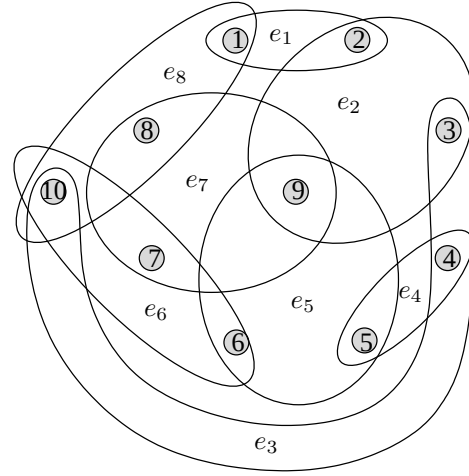


Figure 1: The hypergraph H_{AG} , which is the structure of the CSP instance P_{AG} , defined in Example 2.2.

3 Guarded Decompositions

Two CSP instances with the same set of variables are called *solution equivalent* if they have the same set of solutions. All known tractable structural classes of CSP instances are characterized by showing that the instances they contain can be transformed into solution equivalent instances with simpler structure. The constraints of these transformed instances are obtained by calculating the relational joins of certain constraint relations in the original instances, and then projecting these onto new scopes. To describe this general transformation scheme we introduce the following terminology.

Definition 3.1 A **guarded block** of a hypergraph H is a pair $\langle \lambda, \chi \rangle$ where the **guard**, λ , is a set of hyperedges of H , and the **block**, χ , is a subset of the vertices of the guard.

Definition 3.2 For any CSP instance P , and any guarded block $\langle \lambda, \chi \rangle$ of $\sigma(P)$, the constraint **generated** by P on $\langle \lambda, \chi \rangle$ is the constraint $\langle \chi, \rho \rangle$, where ρ is the projection onto χ of the relational join of all the constraints of P whose scopes are elements of λ .

Given a CSP instance P , with structure H , and a collection of guarded blocks of H , we can generate constraints on each of these guarded blocks to obtain a new collection of constraints, and hence a new CSP instance. In some cases, if the guarded blocks are carefully chosen, this new instance will be solution equivalent to P , and so can be used to solve P . If this property holds for any CSP instance with structure H , then the collection of guarded blocks will be called a *guarded decomposition* of H , which is formally defined as follows:

Definition 3.3 A set of guarded blocks Ξ of a hypergraph H is called a **guarded decomposition** of H if for every CSP instance $P = \langle V, D, C \rangle \in \Psi(H)$, the instance $P' = \langle V, D, C' \rangle$, where C' is the set of constraints generated by P on the members of Ξ , is solution equivalent to P .

How can we determine whether a given collection of guarded blocks is in fact a guarded decomposition? Theorem 3.5 be-

low gives a simple and efficient way to answer this question, based on the following properties.

Definition 3.4 A guarded block $\langle \lambda, \chi \rangle$ of a hypergraph H **covers** a hyperedge e of H if e is contained in χ .

A set of guarded blocks Ξ of a hypergraph H is called a **guarded cover** for H if each hyperedge of H is covered by some guarded block of Ξ .

A set of guarded blocks Ξ of a hypergraph H is called a **complete guarded cover** for H if each hyperedge e of H occurs in the guard of some guarded block of Ξ which covers e .

Theorem 3.5 A set of guarded blocks Ξ of a hypergraph H is a guarded decomposition of H if and only if it is a complete guarded cover for H .

Proof: Let Ξ be a set of guarded blocks of a hypergraph $H = \langle V, E \rangle$.

Suppose that Ξ is not a complete guarded cover for H . Choose $e \in E$ such that there is no guarded block $\langle \lambda, \chi \rangle \in \Xi$ for which $e \in \lambda$ and $e \subseteq \chi$. We will construct an instance P_e with structure H which will witness the fact that Ξ is not a guarded decomposition of H .

Let the domain of P_e be $D = \{0, 1\}$. For each edge $f \in E, f \neq e$, let the constraint of P_e with scope f allow all mappings from f to D . Finally let the constraint of P_e with scope e, c_e , allow all mappings from e to D except for the mapping that assigns the value 0 to all the vertices of e simultaneously.

Now let $\langle \lambda, \chi \rangle$ be any guarded block in the set Ξ . If $e \notin \lambda$ then the constraint generated by P_e on $\langle \lambda, \chi \rangle$ allows every assignment. On the other hand, if $e \in \lambda$, then, by the choice of e , we know that $e \not\subseteq \chi$. In this case we observe that any projection of c_e onto any proper subset of e allows all assignments. So again the constraint generated by P_e on $\langle \lambda, \chi \rangle$ allows every assignment.

Since all constraints generated by P_e on all elements of Ξ allow every assignment, the resulting CSP instance is not solution equivalent to P_e , and hence Ξ is not a guarded decomposition of H .

For the converse, suppose that Ξ is a complete guarded cover for H . Let P be an arbitrary CSP instance with structure H . We have to show that the CSP instance P_Ξ obtained by taking the constraints generated by P on each element of Ξ is solution equivalent to P .

Clearly, any solution to P is a solution to P_Ξ . On the other hand, by the completeness of Ξ , and the construction of P_Ξ , the projection of any solution to P_Ξ onto any edge $e \in E$ must be allowed by the constraint of P with scope e . Hence any solution to P_Ξ is also a solution to P . $\square$

4 Tractability

We have shown that guarded decompositions can be used to transform a given CSP instance to a solution equivalent instance which may have a different structure.

In order to be able to use guarded decompositions to identify *tractable* structural classes we need to impose some additional conditions to ensure we have the following properties.

Tractable construction It must be possible to generate each of the new constraints in polynomial time.

Tractable solution It must be possible to solve the resulting instances in polynomial time.

Tractable discovery It must be possible to obtain a guarded decomposition of the type we are considering for a given hypergraph (or else discover that no such decomposition exists) in polynomial time.

We will now consider extra conditions that can be imposed on a guarded decomposition to ensure that it has each of these properties.

4.1 Tractable Construction

The time complexity of a relational join operation is $O(r^k)$, where k is the number of relations being joined, and r is the maximum number of tuples in any of these relations. Hence to ensure that we have the *tractable construction* property, it is sufficient to bound the number of constraint relations that need to be combined using the relational join operation. This can be done by bounding the number of hyperedges in the guard of any guarded block used in the decomposition.

Definition 4.1 The **width** of a set of guarded blocks is the maximum number of hyperedges in any of its guards.

For any fixed value of k , the class of guarded decompositions of width at most k has the tractable construction property.

4.2 Tractable Solution

One way to ensure that the new instances obtained by using a guarded decomposition have the *tractable solution* property, is to ensure that the structure of these new instances is *acyclic* [Beeri et al., 1983]. The property of being acyclic can be defined as follows:

Definition 4.2 A **join tree** of a hypergraph H is a connected tree, T , whose nodes are the hyperedges of H , such that, whenever the vertex x of H occurs in two hyperedges e_1 and e_2 of H then x occurs in each node of the unique path connecting e_1 and e_2 in T . In other words, the set of nodes of T in which x occurs induces a (connected) subtree of T .

A hypergraph is called **acyclic** if it has a join tree.

Theorem 4.3 ([Gyssens et al., 1994]) Any CSP instance whose structure is acyclic can be solved in polynomial time.

Definition 4.4 A **join tree** of a set of guarded blocks Ξ of H is a connected tree, T , whose nodes are the elements of Ξ , such that, whenever the vertex x of H occurs in two blocks of Ξ then x occurs in each block of the unique path connecting them in T . In other words, the set of nodes of T for which x occurs in the block induces a (connected) subtree of T .

A set of guarded blocks is **acyclic** if it has a join tree.

By Theorem 4.3, any class of acyclic guarded decompositions has the tractable solution property. Using Theorem 3.5 and Definition 4.4 we now show that an acyclic guarded decomposition can be obtained from an arbitrary acyclic guarded cover without increasing the width, or significantly increasing the number of guarded blocks, by adding appropriate additional guarded blocks to make a complete guarded cover.

Theorem 4.5 *If the set of guarded blocks Ξ is an acyclic guarded cover for H then the set $\Xi \cup \{\langle\{e\}, e\rangle \mid e \in E\}$ is an acyclic guarded decomposition of H .*

In view of this result, we shall focus on acyclic guarded covers and use them to define classes of decompositions.

4.3 Tractable Discovery

The class of hypergraphs having a guarded cover in some fixed class Δ , with width at most k , will be denoted $C(\Delta, k)$.

Definition 4.6 *A class of guarded covers Δ will be called **parameterized tractable**² if for each value of k there exists a polynomial-time algorithm which, given any hypergraph $H \in C(\Delta, k)$, returns a guarded cover of H with width at most k , and given any hypergraph $H \notin C(\Delta, k)$, reports failure.*

It follows from the results above that if Δ is a class of acyclic guarded covers, which is parameterized tractable, then $C(\Delta, k)$ will be a tractable structural class of CSP instances, for each value of k .

5 Known Tractable Structural Classes

We will now show that many existing decomposition methods for the CSP can be defined in terms of finding acyclic guarded covers with particular special properties.

We first note that for historical reasons some existing decomposition methods for the CSP make use of *extended guards* that contain both vertices and hyperedges. To be able to present all of these methods in a uniform way we introduce the idea of transforming a CSP instance by adding a unary constraint c_v , for each variable v , where $c_v = \langle\{v\}, \{v\}^D\rangle$. This corresponds to extending the structure of the instance to ensure that it includes a hyperedge for each variable.

Definition 5.1 *Let $H = \langle V, E \rangle$ be any hypergraph. A guarded cover for the hypergraph $\langle V, E \cup \{\{v\} \mid v \in V\} \rangle$ is called an **extended cover** for H .*

The following result shows that the existence of an extended cover of width k is equivalent to the existence of a standard guarded cover of width k . This means that extended covers are simply a notational convenience and do not allow better decompositions.

Theorem 5.2 *If a hypergraph H has an acyclic extended cover of width k , then H also has an acyclic guarded cover of width at most k with at most the same number of guarded blocks.*

We also note that many existing decomposition methods are based on guarded covers in which every block is exactly the union of the edges of its guard.

Definition 5.3 *A guarded cover Ξ is **edge-defined** if the block of each guarded block in Ξ is exactly the set of vertices contained in the hyperedges of its guard.*

We can now characterize many known structural decomposition methods (see [Gottlob et al., 2000] for traditional definitions) in terms of a corresponding class of acyclic guarded covers with certain additional conditions.

²The notion of being parameterized tractable that we have defined here is strictly weaker than the well-known notion of being fixed parameter tractable.

In each of the cases listed below (except for query decompositions), it has been shown that the additional conditions are sufficient to ensure that the corresponding class of acyclic guarded covers, Δ , is parameterized tractable. Hence, in all cases (except query decompositions), the class of CSP instances in $C(\Delta, k)$ is a tractable structural class.

- A **biconnected-component** decomposition of a hypergraph is an edge-defined acyclic complete guarded cover, Ξ , satisfying an **articulation condition**:

$$\forall \langle \lambda_1, \chi_1 \rangle, \langle \lambda_2, \chi_2 \rangle \in \Xi, \quad |\chi_1 \cap \chi_2| \leq 1$$

- A **cycle-hypercutset** decomposition of a hypergraph $H = \langle V, E \rangle$ is an edge-defined acyclic complete guarded cover, Ξ , satisfying a **simplicity condition**:

$$\exists C \subseteq E, \forall \langle \lambda, \chi \rangle \in \Xi, \quad |\lambda - C| \leq 1$$

- A **hinge-tree** decomposition of a hypergraph H is an edge-defined acyclic complete guarded cover, Ξ , satisfying a **separation condition**:

$$\forall \langle \lambda_1, \chi_1 \rangle, \langle \lambda_2, \chi_2 \rangle \in \Xi, \exists e \in \lambda_1 \cap \lambda_2, \chi_1 \cap \chi_2 \subseteq e.$$

- A **cycle-cutset** decomposition of a hypergraph H is an edge-defined acyclic extended cover, Ξ , such that every hyperedge of every guarded block of Ξ consists of a single vertex, and Ξ satisfies a **simplicity condition**:

$$\exists C \subseteq V, \forall \langle \lambda, \chi \rangle \in \Xi, \exists e \in E, (\chi - C) \subseteq e$$

- A **query decomposition** of a hypergraph $H = \langle V, E \rangle$ is a pair $\langle \Xi, T \rangle$ where Ξ is a complete edge-defined acyclic extended cover of H and T is a join tree of Ξ that satisfies a **connectedness condition**:

$$\forall e \in E, \quad \{\langle \lambda, \chi \rangle \in \Xi \mid e \in \lambda\} \text{ is connected in } T.$$

- A **hypertree decomposition** of a hypergraph H is a pair $\langle \Xi, T \rangle$, where Ξ is an acyclic guarded cover and T is a rooted join tree of Ξ , which satisfies the following **descendant condition**:

$$\forall \langle \lambda, \chi \rangle \in \Xi, \quad \left(\left(\bigcup_{e \in \lambda} e \right) \cap \bigcup_{\langle \lambda_i, \chi_i \rangle \in D_{\langle \lambda, \chi \rangle}} \chi_i \right) \subseteq \chi$$

where $D_{\langle \lambda, \chi \rangle}$ is the set of all descendants of $\langle \lambda, \chi \rangle$ in T .

The minimum width of any hypertree decomposition of a hypergraph H is called the **hypertree width** of H .

6 Comparing Decompositions

The relative strengths of different decomposition techniques derived from acyclic guarded covers can be compared using the measures developed by Gottlob et al [2000].

Definition 6.1 *Let Δ_1 and Δ_2 be any two classes of guarded covers. We say that Δ_1 **generalizes** Δ_2 if there exists a value $\varepsilon > 0$ such that, for every k , $C(\Delta_2, k) \subseteq C(\Delta_1, k + \varepsilon)$.*

*We say that Δ_1 **strongly generalizes** Δ_2 if Δ_1 generalizes Δ_2 , and there exists k for which there does not exist $\varepsilon > 0$ with $C(\Delta_1, k) \subseteq C(\Delta_2, k + \varepsilon)$.*

Hypertrees have been shown to strongly generalize all the other parameterized tractable classes of decompositions defined in the previous section [Gottlob *et al.*, 2000]. Furthermore it has recently been shown that the hypertree width of any hypergraph is at most three times the minimal possible width of any acyclic guarded cover [Adler, 2004]. It follows that hypertrees cannot themselves be strongly generalized by any other class of acyclic guarded covers.

However, we will show in the next section that it is possible to define a parameterized tractable class Δ of acyclic guarded covers, such that for some families of hypergraphs, the minimal width of a guarded cover in Δ is smaller than the hypertree width by some constant factor. Such an improvement allows the corresponding instances to be solved exponentially faster than by using hypertree decompositions.

Definition 6.2 Let Δ_1 and Δ_2 be any two classes of guarded covers. We say that Δ_1 **can be superior** to Δ_2 if there is some $r > 1$, such that for every k , $C(\Delta_2, rk) \not\subseteq C(\Delta_1, k)$.

7 Spread Cuts

In this section we will define a new class of acyclic guarded covers, called *spread cuts*. We show that classes of instances with bounded spread cut width are tractable and that spread cuts can be superior to hypertrees.³

Throughout this section, we will write $\bigcup \lambda$ to refer to the vertices of a set of hyperedges λ ; that is, $\bigcup \lambda = \bigcup_{e \in \lambda} e$.

Definition 7.1 Let $H = \langle V, E \rangle$ be a hypergraph and $\chi \subseteq V$ be a set of vertices. We say that two hyperedges $e, f \in E$ are χ -**adjacent** if $e \cap f \not\subseteq \chi$.

A χ -**path** connecting e to f is a sequence $e = e_0, e_1, \dots, e_{r-1}, e_r = f$ such that e_i is χ -adjacent to e_{i+1} , for $i = 0, \dots, r-1$.

A set of hyperedges $C \subseteq E$ is χ -**connected** if, for every pair of hyperedges in C , there is a χ -path connecting them.

A set of hyperedges is a **hyperedge χ -component** of H if it is a maximal non-empty χ -connected subset of E .

A non-empty set of vertices C is a **vertex χ -component** of H if there is some hyperedge χ -component C_χ for which $C = \bigcup C_\chi - \chi$.

In this paper, unless otherwise stated, a component will mean a vertex component.

Example 7.2 Consider again the hypergraph H_{AG} defined in Example 2.5 and illustrated in Figure 1.

Now consider the guarded block $\langle \lambda, \chi \rangle$ with $\lambda = \{e_2, e_6\}$ and $\chi = \{3, 6, 7, 9, 10\}$.

There are three hyperedge χ -components: $\{e_6\}$, $\{e_1, e_2, e_7, e_8\}$ and $\{e_3, e_4, e_5\}$. Hence, there are two vertex χ -components: $\{4, 5\}$ and $\{1, 2, 8\}$.

There are four hyperedge $\bigcup \lambda$ -components: $\{e_2\}$, $\{e_6\}$, $\{e_1, e_7, e_8\}$ and $\{e_3, e_4, e_5\}$. Hence, there are two vertex $\bigcup \lambda$ -components: $\{4, 5\}$ and $\{1, 8\}$.

Notice that each vertex χ -component meets (has non-empty intersection with) exactly one vertex $\bigcup \lambda$ -component.

³It is still an open research question whether hypertrees can be superior to spread-cuts.

This special property of this guarded block relies on χ being “large enough”. If χ were empty, then there would be just one vertex χ -component and this component would meet both vertex $\bigcup \lambda$ -components. $\square$

If we restrict the possible blocks for a given guard appropriately then we can polynomially bound the number of guarded blocks, whilst still getting effective decompositions. This motivates the following definition.

Definition 7.3 A guarded block $\langle \lambda, \chi \rangle$ of a hypergraph H has **unbroken components** if each χ -component of H meets at most one $\bigcup \lambda$ -component of H and $\{e_1 \cap e_2 \mid e_1, e_2 \in \lambda\} \subseteq \chi$.

A **spread cut** of H is an acyclic guarded cover for H in which all of the guarded blocks have unbroken components.

Example 7.4 Consider again the hypergraph H_{AG} defined in Example 2.5 and illustrated in Figure 1.

The minimal width of any hypertree decomposition of H_{AG} is three⁴.

The following set of guarded blocks is a spread cut of H_{AG} of width two: $\langle \{e_2, e_6\}, \{3, 6, 7, 9, 10\} \rangle$, $\langle \{e_3, e_5\}, \{3, 4, 5, 6, 9, 10\} \rangle$, $\langle \{e_2, e_8\}, \{1, 2, 3, 8, 9, 10\} \rangle$, $\langle \{e_3, e_7\}, \{3, 7, 8, 9, 10\} \rangle$. $\square$

7.1 Spread Cuts are Parameterized Tractable

We will now define a canonical form for guarded blocks with unbroken components, and show that if a hypergraph H has a spread cut of width k , then it also has a spread cut of width at most k , in which each guarded block is canonical.

Definition 7.5 Let λ be any set of hyperedges of H . We define the **label**, $L_\lambda(v)$, of any vertex of H to be the set of hyperedge $\bigcup \lambda$ -components which include a hyperedge containing v .

We say that a guarded block $\langle \lambda, \chi \rangle$ is **canonical** if for each hyperedge $e \in \lambda$ the vertices of e outside of χ are exactly those with a particular label. That is,

$$\forall v \in e - \chi, \forall w \in e, \quad w \notin \chi \leftrightarrow L_\lambda(w) = L_\lambda(v).$$

Theorem 7.6 If H has a spread cut of width k then H has a spread cut of width at most k , in which each guarded block is canonical.

Proof: Due to space limitations, we can only give an outline of the proof.

The idea is to show that, for any guarded block $\langle \lambda, \chi \rangle$ with unbroken components, and for any hyperedge $e \in \lambda$, $|\bigcup_{v \in e - \chi} L_\lambda(v)| \leq 2$. Given this fact, we can start with any spread cut and, for each hyperedge of each guard, remove from the block any vertex with the same label as any vertices already outside of the block. It is straightforward to show that the result is still an acyclic guarded cover, and hence still a spread cut with the same width, where each guarded block is now canonical. $\square$

Using this canonical form, we can establish the following result.

Theorem 7.7 Spread cuts are parameterized tractable.

⁴This is most easily shown by using the “Robbers and Marshals” characterization of hypertree width [Gottlob *et al.*, 2003]. However, due to space restriction we omit this proof.

Proof: Due to space limitations, we can only give an outline of the proof.

The proof idea is that we only have to establish in polynomial time whether there exists a spread cut of width at most k in which each guarded block is canonical. We do this by first defining an object called a k -spread, which is a triple, $\langle \lambda, \chi, C \rangle$, where $\langle \lambda, \chi \rangle$ is any canonical guarded block with unbroken components, $|\lambda| \leq k$, and C is a χ -component.

We then define the predicate $k\text{-split}()$ (recursively) on k -spreads. We say that $k\text{-split}(\langle \lambda, \chi, C \rangle)$ holds if C is empty, or if there exists some canonical guarded block, $\langle \lambda_s, \chi_s \rangle$, where $\chi_s \subseteq C \cup \chi$ and, for every k -spread $\langle \lambda_s, \chi_s, C_s \rangle$:

$$C_s \cap C \neq \emptyset \Rightarrow \forall e \in E, (e \cap C_s \neq \emptyset \Rightarrow e \cap \chi \subseteq \chi_s), \text{ and } C_s \subseteq C \Rightarrow k\text{-split}(\langle \lambda_s, \chi_s, C_s \rangle).$$

It is easy to show that the number of k -spreads of a hypergraph $H = \langle V, E \rangle$ is at most $(|E| + 1)^{k+2}$. It follows (as in the corresponding proof for hypertrees, Lemma 14, Page 30, of Gottlob et al [2002]) that $k\text{-split}(\langle \emptyset, \emptyset, V \rangle)$ can be evaluated in at most $|E|^{4k+5}$ steps.

Finally, we show that H has a spread cut with canonical guarded blocks of width at most k if and only if $k\text{-split}(\langle \emptyset, \emptyset, V \rangle)$. The only hard part of this proof is to show that the set of guarded blocks used in the proof of $k\text{-split}(\langle \emptyset, \emptyset, V \rangle)$ is a guarded cover. This proof is exactly analogous to the corresponding proof for the algorithm that finds normal form hypertree decompositions (See the proof of Property 1 in Lemma 13, Page 28, of Gottlob et al [2002]). $\square$

The proof of Theorem 7.7 also shows that classes of instances with bounded spread cut width are tractable.

7.2 Spread Cuts and Hypertrees

To establish that spread cuts strongly generalize all of the parameterized tractable decomposition methods listed in Section 5 excluding hypertrees we simply refer to the proofs of strong generalization given by Gottlob et al [2000] and observe that every hypertree (of small width) used to establish strong generalization in that paper is also a spread cut.

To show that spread cuts can be superior to hypertrees we establish the following result.

Proposition 7.8 *There exists a family of hypergraphs $H_n, n = 1, 2, 3, \dots$, such that H_n has a spread cut with width $2n$, but the hypertree width of H_n is at least $3n$.*

Proof: Due to space restrictions we can only give an outline of the proof.

The idea is to use n copies of the hypergraph H_{AG} defined in Example 2.5 to construct H_n . We add edges linking vertices in different copies if they correspond to vertices of hyperedges that meet in H_{AG} . Using the “Robbers and Marshals” characterization of hypertree width [Gottlob et al., 2003], we then show that the hypertree width of H_n is at least $3n$. (If there are fewer than $3n$ marshals, the robber can always choose to play in the copy of H_{AG} with at most two marshals on its hyperedges.)

Finally, the width $2n$ spread cut is just the spread cut of H_{AG} with width 2 described in Example 7.4 “multiplied” over the n copies of H_{AG} . $\square$

8 Conclusion

We have introduced the general notion of a guarded decomposition and shown how it can be used to describe many known tractable structural subproblems of the CSP. We have then defined a new form of decomposition, the spread cut, which sits in the same place in the generalization hierarchy as hypertrees. We have shown that spread cuts are parameterized tractable and that they allow us to obtain decompositions with a smaller width than any hypertree decomposition for some classes of hypergraphs.

It is an open question whether the general class of all acyclic guarded covers is itself parameterized tractable.

References

- [Adler, 2004] I. Adler. Marshals, monotone marshals, and hypertree-width. *Journal of Graph Theory*, 47(4):275–296, December 2004.
- [Beeri et al., 1983] C. Beeri, R. Fagin, D. Maier, and M. Yannakakis. On the desirability of acyclic database schemes. *Journal of the ACM*, 30:479–513, 1983.
- [Dechter and Pearl, 1988] R. Dechter and J. Pearl. Network-based heuristics for constraint satisfaction problems. *Artificial Intelligence*, 34(1):1–38, 1988.
- [Dechter and Pearl, 1989] R. Dechter and J. Pearl. Tree clustering for constraint networks. *Artificial Intelligence*, 38:353–366, 1989.
- [Dechter, 2003] R. Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.
- [Freuder, 1985] E.C. Freuder. A sufficient condition for backtrack-bounded search. *Journal of the ACM*, 32:755–761, 1985.
- [Gottlob et al., 2000] G. Gottlob, N. Leone, and F. Scarcello. A comparison of structural CSP decomposition methods. *Artificial Intelligence*, 124:243–282, 2000.
- [Gottlob et al., 2002] G. Gottlob, N. Leone, and F. Scarcello. Hypertree decomposition and tractable queries. *Journal of Computer and System Sciences*, 64(3):579–627, 2002.
- [Gottlob et al., 2003] G. Gottlob, N. Leone, and F. Scarcello. Robbers, marshals, and guards: game theoretic and logical characterizations of hypertree width. *Journal of Computer and System Sciences*, 66:775–808, 2003.
- [Gyssens et al., 1994] M. Gyssens, P.G. Jeavons, and D.A. Cohen. Decomposing constraint satisfaction problems using database techniques. *Artificial Intelligence*, 66(1):57–89, 1994.
- [Kolaitis and Vardi, 2000] Ph.G. Kolaitis and M.Y. Vardi. Conjunctive-query containment and constraint satisfaction. *Journal of Computer and System Sciences*, 61:302–332, 2000.
- [Mackworth, 1977] A.K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8:99–118, 1977.
- [Marriott and Stuckey, 1998] K. Marriott and P. Stuckey. *Programming with Constraints*. MIT Press, Cambridge, Massachusetts, 1998.

Phase Transitions of Dominating Clique Problem and Their Implications to Heuristics in Satisfiability Search

Joseph Culberson, Yong Gao, Călin Anton

Department of Computing Science

University of Alberta

Edmonton, Alberta, Canada T6G 2E8

{joe,ygao,asy}@cs.ualberta.ca

Abstract

We study a monotone NP decision problem, the dominating clique problem, whose phase transition occurs at a very dense stage of the random graph evolution process. We establish the exact threshold of the phase transition and propose an efficient search algorithm that runs in super-polynomial time with high probability. Our empirical studies reveal two even more intriguing phenomena in its typical-case complexity: (1) the problem is “uniformly hard” with a tiny runtime variance on negative instances. (2) Our algorithm and its CNF-tailored implementation, outperform several SAT solvers by a huge margin on dominating cliques and some other SAT problems with similar structures.

1 Introduction

In this paper, we study the phase transition behavior of the dominating clique problem (DOMC), an NP-complete decision problem which for a given graph, asks if there is a subset of vertices that induces a clique and is a dominating set. Our interest in DOMC stems from an initial observation that its phase transition should occur when the random graph has a constant edge probability (i.e., has on average $\Omega(n^2)$ edges). This distinguishes DOMC from many other NP-complete problems previously studied from a phase transition perspective; Well-studied problems such as random SAT, random graph coloring, and Hamiltonian Cycle all have a phase transition when the underlying random graphs are sparse. For random SAT and graph coloring, it is known that randomly-generated instances become typically easy when the density parameter increases as a certain function of the problem size [Franco and Gelder, 2003; Krivelevich, 2002]. For Hamiltonian Cycle, whose edges-vertices ratio threshold is $\Omega(\log(n))$, instances randomly generated at the phase transition are already typically easy [Vandregend and Culberson, 1998].

Another unique feature is that DOMC is a monotone decision problem without any extra parameter other than the problem size and edge density. This is in contrast to other similar partitioning and covering problems studied in the phase transition literature, such as the vertex cover [Hartmann and

Weigt, 2003], in which an additional size parameter has to be used as an input.

There has also been recent interest in the study of the typical-case behavior of random problems with some deterministic structures, hoping that these problems might be able to capture more characteristics in real-world applications. Among others, the Quasigroup completion problem [Gomes and Shmoys, 2002], problems defined on the small-world graphs [Walsh, 1999], and problems obtained by “morphing” different problems [Gent *et al.*, 1999] have been proposed as alternatives to random problems with a uniform distribution. The dominating clique problem on random graphs and its CNF-encoding provide another type of distribution with (in some sense) extreme characteristics.

In Section 2, we establish the exact threshold of the phase transition of the dominating clique problem in random graphs. In Section 3, we present a backtracking algorithm for DOMC which will be used in our empirical investigation. In Section 4, we report two sets of interesting observations from our empirical investigation. We conclude in Section 5.

2 Dominating Clique Problem and its Phase Transitions

Consider a graph $G(V, E)$. A subset of vertices $V' \subset V$ is called a *dominating clique* of $G(V, E)$ if V' induces a clique and for any $v \in V \setminus V'$, there is a vertex $u \in V'$ such that $(u, v) \in E$. The dominating clique problem is the one that asks the question whether a given graph has a dominating clique.

The property of having a dominating clique is monotone and has a phase transition when the random graph is at the dense stage of the graph evolution process. The following theorem shows the phase transition has an exact threshold $p^* = \frac{3-\sqrt{5}}{2}$ for a random graph $G(n, p)$ where each of the $n(n-1)/2$ potential edges appears in the graph with probability p .

Theorem 2.1.

$$\lim_n \Pr\{G(n, p) \text{ has a dominating clique}\} = \begin{cases} 0, & \text{if } p < \frac{3-\sqrt{5}}{2}; \\ 1, & \text{if } p > \frac{3-\sqrt{5}}{2}. \end{cases} \quad (2.1)$$

To simplify the notations, we will write $b = \frac{1}{p}$. Let $DC(n, r)$ be the number of dominating cliques of size r . The expectation of $DC(n, r)$ is

$$E[DC(n, r)] = \binom{n}{r} p^{\binom{r}{2}} (1 - (1 - p)^r)^{n-r}. \quad (2.2)$$

By Stirling's formula, we have

$$E[DC(n, r)] \sim \frac{1}{\sqrt{2\pi r}} \left(\frac{enp^{(r-1)/2}}{r} \right)^r (1 - (1 - p)^r)^{n-r}. \quad (2.3)$$

Lemma 2.1. *If the edge probability p is such that $\log_b(b - 1) > \frac{1}{2}$, then*

$$\lim_n \sum_{r=1}^n E[DC(n, r)] = 0. \quad (2.4)$$

Proof. Let

$$A(n) = \sum_{r=1}^{2 \log_b n} \binom{n}{r} p^{\binom{r}{2}} (1 - (1 - p)^r)^{n-r}$$

and

$$B(n) = \sum_{r > 2 \log_b n} \binom{n}{r} p^{\binom{r}{2}} (1 - (1 - p)^r)^{n-r}$$

Based on the bound $\binom{n}{r} \leq \left(\frac{en}{r}\right)^r$, $B(n)$ can be estimated as follows:

$$\begin{aligned} B(n) &= \sum_{r > 2 \log_b n} \binom{n}{r} p^{\binom{r}{2}} (1 - (1 - p)^r)^{n-r} \\ &\leq \sum_{r > 2 \log_b n} \left(\frac{en}{r}\right)^r p^{\binom{r}{2}} \\ &= \sum_{r > 2 \log_b n} \left(\frac{\sqrt{ben}}{rb^{\frac{r}{2}}}\right)^r \\ &\leq \sum_{r > 2 \log_b n} \left(\frac{\sqrt{be}}{r}\right)^r \end{aligned}$$

Since $\frac{\sqrt{be}}{r} < 1$ if n is sufficiently large, we have $\lim_n B(n) = 0$.

For $A(n)$, we have

$$\begin{aligned} A(n) &= \sum_{r=1}^{2 \log_b n} \binom{n}{r} p^{\binom{r}{2}} (1 - (1 - p)^r)^{n-r} \\ &\leq \sum_{r=1}^{2 \log_b n} n^r \left(1 - \left(\frac{b-1}{b}\right)^r\right)^{n-r} \\ &\leq (2 \log_b n) e^{2 \ln n \log_b n} \left(1 - \left(\frac{b-1}{b}\right)^{2 \log_b n}\right)^{n-2 \log_b n} \\ &\sim (2 \log_b n) e^{2 \ln n \log_b n} e^{-\left(\frac{b-1}{b}\right)^{2 \log_b n} (n-2 \log_b n)} \\ &= (2 \log_b n) e^{2 \ln n \log_b n} e^{-\frac{1}{n^2} n^{2 \log_b(b-1)} (n-2 \log_b n)} \\ &\sim (2 \log_b n) e^{2 \ln n \log_b n} e^{-n^{2 \log_b(b-1)-1}} \end{aligned}$$

Therefore, we have $\lim_n A(n) = 0$ if $\log_b(b - 1) > \frac{1}{2}$. $\square$

Solving the inequality about b , we get $b > \frac{3+\sqrt{5}}{2}$, that is, $p < \frac{3-\sqrt{5}}{2} = 0.381966 \dots$. To prove the case of $p > \frac{3-\sqrt{5}}{2}$, i.e., $\log_b(b - 1) < \frac{1}{2}$, we have to consider the variance of the number of dominating cliques in $G(n, p)$. Details will be given in the Appendix. Here, we only mention that the first crucial observation is that in this case, we have (A) $\frac{1}{1-\log_b(b-1)} < 2$ and (B) the expected number of dominating cliques of size $c \log_b n$ tends to infinity for any constant c such that $\frac{1}{1-\log_b(b-1)} < c < 2$.

Lemma 2.2. *Let c be a constant such that $\frac{1}{1-\log_b(b-1)} < c < 2$ and let $r = c \log_b n$. Then,*

$$\lim_n E[DC(n, r)] = +\infty.$$

Proof. By (2.3), we have

$$\begin{aligned} E[DC(n, r)] &\sim \frac{1}{\sqrt{2\pi r}} \left(\frac{\sqrt{ben}}{rb^{\frac{r}{2}}}\right)^r e^{-n^{c(\log_b(b-1)-1)+1}} \\ &= \frac{1}{\sqrt{2\pi r}} \left(\frac{en^{1-\frac{c}{2}}}{c \log_b n}\right)^{c \log_b n} e^{-n^{c(\log_b(b-1)-1)+1}}. \end{aligned}$$

The lemma follows since by assumption we have $c(\log_b(b - 1) - 1) + 1 < 0$ and $1 - \frac{c}{2} > 0$. $\square$

3 An Algorithm for the Dominating Clique Problem

We present a brief overview of the elementary algorithm we developed for this problem.

We assume we are given a simple undirected graph $G = (V, E)$. The algorithm is a basic depth first backtrack algorithm that recursively constructs a potential dominating clique D . We call the set of vertices not adjacent to any vertex in D the uncovered set U . For a vertex to be a candidate for addition to D it must be adjacent to all vertices in D . We call this the selection set $S = \{x : \forall w \in D, \{x, w\} \in E\}$.

At each node in the backtrack tree, we select a pivot set $P \subseteq S$ such that if the current D can be extended to cover the graph at least one of the vertices must be in P . To this end, we first find a vertex $w \in U$ which has a minimum number of neighbors in S . Clearly either w or one of its neighbors in the current S must be in the final D . The effect of this is to greatly reduce the branching factor and the amount of redundant search.

For $A \subseteq V, w \in V$ we define the closed neighborhood of a vertex w in a subset A as $\hat{N}(A, w) = \{x \in A : x = w \text{ or } \{x, w\} \in E\}$, and the open neighborhood $N(A, w) = \hat{N}(A, w) \setminus \{w\}$. Initially, $U = S = V$ and $D = \emptyset$.

At first glance it appears we have eliminated the vertex of minimum degree, w , as a possible member of D . Suppose w is the only vertex in D , then since w is of minimum degree G is a clique and any vertex will do. Otherwise, on the initial step w will be in S' and so is available for inclusion later in the recursion. Correctness then follows by induction on the

```

DomClq( $G, S, D, U$ ) :: BOOLEAN
  If  $U = \emptyset$  then
     $DOMCLQ = D$ .
    return yes.
  endif
  Find  $w \in U$  such that  $|N(S, w)| = \min_{v \in U} |N(S, v)|$ .
   $P = N(S, w)$ .
  if  $P = \emptyset$  return no.
   $S'' = S$ .
  while  $P \neq \emptyset$ 
    Select  $v \in P$ .
     $D = D \cup \{v\}$ .
     $S' = N(S'', v)$ .
     $U' = U \setminus \hat{N}(U, v)$ .
    If DomClq( $G, S', D, U'$ ) return yes.
     $D = D \setminus \{v\}$ .
     $S'' = S'' \setminus \{v\}$ .
     $P = P \setminus \{v\}$ .
  endwhile
  return no;
end DomClq.

```

observations given above. Our assumption is that usually the vertex of minimum degree is not in D and so we may reduce the search cost by leaving it out of the initial P . Note that after the initial step S and U are disjoint.

In our program we made no effort to reorder the vertices of P for selection (first statement in the while loop). One obvious candidate is to select $v \in P$ to maximize $N(U, v)$. We would guess that in positive instances this would tend to find solutions quicker. However, the results for the negative instances in table 2 indicate this is unlikely to help in those cases, these are the most expensive and there would be a fair bit of overhead cost to this heuristic. We leave further investigation of selection heuristics as future work.

4 Empirical Studies

In this section, we discuss our empirical investigation on the dominating clique problem at phase transitions. In the first set of experiments, we study the threshold behavior and typical-case hardness of the dominating clique problem itself. In the second set of experiments, we compare the performance of our algorithm and some state-of-the-art SAT solvers.

4.1 Threshold and Uniformity of Hardness at Phase Transitions

Table 1 shows the preliminary result on empirical value of the threshold. Table 2 summarizes the typical case hardness at $p = 0.371$ for $n = 250 \dots 1000$. The data indicate that the typical-case hardness at phase transitions increases at a super-polynomial rate. More interesting is the fact that for negative instances the variance in the number of the backtracks is tiny as compared to the total number. In other words, randomly-generated dominating clique instances just beyond the phase transition seem to be “uniformly” hard. This suggests that randomizing the selection within the pivot set P is unlikely

n	Average	Std.	Min	Max
50	0.3725	0.0203	0.3176	0.4106
75	0.3698	0.0150	0.3258	0.4040
100	0.3685	0.0109	0.3358	0.3826
125	0.3700	0.0110	0.3406	0.3951
150	0.3663	0.0110	0.3316	0.3817
175	0.3660	0.0096	0.3376	0.3794
200	0.3689	0.0078	0.3502	0.3853
225	0.3680	0.0075	0.3448	0.3827
250	0.3669	0.0072	0.3483	0.3842
275	0.3669	0.0084	0.3445	0.3814
300	0.3674	0.0060	0.3523	0.3775
325	0.3685	0.0073	0.3434	0.3784
350	0.3671	0.0083	0.3407	0.3803
500	0.3675	0.0063	0.3357	0.3786
750	0.3680	0.0061	0.3568	0.3770

Table 1: The average threshold of Dominating Clique, expressed as a fraction of $\binom{n}{2}$ possible edges, for various n . Fifty instances at each n , except $n=750$ where only 8 instances completed. Std is the standard deviation, Min (Max) is the minimum (maximum) fraction encountered in the sample.

to produce the heavy-tailed running time behavior frequently observed in many other problems [Gomes *et al.*, 2000].

4.2 Significance of Using the Right Heuristics: How SAT solvers fail?

The dominating clique problem has a very natural and simple CNF encoding. To make the presentation easier, we adopt the convention that a clause can be viewed as a set of literals. Given a graph $G = G(V, E)$ with $V = \{v_1, \dots, v_n\}$, the dominating clique problem can be encoded as follows.

1. There are n boolean variables $\{x_i, 1 \leq i \leq n\}$ where $x_i = 1$ signifies that the corresponding vertex v_i is in the dominating clique.
2. For each variable x_i , there is a clause $C_i = \{x_i\} \cup N(x_i)$, where $N(x_i)$ means the variables representing the vertices adjacent to v_i . This indicates that each vertex has to be in the dominating clique or have a neighbor in a dominating clique. These clauses will be called the *long-positive clauses*.
3. For each pair of variables (x_i, x_j) not in E , there is a clause $D_{i,j} = \{\bar{x}_i, \bar{x}_j\}$, indicating that at most one of the two non-adjacent vertices can be in a dominating clique. We will call these clauses the *short-negative clauses*.

Other encodings are also possible such as the support encoding [Gent, 2002] that maps maintaining arc-consistency of CSP algorithms to unit-propagation in SAT algorithms. But similar CNF encoding of the CSP representation for the dominating clique problem at the phase transitions will result in $\Omega(n^3)$ clauses of length $\Omega(n)$.

Backtrack nodes for YES instances				
N	Mean	Std.	Min.	Max
600	179031.3	127471.9	836.0	517596.0
700	347996.4	266229.1	460.0	1043853.0
800	658643.6	496335.5	1890.0	2199760.0
900	1484544.5	978648.8	100955.0	3992411.0
1000	2354966.6	1796775.3	9259.0	6586573.0
Backtrack nodes for NO instances				
N	Mean	Std.	Min.	Max
600	525205.8	7602.0	510705.0	537705.0
700	1121825.4	13552.0	1089553.0	1153289.0
800	2205960.3	19017.3	2166309.0	2254168.0
900	4063264.6	32241.0	3989194.0	4138323.0
1000	7103729.3	55926.8	6958133.0	7218638.0

Table 2: The number of backtrack nodes split on instances with and without a dominating clique. 100 instances at each N, $p = 0.371$. Note the remarkable consistency in the NO instances.

A SAT algorithm for the CNF-encoding of DOMC

We noticed that SATZ and ZChaff cannot solve any of the DOMC instances of size more than 600 in reasonable time. Therefore we translated our dominating clique algorithm to a SAT solver, called DCS, that solves the CNF-encoding of the dominating clique problem almost as efficiently as the original algorithm.

The solver does not use any of the current efficient data structures such as the technique of watched literals, nor does it use any advanced learning mechanism and heuristics, other than those directly translated from our dominating clique search algorithm. In the context of CNF formulas our algorithm reads as follows: (1) always pick an unassigned variable in the shortest remaining long-positive clause; (2) conduct a one-step unit propagation based on the information in short-negative clauses; and (3) stop as soon as all the long positive clauses are satisfied, or backtrack if a long-positive clause becomes empty. As has been discussed at the end of Section 3, our solver’s performance may be further improved on satisfiable instances if we add the heuristic to select an unassigned variable that satisfies a maximum number of long-positive clauses.

N	DCS	BM	MQ	SZ	Solution Prob.
150	0.04	0.119	12.715	1.73	0.47
300	1.11	3.904	≥ 600	58.93	0.51

Table 3: Median running time (in seconds) of SAT solvers on random instances of the dominating clique problem. 100 instances for each N. The last column is the percentage of the satisfiable instances. Cutoff time is 600 seconds.

Results on the CNF encodings of DOMC and Subgraph Isomorphism

We have performed a series of experiments comparing the performance of our solver DCS and some state-of-the-art

SAT solvers, including BerkMin(BM)¹, MarchEq(MQ)² and Satz(SZ).

First, we compared the performance of our solver DCS and other solvers on the CNF encoding of DOMC at phase transitions. We generated 100 random instances of the dominating clique problem at the edge probability $p = 0.368$ for $n = 150$ and 300. According to our experiment on the threshold, $p = 0.368$ is roughly the edge probability where 50 percent of the instances are satisfiable. The median running time (in seconds) of these solvers is summarized in Table 3.

Inspired by the efficiency of DCS on DOMC we investigated its performance on Subgraph Isomorphism (SGI) instances. Given two graphs $G(V_G, E_G)$ and $H(V_H, E_H)$, the *subgraph isomorphism problem* asks if there exists a subgraph H_1 of H such that G is isomorphic to H_1 . A SGI instance can be naturally converted to SAT using an encoding similar to the one used for DOMC (direct encoding); the resulting SAT instance has $|V_G| \times |V_H|$ variables, $|V_G|$ long positive clauses and $O(|V_G|^2 \times |V_H|^2)$ short-negative clauses. SAT encodings of random SGI (SERSGI) are difficult for (deterministic) SAT solvers.³

We generated 5 sets of SERSGI instances with 528 variables. Each set consisted of 50 instances generated around the cross over point of SGI. DCS consistently outperformed all the other solvers on each set (Table 4). DCS is one or two orders of magnitude faster than MarchEq and Satz. This is remarkable especially considering the simplicity of DCS. To further investigate the speed up factor we compared the performances of DCS and BerkMin on the SERSGI instances used in the 2004 SAT competition (Table 5). With a single exception, DCS performance was comparable or better than that of BerkMin. Thus we conclude that DCS outperforms state of the art solvers on SERSGI instances.

It may be possible to extend this conclusion to other classes of SAT instances which have only long-positive and short negative clauses, but more work needs to be done to see what are the structural properties that DCS can exploit. However, we do not expect that DCS will work efficiently on any type of CNF formulas with a long-short clause structure. We noticed that on random k-SAT instances, it is at least one order of magnitude slower than Satz. We consider this behavior an evidence against the idea of an efficient universal sat solver that does not consider any problem structure.

5 Conclusions

Most of the well-investigated NP-complete problems in the literature have a phase transition when the underlying graph is very sparse. There is also evidence that randomly-generated instances at the phase transition are typically not hard if the

¹BerkMin is the predecessor of Forklift the winner of the Industrial Category at the 2003 SAT competition. Our experience indicates that Zchaff (winner of 2004) is not efficient for this type of instances

²Winner of the Hand Crafted Category at the 2004 SAT competition

³At the 2004 SAT competition[LeBerre and Simon, 2004], no deterministic solver could solve any SERSGI instance.

	Set 1	Set 2	Set 3	Set 4	Set 5
% Solved					
DCS	100	100	100	96	100
BM	100	90	80	58	44
MQ	94	66	36	12	2
SZ	50	22	2	14	2
Median Solution Time (sec)					
DCS	14.48	19.14	54.92	131.23	135.50
BM	30.8	97.03	250.00	919.28	≥ 1200
MQ	233.74	905.15	≥ 1200	≥ 1200	≥ 1200
SZ	1111.75	≥ 1200	≥ 1200	≥ 1200	≥ 1200
Mean Solution Time (sec)					
DCS	42.09	44.21	80.98	209.07	240.47
BM	90.47	283.00	479.91	-	-
MQ	383.20	-	-	-	-
SZ	-	-	-	-	-

Table 4: Performances of DCS, BerkMin, MarchEq and Satz on 5 sets of SERSGI instances with 528 variables. 50 instances in each set. Cutoff time 1200 seconds.

Inst#	776	770	772	774	767	765	767
DCS	79.1	73.7	8.7	86.3	128.0	121.4	18.35
BM	526.4	33.3	58.5	79.9	2930.9	366.9	28.21

Table 5: Running time (minutes) of DCS and BerkMin on SERSGI instances used in 2004 SAT competition.

threshold can be accurately established such as the Hamiltonian Cycle problem. DOMC studied in this paper is distinguished from a phase transition and typical-case complexity perspective. On the one hand, it has an exact threshold at a very dense stage of the random graph evolution process. Our empirical results, on the other hand, indicate that randomly-generated instances are still typically (and uniformly for negative instances) hard. As has been demonstrated by our SAT solver's outstanding performance advantage on the CNF-encoding of DOMC and other problems with similar structures, DOMC might serve as a prototypical problem to further investigate the relationship among the problem structure, the encoding efficiency, and the use of relevant heuristics in satisfiability search.

Appendix: proof of Theorem 2.1

To estimate the second moment of the number of dominating cliques of size r , we need the following lemma on the probability that two vertex subsets both dominate the rest of the vertices in a random graph.

Lemma 5.1. *Let U and W be two vertex subsets of size r such that $|U \cap W| = l$ and let $P(l)$ be the probability that U and W both dominate all the vertices outside of $U \cup W$. Then,*

1.

$$P(l) = [1 - 2(1-p)^r + (1-p)^{2r-l}]^{n-(2r-l)} \quad (5.5)$$

2. Let $Q(n, r) = (1 - (1-p)^r)^{2n-2r}$. We have

(a) $\lim_n \frac{P(0)}{Q(n, r)} = 1$; and

(b) There exists a constant C^* and N^* such that if $n > N^*$, $\frac{P(l)}{Q(n, r)} < C^*$ for all $1 \leq l \leq r$.

Proof. (1). Let v be a vertex outside of $U \cup W$. The probability that v is not connected to any vertices in $U \cap W$ is $(1-p)^l$. The probability that v is dominated by both $U \setminus W$ and $W \setminus U$ is

$$(1 - (1-p)^{r-l})^2.$$

Thus, $P(l)$ is

$$\begin{aligned} & [(1 - (1-p)^l) + (1-p)^l(1 - (1-p)^{r-l})^2]^{n-(2r-l)} \\ &= [1 - (1-p)^l + (1-p)^l(1 - 2(1-p)^{r-l} \\ &\quad + (1-p)^{2(r-l)})]^{n-(2r-l)} \\ &= [1 - 2(1-p)^r + (1-p)^{2r-l}]^{n-(2r-l)} \end{aligned}$$

(2). Since $\log_b(b-1) - 1 < 0$, we have

$$\begin{aligned} \lim_n \frac{P(0)}{Q(n, r)} &= \lim_n \frac{(1 - (1-p)^r)^{2(n-2r)}}{(1 - (1-p)^r)^{2n-2r}} \\ &= \lim_n (1 - (1-p)^r)^{-2r} \\ &= \lim_n e^{-(1-p)^r(-2r)} \\ &= \lim_n e^{2rn^{c(\log_b(b-1)-1)}} \\ &= 1. \end{aligned}$$

For $1 \leq l \leq r$, we have

$$\begin{aligned} \frac{P(l)}{Q(n, r)} &= \frac{[1 - 2(1-p)^r + (1-p)^{2r-l}]^{n-(2r-l)}}{(1 - (1-p)^r)^{2n-2r}} \\ &\leq \frac{[1 - 2(1-p)^r + (1-p)^{2r-l}]^{n-2r}}{(1 - (1-p)^r)^{2n-2r}} \\ &\leq \frac{[1 - 2(1-p)^r + (1-p)^r]^{n-2r}}{(1 - (1-p)^r)^{2n-2r}} \\ &= \left(\frac{1}{1 - (1-p)^r} \right)^n \end{aligned}$$

Since

$$\begin{aligned} \lim_n \left(\frac{1}{1 - (1-p)^r} \right)^n &= \lim_n \left(1 + \frac{(1-p)^r}{1 - (1-p)^r} \right)^n \\ &= \lim_n e^{n(1-p)^r \frac{1}{1 - (1-p)^r}} \\ &= O(1) \lim_n e^{n^{c(\log_b(b-1)-1)+1}} \end{aligned}$$

and $c(\log_b(b-1) - 1) + 1 < 0$, conclusion 2(a) follows. $\square$

Proof of Theorem 2.1. The fact that

$$\lim_n \Pr\{G(n, p) \text{ has a dominating clique}\} = 0$$

for $p < \frac{3-\sqrt{5}}{2}$ follows from Lemma 2.1 and Markov's inequality.

To prove that $\lim_n Pr\{G(n, p) \text{ has a dominating clique}\} = 1$ for $p > \frac{3-\sqrt{5}}{2}$, we will show that

$$\lim_n Pr\{DC(n, r) = 0\} = 0$$

where $r = c \log_b n$ and $\frac{1}{1-\log_b(b-1)} < c < 2$. From Lemma 2.2, we have

$$\lim_n E[DC(n, r)] = +\infty.$$

By Chebyshev's inequality, we have

$$Pr\{DC(n, r) = 0\} \leq \frac{E[DC^2(n, r)]}{E^2[DC(n, r)]} - 1.$$

The expectation of $DC^2(n, r)$ is

$$E[DC^2(n, r)] = \sum_{l=0}^r \binom{n}{r} \binom{r}{l} \binom{n-r}{r-l} p^{2\binom{r}{2}-\binom{l}{2}} P(l).$$

where $\binom{n}{r} \binom{r}{l} \binom{n-r}{r-l}$ is the total number of pairs of vertex subsets of size r that have l vertices in common, and $p^{2\binom{r}{2}-\binom{l}{2}} P(l)$ is the probability that such a pair of vertex subsets are both dominating cliques. Recall that

$$\begin{aligned} E^2[DC(n, r)] &= \left(\binom{n}{r} p^{\binom{r}{2}} (1 - (1-p)^r)^{n-r} \right)^2 \\ &= \binom{n}{r}^2 p^{2\binom{r}{2}} Q(n, r). \end{aligned}$$

We write $\frac{E[DC^2(n, r)]}{E^2[DC(n, r)]}$ as

$$\begin{aligned} \frac{E[DC^2(n, r)]}{E^2[DC(n, r)]} &= \frac{\sum_{l=0}^r \binom{n}{r} \binom{r}{l} \binom{n-r}{r-l} p^{2\binom{r}{2}-\binom{l}{2}} P(l)}{\binom{n}{r}^2 p^{2\binom{r}{2}} Q(n, r)} \\ &= a_n + b_n, \end{aligned}$$

where

$$\begin{aligned} a_n &= \frac{\sum_{l=0}^1 \binom{n}{r} \binom{r}{l} \binom{n-r}{r-l} p^{2\binom{r}{2}-\binom{l}{2}} P(l)}{\binom{n}{r}^2 p^{2\binom{r}{2}} Q(n, r)} \\ &= \binom{n}{r}^{-1} \left[\binom{n-r}{r} \frac{P(0)}{Q(n, r)} + r \binom{n-r}{r-1} \frac{P(1)}{Q(n, r)} \right]. \end{aligned}$$

and

$$b_n = \binom{n}{r}^{-1} \sum_{l=2}^r \binom{r}{l} \binom{n-r}{r-l} p^{-\binom{l}{2}} \frac{P(l)}{Q(n, r)}.$$

We will prove that $\lim_n a_n = 1$ and $\lim_n b_n = 0$. Since $\frac{\binom{n-r}{r}}{\binom{n}{r}} \rightarrow 1$ and $\frac{r \binom{n-r}{r-1}}{\binom{n}{r}} \rightarrow 0$, by Lemma 5.1(2.a and 2.b) we have

$$\lim_n a_n = 1.$$

From Lemma 5.1 (2.b), we have for $n > N^*$

$$\begin{aligned} b_n &\leq C^* \sum_{l=2}^r \binom{n}{r}^{-1} \binom{r}{l} \binom{n-r}{r-l} p^{-\binom{l}{2}} \\ &\leq O(1) \sum_{l=2}^r \frac{r^{2l}}{l! n^l} b^{\binom{l}{2}} \\ &= O(1) \sum_{l=2}^r \left(\frac{r^2 b^{l/2}}{n \sqrt{b}} \right)^l \end{aligned}$$

Since $r = c \log_b n$ with $c < 2$, we have

$$b_n \leq O(1) r \left(\frac{r^2}{n^{1-c/2} \sqrt{b}} \right)^r \rightarrow 0.$$

This completes the proof. $\square$

References

- [Franco and Gelder, 2003] J. Franco and A. V. Gelder. A perspective on certain polynomial-time solvable classes of satisfiability. *Discrete Applied Mathematics*, 125(2-3):177–214, 2003.
- [Gent et al., 1999] I. Gent, H. Hoos, P. Prosser, and T. Walsh. Morphing: Combining structure and randomness. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI'99)*, pages 654–660, Orlando, Florida, 1999.
- [Gent, 2002] I. Gent. Arc consistency in SAT. In *Proceedings of the 15th European Conference on Artificial Intelligence (ECAI 2002)*, pages 121–125, 2002.
- [Gomes and Shmoys, 2002] C. Gomes and D. Shmoys. Completing quasigroups or latin squares: A structured graph coloring problem. In *Proceedings of the Computational Symposium on Graph Coloring and Extensions*, 2002.
- [Gomes et al., 2000] C. Gomes, B. Selman, N. Crato, and H. Kautz. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *Journal of Automated Reasoning*, 24(1-2):67–100, 2000.
- [Hartmann and Weigt, 2003] A. Hartmann and M. Weigt. Statistical mechanics of the vertex-cover problem. *Journal of Physics A: Mathematical and General (a special issue)*, 36(43):11069–11093, 2003.
- [Krivelevich, 2002] M. Krivelevich. Coloring random graphs - an algorithmic perspective. In *Proceedings of the 2nd Colloquium on Mathematics and Computer Science (MathInfo'2002)*, pages 175–195. Birkhauser, 2002.
- [LeBerre and Simon, 2004] D. LeBerre and L. Simon. SAT competition - webpage, 2004. Available from: satlive.org/SATCompetition/2004.
- [Vandegriend and Culberson, 1998] B. Vandegriend and J. Culberson. The $G_{n,m}$ phase transition is not hard for the Hamiltonian Cycle problem. *Journal of Artificial Intelligence Research*, 9:219–245, 1998.
- [Walsh, 1999] T. Walsh. Search in a small world. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99)*, pages 1172–1177, 1999.

Existential arc consistency: Getting closer to full arc consistency in weighted CSPs*

Simon de Givry
degivry@toulouse.inra.fr
Matthias Zytnicki
zytnicki@toulouse.inra.fr
INRA
Toulouse, France

Federico Heras
fheras@lsi.upc.edu
Javier Larrosa
larrosa@lsi.upc.edu
LSI, UPC
Barcelona, Spain

Abstract

The weighted CSP framework is a soft constraint framework with a wide range of applications. Most current state-of-the-art complete solvers can be described as a basic depth-first branch and bound search that maintain some form of arc consistency during the search. In this paper we introduce a new stronger form of arc consistency, that we call *existential directional arc consistency* and we provide an algorithm to enforce it. The efficiency of the algorithm is empirically demonstrated in a variety of domains.

1 Introduction

Weighted constraint satisfaction problems (WCSP) is a well-known optimization version of the CSP framework with many practical applications. Recently, the celebrated arc consistency property has been generalized from the classical CSP framework to WCSP [Cooper and Schiex, 2004; Larrosa and Schiex, 2004]. There are three known generalizations: AC*, FDAC* and FAC*, all of them collapsing to classical arc consistency in the CSP case. In [Larrosa and Schiex, 2003] it was shown that maintaining FDAC* during search was usually the best option.

In this paper we introduce a new form of local consistency called *existential directional arc consistency* (EDAC*). We show that EDAC* is stronger than FDAC* and we introduce an algorithm to enforce EDAC* that runs in time $O(ed^2 \times \max\{nd, \top\})$, where e , n and d are the number of constraints, variables and domain values, respectively; $\top$ is the upper bound of the WCSP instance (to be defined later). The main use of EDAC* and its associated filtering algorithm is to embed it into a complete depth-first branch and bound solver that *maintains* EDAC* at every visited node. We have experimentally evaluated this idea on random Max-SAT, Max-CSP, and *uncapacitated warehouse location problems*. We observe that maintaining EDAC* is never worse than maintaining FDAC* and, in many instances, it is orders of magnitude better.

*This research is partially supported by the French-Spanish collaboration PICASSO 05158SM - Integrated Action HF02-69 and the REPLI project TIC-2002-04470-C03.

2 Preliminaries

Valuation structures are algebraic objects to specify costs in valued constraint satisfaction problems [Schiex *et al.*, 1995]. They are defined by a triple $S = (E, \oplus, \succeq)$, where E is the set of costs totally ordered by $\succeq$. The maximum and a minimum costs are noted $\top$ and $\perp$, respectively. $\oplus$ is an operation on E used to combine costs.

Following [Larrosa and Schiex, 2004], the valuation structure of Weighted CSP (WCSP) is, $S(k) = ([0..k], \oplus, \succeq)$ where $k > 0$ is a natural number; $\oplus$ is defined as $a \oplus b = \min\{k, a + b\}$; $\succeq$ is the standard order among naturals. Observe that in $S(k)$, we have $0 = \perp$ and $k = \top$. It is useful to define the *subtraction* $\ominus$ of costs. Let $a, b \in [0..k]$ be two costs such that $a \geq b$,

$$a \ominus b = \begin{cases} a - b & : a \neq k \\ k & : a = k \end{cases}$$

A binary *weighted constraint satisfaction problem* (WCSP) is a tuple $P = (S(k), \mathcal{X}, \mathcal{D}, C)$. $S(k)$ is the valuation structure. $\mathcal{X} = \{x_1, \dots, x_n\}$ is a set of variables that we will often call by just their index. Each variable $x_i \in \mathcal{X}$ has a finite domain $D_i \in \mathcal{D}$ of values that can be assigned to it. (i, a) denotes the assignment of value $a \in D_i$ to variable x_i . C is a set of unary and binary weighted constraints (namely, cost functions) over the valuation structure $S(k)$. A unary weighted constraint C_i is a cost function $C_i(x_i) \rightarrow [0..k]$. A binary constraint C_{ij} is a cost function $C_{ij}(x_i, x_j) \rightarrow [0..k]$. We assume the existence of a unary constraint C_i for every variable, and a *zero-arity* constraint (i.e. a constant), noted $C_\emptyset$ (if no such constraint is defined, we can always define *dummy* ones: $C_i(x_i) = \perp$, $C_\emptyset = \perp$).

When a constraint C assigns cost $\top$, it means that C forbids the corresponding assignment, otherwise it is permitted by C with the corresponding cost. The *cost* of an assignment $X = (x_1, \dots, x_n)$, noted $\mathcal{V}(X)$, is the sum over all the problem cost functions,

$$\mathcal{V}(X) = \sum_{C_{ij} \in C} C_{ij}(x_i, x_j) \oplus \sum_{C_i \in C} C_i(x_i) \oplus C_\emptyset$$

An assignment X is *consistent* if $\mathcal{V}(X) < \top$. The usual task of interest is to *find a consistent assignment with minimum cost*, which is NP-hard. Observe that WCSP with $k = 1$ reduces to classical CSP.

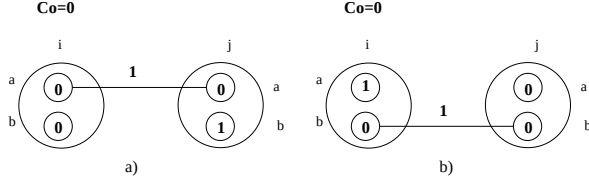


Figure 1: Two equivalent WCSP instances ($T = 2$).

Algorithm 1: Algorithms to propagate costs.

Function PruneVar(i) : boolean

```

 $flag := false$ ;
foreach  $a \in D_i$  do
  if  $(C_\emptyset \oplus C_i(a) \geq T)$  then
     $D_i := D_i - \{a\}$ ;
     $flag := true$ ;
return  $flag$ ;

```

Procedure ProjectUnary(i)

```

 $\alpha := \min_{a \in D_i} \{C_i(a)\}$ ;
 $C_\emptyset := C_\emptyset \oplus \alpha$ ;
foreach  $a \in D_i$  do  $C_i(a) := C_i(a) \ominus \alpha$ ;

```

Procedure Project(i, a, j, α)

```

 $C_i(a) := C_i(a) \oplus \alpha$ ;
foreach  $b \in D_j$  do  $C_{ij}(a, b) := C_{ij}(a, b) \ominus \alpha$ ;

```

Procedure Extend(i, a, j, α)

```

foreach  $b \in D_j$  do  $C_{ij}(a, b) := C_{ij}(a, b) \oplus \alpha$ ;
 $C_i(a) := C_i(a) \ominus \alpha$ ;

```

Example 1 Consider the problem depicted in Figure1.a. It has two variables i, j with two values (a, b) in their domains. Unary costs are depicted within small circles. Binary costs are represented by edges connecting the corresponding values. The label of each edge is the corresponding cost. If two values are not connected, the binary cost between them is 0. In this problem the optimal cost is 0 and it is attained with the assignment (b, a) .

3 Some local consistencies in WCSP

Two WCSPs defined over the same variables are said to be *equivalent* if they define the same cost distribution on complete assignments. Local consistency properties are widely used to transform problems into equivalent simpler ones. In general, constraints can be given explicitly as tables of costs, or implicitly as mathematical expressions or algorithmic procedures. For simplicity in our exposition, we will assume the explicit form. This is done without loss of generality as discussed in [Cooper and Schiex, 2004]. The simplest form of local consistency is *Node consistency* (NC*).

Definition 1 Variable x_i is *node consistent* if: for all values $a \in D_i$, $C_\emptyset \oplus C_i(a) < T$, and there exists a value $a \in D_i$ such that $C_i(a) = \perp$. A WCSP is *node consistent* (NC*) if every variable is node consistent.

Any WCSP can be easily transformed into an equivalent NC* instance by projecting every unary constraints towards $C_\emptyset$ and subsequently pruning every unfeasible value. Functions ProjectUnary and PruneVar (Algorithm1) perform the projection of a unary constraint towards $C_\emptyset$ and prune unfeasible values, respectively.

Arc consistency properties are based on the notion of *simple* and *full support*. Given a binary constraint C_{ij} , we say that $b \in D_j$ is a *simple support* for $a \in D_i$ if $C_{ij}(a, b) = \perp$. Similarly, we say that $b \in D_j$ is a *full support* for $a \in D_i$ if $C_{ij}(a, b) \oplus C_j(b) = \perp$.

Definition 2 Variable x_i is *arc consistent* if every value $a \in D_i$ has a simple support in every constraint C_{ij} . A WCSP is *arc consistent* (AC*) if every variable is node and arc consistent.

Definition 3 Variable x_i is *full arc consistent* if every value $a \in D_i$ has a full support in every constraint C_{ij} . A WCSP is *full arc consistent* (FAC*) if every variable is node and full arc consistent.

Simple supports for $a \in D_i$ in D_j can be enforced by projecting binary costs $C_{ij}(a, \cdot)$ towards $C_i(a)$, as performed by procedure Project (Algorithm 1). Procedure FindSupports (Algorithm 2) enforces simple supports in D_j for every value in D_i .

Example 2 The WCSP depicted in Figure2.a is not AC* because value $a \in D_i$ does not have a simple support in variable j . AC* is enforced by executing FindSupports(i, j) (producing the problem in 2.b) and then ProjectUnary(i) (producing the problem in 2.c).

To enforce full supports for D_i values in D_j (Procedure FindFullSupports in Algorithm 2) we need to extend unary costs from $C_j(\cdot)$ towards $C_{ij}(\cdot, \cdot)$ (procedure Extend in Algorithm 1). Then, binary costs are projected towards $C_i(a)$.

In the CSP case (i.e, $k = 1$), being a simple support for (i, a) is equivalent to being a full support for it and both notions reduce to the classical notion of support. Therefore, AC* and FAC* reduce to classical arc consistency. In the WCSP case, however, being a full support is stronger than being a simple support. In [Larrosa and Schiex, 2004] it was shown that every WCSP can be transformed into an equivalent arc consistent one in time $O(ed^3)$. Unfortunately, not every WCSP can be transformed into an equivalent full arc consistent one.

Example 3 The problem in Figure1.a is AC*, but it is not FAC* because value $a \in D_i$ is not fully supported. We can enforce the full support by calling FindFullSupports(i, j). The resulting problem, depicted in Figure1.b is not FAC* because value $b \in D_j$ is not fully supported. If we enforce the full support by calling FindFullSupports(j, i), we return to the problem in Figure1.a. It can be easily proved that this problem does not have an equivalent FAC* one.

Therefore, FAC* is not a practical property. To circumvent this problem, a weaker property has been proposed. In the sequel, we assume that the set of variables X is totally ordered by $>$.

Definition 4 Variable x_i is *directional arc consistent* if every value $a \in D_i$ has a full support in every constraint C_{ij} such

Algorithm 2: Algorithms to enforce supports.

Function FindSupports(i, j) : boolean

```

 $flag := false$ ;
foreach  $a \in D_i$  do
   $\alpha := \min_{b \in D_j} \{C_{ij}(a, b)\}$ ;
  if  $(\alpha > \perp \wedge C_i(a) = \perp)$  then  $flag := true$ ;
  Project( $i, a, j, \alpha$ );
ProjectUnary( $i$ );
return  $flag$ ;

```

Function FindFullSupports(i, j) : boolean

```

 $flag := false$ ;
foreach  $a \in D_i$  do
   $P[a] := \min_{b \in D_j} \{C_{ij}(a, b) \oplus C_j(b)\}$ ;
  if  $(P[a] > \perp \wedge C_i(a) = \perp)$  then  $flag := true$ ;
foreach  $b \in D_j$  do
   $E[b] := \max_{a \in D_i} \{P[a] \ominus C_{ij}(a, b)\}$ ;
foreach  $b \in D_j$  do Extend( $j, b, i, E[b]$ );
foreach  $a \in D_i$  do Project( $i, a, j, P[a]$ );
ProjectUnary( $i$ );
return  $flag$ ;

```

Function FindExistentialSupport(i) : boolean

```

 $flag := false$ ;
1  $\alpha := \min_{a \in D_i} \{C_i(a) \oplus \bigoplus_{C_{ij} \in C \text{ s.t. } j < i} \min_{b \in D_j} \{C_{ij}(a, b) \oplus C_j(b)\}\}$ ;
if  $(\alpha > \perp)$  then
2 foreach  $C_{ij} \in C$  s.t.  $j < i$  do
   $flag := flag \vee \text{FindFullSupports}(i, j)$ ;
return  $flag$ ;

```

that $j > i$. It is full directional arc consistent (FDAC) if, in addition, every value $a \in D_i$ has a simple support in every constraint C_{ij} such that $j < i$. A WCSP is full directional arc consistent (FDAC*) if every variable is node and full directional arc consistent.

Example 4 The problem in 2.c is AC* but not FDAC*, because value $a \in D_i$ does not have a full support in variable j . FDAC* is enforced by executing FindFullSupports(i, j) (producing the problem in Figure 2.d).

Every WCSP can be transformed into an equivalent FDAC* WCSP with time complexity $O(end^3)$ [Larrosa and Schiex, 2003]. In the CSP case, FDAC* also reduces to classical arc-consistency. In the general case FAC* implies FDAC* and FDAC* implies AC* (namely, the satisfaction of one property implies the satisfaction of the other). An important question is whether it is possible to strengthen the FDAC* property to make it closer to FAC*, but still having the guarantee of existence for an arbitrary WCSP instance. In the next Section we address this issue.

4 Existential arc consistency: a stronger new property

Consider the FDAC* problem in Figure 2.d. Observe that value $a \in D_k$ has a simple support in x_j (required by the FDAC* property) but does not have any full support. We have the same situation with value $b \in D_k$ and variable x_i .

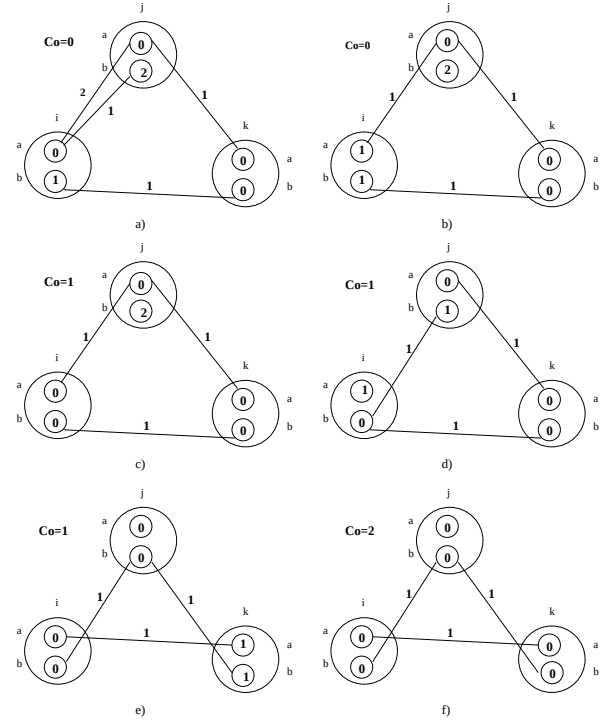


Figure 2: Six equivalent WCSP problems ($T = 4$): (a) original problem (c) made AC*, (d) made FDAC* with $i < j < k$ and (f) made EDAC*.

As a consequence, the unary cost of both values in D_k can be increased by enforcing full supports (Figure 2.e). Since all unary costs of D_i values are larger than zero, node consistency is lost and the lower bound can be increased (Figure 2.f). In general, full supports can be safely enforced in both directions if their enforcement produces an increment in the lower bound. Next, we show that there is a natural local consistency property behind this observation, that we call existential arc consistency (EAC*).

Definition 5 Variable x_i is existential arc consistent if there is at least one value $a \in D_i$ such that $C_i(a) = \perp$ and it has a full support in every constraint C_{ij} . A WCSP is existential arc consistent (EAC*) if every variable is node and existential arc consistent.

All the previous forms of arc consistency required the same condition for every domain value. However, EAC* requires the *existence* for every variable of a special value. The important fact to note is: if a variable x_i is not EAC*, then for all domain value a with $C_i(a) = \perp$ there is a variable x_j such that $\forall b \in D_j, C_{ij}(a, b) \oplus C_j(b) > \perp$. Thus enforcing full supports in x_i will break the NC* property and it will be possible to increase the lower bound. It is possible to integrate EAC* with FDAC* in order to exploit the benefits of each. It is achieved by the existential directional arc consistency (EDAC*) property.

Definition 6 A WCSP is EDAC* if it is FDAC* and EAC*.

In words, EDAC* requires that every value is fully supported in one direction and simply supported in the other direction (to satisfy FDAC*). Additionally, at least one value per variable must be fully supported in both directions (to satisfy EAC*). The special value is called the *fully supported* value. Observe that in the CSP case EDAC* instantiates to classical arc consistency. Besides, EDAC* is weaker than FAC*, but stronger than FDAC*.

5 Algorithm

We present an algorithm for the enforcement of EDAC* in the case of binary WCSPs. EDAC* (Algorithm 3) transforms an arbitrary problem into an *equivalent* one verifying the EDAC* local property. It uses three propagation queues, Q , R and P , that are implemented as priority queues so that the lowest or the highest variable in the queue can be popped in constant time. If $j \in Q$, it means that some value in D_j has been pruned (neighbors of j higher than j may have lost their simple support and must be revised). If $j \in R$, it means that some value in D_j has increased its unary cost from $\perp$ (neighbors of j lower than j may have lost their full support and must be revised). If $i \in P$, it means that some value in D_j ($j < i$ neighbor of i) has increased its unary cost from $\perp$ (i may have lost the full support of its fully supported value and must be revised). Besides, there is an auxiliary queue S that is used to efficiently build P .

The algorithm is formed by a main loop, with four inner loops. The **while** loops at lines 5, 8 and 10 respectively enforce EAC*, DAC* and AC*; the line 11 enforces NC*. Each time some costs are projected by the enforcement of a local property, another property may be broken. The variables for which the local property may be broken are stored in a queue for revision. The enforcement of NC*, AC* and DAC* is achieved as proposed in [Larrosa and Schiex, 2003]. Thus, we focus on the enforcement of EAC*.

EAC* enforcement is mainly encoded by Function `FindExistentialSupport(i)`, which enforces the existential support in x_i by finding full supports wrt. every lower neighbor x_j (it does not have to worry for higher neighbors because DAC* enforcement takes care of them). While enforcing the existential support in x_i , the cost function C_j decreases or remains the same. As a consequence, we do not need to revise the existential consistency of x_j nor the directional arc consistencies of the lowest neighbors of x_j . Furthermore, as the existential consistency is stronger than the arc one, enforcing AC* between x_i and x_j is useless. However, the directional arc consistency of variable x_i should be checked, because some unary cost of x_i could have been increased (line 6). Similarly, the existential consistency of higher neighbors of x_i should also be verified (line 7).

`PruneVar()` enforces node consistency after unary cost and lower bound increments (line 11). This function can be executed more often in order to avoid unnecessary value examinations during the **while** loops. It does not change the worst-case time complexity. For simplicity reasons, the case of inconsistent problems where $C_\emptyset$ reaches $\top$ is not described.

Theorem 1 *The complexity of EDAC* is time $O(ed^2 \max\{nd, \top\})$ and space $O(ed)$.*

Algorithm 3: Enforcing EDAC*, initially, $Q = R = S = x$.

Procedure EDAC*

```

3  while ( $Q \neq \emptyset \vee R \neq \emptyset \vee S \neq \emptyset$ ) do
4     $P := \{l \mid i \in S, l > i, C_{il} \in C\} \cup S$ ;
5     $S := \emptyset$ ;
6    while ( $P \neq \emptyset$ ) do
7       $i := \text{popMin}(P)$ ;
8      if FindExistentialSupport(i) then
9         $R := R \cup \{i\}$ ;
10       foreach  $C_{ij} \in C$  s.t.  $j > i$  do  $P := P \cup \{j\}$ ;
11     while ( $R \neq \emptyset$ ) do
12        $j := \text{popMax}(R)$ ;
13       foreach  $C_{ij} \in C$  s.t.  $i < j$  do
14         if FindFullSupports(i, j) then
15            $R := R \cup \{i\}$ ;
16            $S := S \cup \{i\}$ ;
17     while ( $Q \neq \emptyset$ ) do
18        $j := \text{popMin}(Q)$ ;
19       foreach  $C_{ij} \in C$  s.t.  $i > j$  do
20         if FindSupports(i, j) then
21            $R := R \cup \{i\}$ ;
22            $S := S \cup \{i\}$ ;
23     foreach  $i \in x$  do
24       if PruneVar(i) then  $Q := Q \cup \{i\}$ ;

```

proof 1 *Regarding space, we use the structure suggested by [Cooper and Schiex, 2004] to bring the space complexity to $O(ed)$. [Larrosa and Schiex, 2003] proved that `ProjectUnary`, `Project`, `Extend` and `PruneVar` are time $O(d)$; `FindSupports` and `FindFullSupports` are time $O(d^2)$. Let us now focus on the **while** loop complexities in EDAC*. The loop at line 8 enforces DAC*. As described in previous papers, it is time $O(ed^2)$ because each variable is pushed in R at most once and thus each constraint C_{ij} is checked once (line 9). Similarly, the loop at line 10 which enforces AC*, directed from the lowest variables to the highest ones, is also $O(ed^2)$.*

The loop at line 5 enforces EAC directed to the highest variables. In a single run of this loop, a variable will never be pushed twice in P because the popped variable x_i is always the lowest one, and the pushed variables are higher than x_i . It implies that the **while** loop iterates at most n times, leading to a $O(e)$ time complexity for line 7. The amortized complexity time of `FindExistentialSupport` is $O(ed^2)$ because this function may be called with each variable, so every binary constraint at line 1 and 2 is observed once. Thus, the time complexity of EAC* is also $O(ed^2)$.*

*The line 4 is time $O(n)$. The **for** at line 11 is time $O(nd)$ (which is less than $O(ed)$, as the graph is supposed to be connected). Compiling the different results, the complexity inside the **while** at line 3 is $O(ed^2)$. It loops when:*

- either Q is not empty: the **for** at line 11 has pruned a value and this is done at most nd times;
- either R is not empty: AC* has enqueued a variable in R and by the preceding remark this cannot be done more than $O(nd)$ times;

- or S is not empty: an element has been enqueued while enforcing AC^* or DAC^* . The first case cannot happen more than $O(nd)$. The second case happens $O(\max\{nd, \top\})$ times because the condition at line 8 is true when AC^* has added an element in R ($O(nd)$ times) or EAC^* has added an element ($O(\top)$ times because each time EAC^* is violated, $C_\emptyset$ increases).

Consequently, the overall complexity is $O(ed^2 \max\{nd, \top\})$.

6 Experimental results

In this Section, we perform an empirical comparison of EDAC* with FDAC* for the task of proving optimality. The main use of the EDAC* and FDAC* properties is to use them inside a depth-first branch and bound solver. The idea is to maintain EDAC* and FDAC* during the search (algorithms called MEDAC* and MFDAC*, respectively). We have implemented (C code) this idea in an efficient incremental version that uses tables of supports *à la* AC2001 [Bessiere and Regin, 2001]. In non-binary problems, we delay the propagation of non-binary constraints until they become binary. For variable selection we use the *dom/deg* heuristics which selects the variable with the smallest ratio of domain size divided by future degree. For value selection we consider values in increasing order of unary cost C_i . The variable ordering for directional arc consistency is given by their index. The optimum of each problem instance (except for MAX-SAT where only a good upper bound is provided) is given to the branch and bound as a first upper bound. We experimented with three domains easily modeled as WCSP. Given a CNF formula, **Max-SAT** is the problem of finding a complete assignment with a maximum number of satisfied clauses. We have generated and solved 80-variable 2-SAT and 40-variable 3-SAT instances with varying the number of clauses using *Cn-fgen*¹, a random k -SAT generator. Note that this generator prevents duplicate or opposite literals in clauses but not duplicate clauses. Given an overconstrained CSP, **Max-CSP** is the problem of finding a complete assignment with a maximum number of satisfied constraints. We have generated 6 classes of binary random problems with domain size set to 10 (Sparse-Loose (constraint graph connectivity of 12% for 40 variables), Sparse-Tight, Dense-Loose (constraint graph connectivity of 25%), Dense-Tight, Complete-Loose, and Complete-Tight) as proposed in [Larrosa and Schiex, 2003]. In Max-SAT (resp. Max-CSP), samples have 10 (resp. 50) instances and we report mean values. In the **uncapacitated warehouse location problem** (UWLP) a company considers opening warehouses at some candidate locations in order to supply its existing stores. The objective is to determine which warehouses to open, and which of these warehouses should supply the various stores, such that the sum of the maintenance and supply costs is minimized. Each store must be supplied by exactly one open warehouse. We simply model the problem by l boolean variables for the candidate locations, s integer variables for the stores with domain size set to l , $l + s$ soft unary constraints for the costs, and $l \times s$ hard binary constraints connecting stores and warehouses. We took

and solved the problem instances cap71-134 from the standard OR Library benchmarks for UWLP, as well as the MO*-MP* instances by courtesy of J. Kratica [Kratica *et al.*, 2001]. The M* instances are very challenging for mathematical programming approaches because they have a large number of suboptimal solutions.

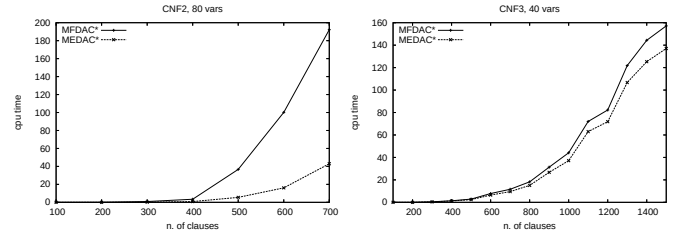


Figure 3: Time in seconds to solve random Max-2SAT and Max-3SAT problems.

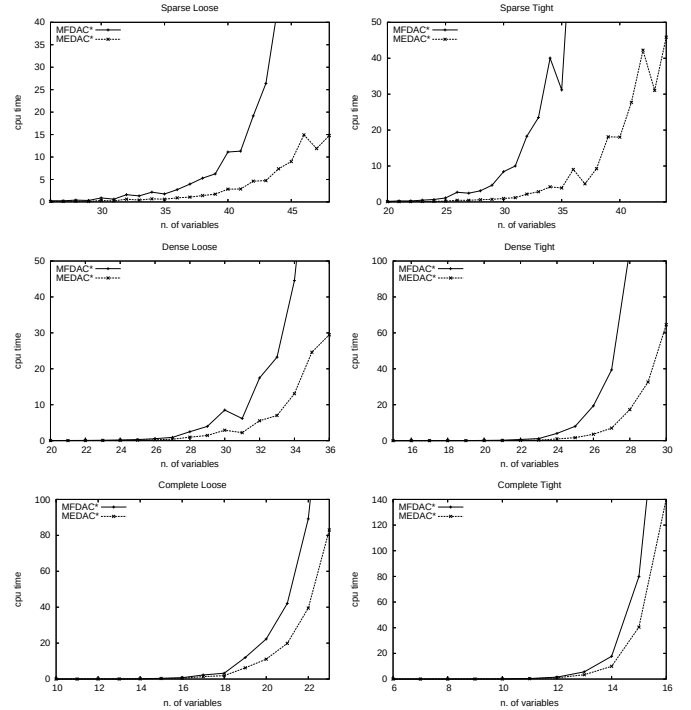


Figure 4: Time in seconds to prove optimality for binary random Max-CSPs.

The experiments were all performed on a 2.0 GHz Pentium 4 computer with 512 MB². Time results in seconds are given in Figures 3 and 4, and Table 5. In all Max-SAT (resp. Max-CSP classes), the search effort seems to grow exponentially with the number of clauses (resp. number of variables). We summed the computation times for all the samples that were completely solved by both algorithms and computed

²The program named TOOLBAR and the benchmarks are available at <http://carlit.toulouse.inra.fr/cgi-bin/awki.cgi/SoftCSP>.

¹A. van Gelder <ftp://dimacs.rutgers.edu/pub/challenge/satisfiability/contributed/UCSC/instances>

the ratio $\text{MFDAC}^*/\text{MEDAC}^*$. In Max-2SAT, MEDAC^* was 5.1 faster than MFDAC^* . The improvement is lower in Max-3SAT. We think this result is due to our simple delaying mechanism for tackling n -ary constraints. For Max-CSP, MEDAC^* was faster than MFDAC^* by a factor ranging from 2.0 (Complete-Tight) to 9.52 (Sparse-Tight). The same results were observed when analyzing the number of nodes expanded by MEDAC^* and MFDAC^* (not reported here for lack of space). In summary, the speed-up obtained was even more important on problems with small constraint graph connectivity.

Problem	Size	MFDAC*	MEDAC*	CPLEX
cap71	16×50	0.01	0.00	0.02
cap72	16×50	0.01	0.01	0.00
cap73	16×50	0.02	0.01	0.00
cap74	16×50	0.02	0.01	0.02
cap101	25×50	0.03	0.02	0.02
cap102	25×50	0.13	0.03	0.01
cap103	25×50	0.35	0.03	0.01
cap104	25×50	0.24	0.04	0.01
cap131	50×50	65.02	0.11	0.05
cap132	50×50	155.08	0.12	0.04
cap133	50×50	194.36	0.14	0.05
cap134	50×50	44.75	0.12	0.05
MO1	100×100	-	216.57	202.48
MO2	100×100	11775.60	44.45	36.67
MO3	100×100	15123.70	74.60	199.84
MO4	100×100	3525.61	31.74	43.26
MO5	100×100	2354.95	33.98	42.96
MP1	200×200	-	3403.37	2296.65
MP2	200×200	-	1443.93	917.92
MP3	200×200	-	981.28	1209.15
MP4	200×200	-	1768.51	1936.57
MP5	200×200	-	1099.88	697.51

Table 5: Time in seconds to prove optimality for uncapacitated warehouse location problems. Problem size is $l \times s$, with l the number of warehouses and s the number of stores. A “-” means the instance was not solved in less than 5 hours.

For UWLP, MEDAC^* is several orders of magnitude faster than MFDAC^* for problem size $l \geq 50$. We concluded that the lower bound computed by EDAC^* for UWLP is stronger and less variable ordering dependent than the FDAC^* one. Although UWLP is well-solved by dedicated mathematical programming approaches [Erlenkotter, 1978; Körkel, 1989] and heuristic search methods [Kratika *et al.*, 2001; Michel and Hentenryck, 2004], it is worth noticing that our generic WCSP algorithm was able to solve to optimality moderately-sized problems in reasonable time (in less than 1 hour). For comparison purposes, we solved UWLP instances using a state-of-the-art mixed-integer programming solver Ilog CPLEX 9.0 with default parameters (and without initial upper bounds) and a direct formulation of the problem. MEDAC^* and CPLEX obtained results within the same order of magnitude in terms of cpu time.

Reversing the order of lines 5 (EAC^* loop), 8 (DAC^* loop), and 10 (AC^* loop) in MEDAC^* slowed down the algorithm

by a factor $\frac{3}{2}$ on our Max-CSP benchmarks. It is important to increase the lower bound as soon as possible as it is done first by EAC^* . We are currently investigating a lazy implementation of EAC^* where the constraints in line 2 are projected only for the purpose of increasing the lower bound and not for pruning the values. This allows to reduce the number of variables inserted in R and P by EAC^* and, consequently, the number of iterations of the different loops.

7 Conclusions and future work

In this paper we have introduced a new local consistency property called EDAC^* and adapted it to WCSP. We have studied its complexity, and despite its theoretical cost, it has been shown that maintaining EDAC^* is always better than maintaining FDAC^* in a variety of domains. In the future, we want to study a lazy version of EDAC^* and to apply it to other problems.

Acknowledgments

The authors are grateful to Thomas Schiex for useful comments on an early version of the paper and to Julia Larrosa for being born after the submission deadline.

References

- [Bessiere and Regin, 2001] C. Bessiere and J.C. Regin. Refining the basic constraint propagation algorithm. In *IJCAI-01*, pages 309–315, 2001.
- [Cooper and Schiex, 2004] M. Cooper and T. Schiex. Arc consistency for soft constraints. *Artificial Intelligence*, 154:199–227, 2004.
- [Erlenkotter, 1978] D. Erlenkotter. A Dual-Based Procedure for Uncapacitated Facility Location. *Operations Research*, 26(6):992–1009, 1978.
- [Körkel, 1989] M. Körkel. On the exact solution of large-scale simple plant location problems. *European Journal of Operational Research*, 39:157–173, 1989.
- [Kratika *et al.*, 2001] J. Kratika, D. Tosic, V. Filipovic, and I. Ljubic. Solving the Simple Plant Location Problems by Genetic Algorithm. *RAIRO Operations Research*, 35:127–142, 2001.
- [Larrosa and Schiex, 2003] J. Larrosa and T. Schiex. In the quest of the best form of local consistency for weighted CSP. In *IJCAI-03*, pages 239–244, 2003.
- [Larrosa and Schiex, 2004] J. Larrosa and T. Schiex. Solving Weighted CSP by Maintaining Arc-consistency. *Artificial Intelligence*, 159(1-2):1–26, 2004.
- [Michel and Hentenryck, 2004] L. Michel and P. Van Hentenryck. A Simple Tabu Search for Warehouse Location. *European Journal on Operations Research*, 157(3):576–591, 2004.
- [Schiex *et al.*, 1995] T. Schiex, H. Fargier, and G. Verfaille. Valued constraint satisfaction problems: hard and easy problems. In *IJCAI-95*, pages 631–637, 1995.

A Uniform Integration of Higher-Order Reasoning and External Evaluations in Answer-Set Programming*

Thomas Eiter, Giovambattista Ianni, Roman Schindlauer, and Hans Tompits

Institut für Informationssysteme, Technische Universität Wien,
Favoritenstraße 9–11, A-1040 Vienna, Austria
{eiter, ianni, roman, tompits}@kr.tuwien.ac.at

Abstract

We introduce HEX programs, which are nonmonotonic logic programs admitting *higher-order atoms* as well as *external atoms*, and we extend the well-known answer-set semantics to this class of programs. Higher-order features are widely acknowledged as useful for performing meta-reasoning, among other tasks. Furthermore, the possibility to exchange knowledge with external sources in a fully declarative framework such as Answer-Set Programming (ASP) is nowadays important, in particular in view of applications in the Semantic Web area. Through external atoms, HEX programs can model some important extensions to ASP, and are a useful KR tool for expressing various applications. Finally, complexity and implementation issues for a preliminary prototype are discussed.

1 Introduction

Answer-Set Programming (ASP) [Gelfond and Lifschitz, 1991] has recently attracted increasing interest as a declarative problem solving paradigm. In this approach, a problem is encoded in terms of a nonmonotonic logic program such that the solutions of the former can be extracted from the *answer sets* of the latter. Due to the availability of efficient answer-set solvers, like Smodels [Simons *et al.*, 2002] or DLV [Leone *et al.*, 2005], and various extensions of the basic language with features such as classical negation, weak constraints, or aggregates, ASP has become an important KR formalism for declaratively solving AI problems in areas including planning, diagnosis, information integration, and reasoning about inheritance. For the challenging area of Semantic Web reasoning, extensions of ASP have been proposed, facilitating interoperability with Description Logic reasoners [Rosati, 1999; Eiter *et al.*, 2004] or aiming at handling infinite, tree-structured models [Heymans and Vermeir, 2003].

*This work was partially supported by the Austrian Science Fund (FWF) under grant P17212-N04, and by the European Commission through the IST Networks of Excellence REVERSE (IST-2003-506779) and CologNeT (IST-2001-33123), and the IST Working Group in Answer Set Programming (IST-2001-37004 WASP).

However, for important issues such as *meta-reasoning* in the context of the Semantic Web, no adequate support is available in ASP to date. Motivated by this fact and the observation that interoperability with other software is (not only in this context) an important issue, we extend in this paper the answer-set semantics to HEX *programs*, that is, *higher-order* logic programs (which accommodate meta-reasoning through *higher-order atoms*) with *external atoms* for software interoperability. Intuitively, a *higher-order atom* allows to quantify values over predicate names, and to freely exchange predicate symbols with constant symbols, like in the rule

$$C(X) \leftarrow \text{subClassOf}(D, C), D(X).$$

An *external atom* facilitates to determine the truth value of an atom through an external source of computation. For instance, the rule

$$\text{reached}(X) \leftarrow \#reach[\text{edge}, a](X)$$

computes the predicate *reached* taking values from the predicate *#reach*, which computes via *#reach[*edge*, *a*]* all the reachable nodes in the graph *edge* from node *a*, delegating this task to an external computational source (e.g., an external deduction system, an execution library, etc.).

Our main contributions are summarized as follows.

(1) We define the syntax and answer-set semantics of HEX programs, extending ASP with higher-order features and powerful interfacing of external computation sources. While answer-set semantics for higher-order logic programs has been proposed earlier by Ross [1994], further extension of that proposal to accommodate external atoms is technically difficult since the approach of Ross is based on the notion of unfounded set, which cannot be easily generalized to this setting. Our approach, instead, is based on a recent notion of program reduct, due to Faber *et al.* [2004], which admits a natural definition of answer-set semantics.

(2) External atoms are a useful abstraction of several extensions to ASP including, among others, aggregates, description logic atoms, or agent programs. External atoms thus facilitate investigating common properties of such extensions, and can serve as a uniform framework for defining semantics of further similar extensions of ASP. Moreover, HEX programs are a basis for the efficient design of generic evaluation algorithms for such extensions in this framework.

(3) By means of HEX programs, powerful meta-reasoning becomes available in a decidable context, e.g., for Semantic Web applications, for meta-interpretation in ASP itself, or for defining policy languages. For example, advanced closed world reasoning or the definition of constructs for an extended ontology language (e.g., of RDF-Schema) is well-supported. Due to the higher-order features, the representation is succinct.

(4) A simple prototype implementation of the language is available, based on a reduction to ordinary ASP.

Note that other logic-based formalisms, like TRIPLE [Sintek and Decker, 2002] or F-Logic [Kifer *et al.*, 1995], feature also higher-order predicates for meta-reasoning in Semantic Web applications. However, TRIPLE is low-level oriented and lack precise semantics, while F-Logic in its implementations (Flora, Florid, Ontoweb) restricts its expressiveness to well-founded semantics for negation, in order to gain efficiency. Our formalism, instead, is fully declarative and offers the possibility of nondeterministic predicate definition with higher complexity. This proved already useful and reasonably efficient for a range of applications with inherent non-determinism, such as diagnosis, planning, or configuration, and thus provides a rich basis for integrating these areas with meta-reasoning.

2 HEX Programs

2.1 Syntax

Let $\mathcal{C}$, $\mathcal{X}$, and $\mathcal{G}$ be mutually disjoint sets whose elements are called *constant names*, *variable names*, and *external predicate names*, respectively. Unless explicitly specified, elements from $\mathcal{X}$ (resp., $\mathcal{C}$) are denoted with first letter in upper case (resp., lower case), while elements from $\mathcal{G}$ are prefixed with “#”. We note that constant names serve both as individual and predicate names.

Elements from $\mathcal{C} \cup \mathcal{X}$ are called *terms*. A *higher-order atom* (or *atom*) is a tuple $(Y_0, Y_1, \dots, Y_n)$, where $Y_0, \dots, Y_n$ are terms; $n \geq 0$ is the *arity* of the atom. Intuitively, Y_0 is the predicate name, and we thus also use the more familiar notation $Y_0(Y_1, \dots, Y_n)$. The atom is *ordinary*, if Y_0 is a constant.

For example, $(x, \text{rdf:type}, c)$, $\text{node}(X)$, and $D(a, b)$, are atoms; the first two are ordinary atoms.

An *external atom* is of the form

$$\#g[Y_1, \dots, Y_n](X_1, \dots, X_m), \quad (1)$$

where $Y_1, \dots, Y_n$ and $X_1, \dots, X_m$ are two lists of terms (called *input* and *output* lists, respectively), and $\#g \in \mathcal{G}$ is an external predicate name. We assume that $\#g$ has fixed lengths $\text{in}(\#g) = n$ and $\text{out}(\#g) = m$ for input and output lists, respectively. Intuitively, an external atom provides a way for deciding the truth value of an output tuple depending on the extension of a set of input predicates.

Example 1 The external atom $\#reach[edge, a](X)$ may be devised for computing the nodes which are reachable in the graph *edge* from the node *a*. Here, we have that $\text{in}(\#reach) = 2$ and $\text{out}(\#reach) = 1$. $\square$

A rule *r* is of the form

$$\alpha_1 \vee \dots \vee \alpha_k \leftarrow \beta_1, \dots, \beta_n, \text{not}\beta_{n+1}, \dots, \text{not}\beta_m, \quad (2)$$

where $m, k \geq 0$, $\alpha_1, \dots, \alpha_k$ are atoms, and $\beta_1, \dots, \beta_m$ are either atoms or external atoms. We define $H(r) = \{\alpha_1, \dots, \alpha_k\}$ and $B(r) = B^+(r) \cup B^-(r)$, where $B^+(r) = \{\beta_1, \dots, \beta_n\}$ and $B^-(r) = \{\beta_{n+1}, \dots, \beta_m\}$. If $H(r) = \emptyset$ and $B(r) \neq \emptyset$, then *r* is a *constraint*, and if $B(r) = \emptyset$ and $H(r) \neq \emptyset$, then *r* is a *fact*; *r* is *ordinary*, if it contains only ordinary atoms.

A HEX program is a finite set *P* of rules. It is *ordinary*, if all rules are ordinary.

2.2 Semantics

We define the semantics of HEX programs by generalizing the answer-set semantics [Gelfond and Lifschitz, 1991]. To this end, we use the recent notion of a reduct as defined by Faber *et al.* [2004] (referred to as *FLP-reduct* henceforth) instead of to the traditional reduct by Gelfond and Lifschitz [1991]. The FLP-reduct admits an elegant and natural definition of answer sets for programs with aggregate atoms, since it ensures answer-set minimality, while the definition based on the traditional reduct lacks this important feature.

In the sequel, let *P* be a HEX program. The *Herbrand base* of *P*, denoted HB_P , is the set of all possible ground versions of atoms and external atoms occurring in *P* obtained by replacing variables with constants from $\mathcal{C}$. The grounding of a rule *r*, $\text{grnd}(r)$, is defined accordingly, and the grounding of program *P* is given by $\text{grnd}(P) = \bigcup_{r \in P} \text{grnd}(r)$. Unless specified otherwise, $\mathcal{C}$, $\mathcal{X}$, and $\mathcal{G}$ are implicitly given by *P*.

Example 2 Given $\mathcal{C} = \{\text{edge}, \text{arc}, a, b\}$, ground instances of $E(X, b)$ are $\text{edge}(a, b)$, $\text{arc}(a, b)$, and $\text{arc}(\text{arc}, b)$; ground instances of $\#reach[\text{edge}, N](X)$ are $\#reach[\text{edge}, \text{edge}](a)$, $\#reach[\text{edge}, \text{arc}](b)$, and $\#reach[\text{edge}, \text{edge}](\text{edge})$, etc. $\square$

An *interpretation relative to P* is any subset $I \subseteq HB_P$ containing only atoms. We say that *I* is a *model* of atom $a \in HB_P$, denoted $I \models a$, if $a \in I$.

With every external predicate name $\#g \in \mathcal{G}$, we associate an $(n+m+1)$ -ary Boolean function $f_{\#g}$ assigning each tuple $(I, y_1, \dots, y_n, x_1, \dots, x_m)$ either 0 or 1, where $n = \text{in}(\#g)$, $m = \text{out}(\#g)$, $I \subseteq HB_P$, and $x_i, y_j \in \mathcal{C}$.

We say that $I \subseteq HB_P$ is a *model* of a ground external atom $a = \#g[y_1, \dots, y_n](x_1, \dots, x_m)$, denoted $I \models a$, if and only if $f_{\#g}(I, y_1, \dots, y_n, x_1, \dots, x_m) = 1$.

Example 3 Let us associate with $\#reach$ a function $f_{\#reach}$ such that $f_{\#reach}(I, E, A, B) = 1$ iff *B* is reachable in the graph *E* from *A*. Let $I = \{e(b, c), e(c, d)\}$. Then, *I* is a model of $\#reach[e, b](d)$ since $f_{\#reach}(I, e, b, d) = 1$. $\square$

Let *r* be a ground rule. We define (i) $I \models H(r)$ iff there is some $a \in H(r)$ such that $I \models a$, (ii) $I \models B(r)$ iff $I \models a$ for all $a \in B^+(r)$ and $I \not\models a$ for all $a \in B^-(r)$, and (iii) $I \models r$ iff $I \models H(r)$ whenever $I \models B(r)$. We say that *I* is a *model* of a HEX program *P*, denoted $I \models P$, iff $I \models r$ for all $r \in \text{grnd}(P)$. We call *P* *satisfiable*, if it has some model.

Given a HEX program *P*, the *FLP-reduct* of *P* with respect to $I \subseteq HB_P$, denoted fP^I , is the set of all $r \in \text{grnd}(P)$ such

that $I \models B(r)$. $I \subseteq HB_P$ is an *answer set* of P iff I is a minimal model of fP^I .

We next give an illustrative example.

Example 4 Consider the following HEX program P :

```
subRelation(brotherOf, relativeOf) ← ;
brotherOf(john, al) ← ;
relativeOf(john, joe) ← ;
brotherOf(al, mick) ← ;
invites(john, X) ∨ skip(X) ← X <> john,
                             #reach[relativeOf, john](X);
R(X, Y) ← subRelation(P, R), P(X, Y);
← #degs[invites](Min, Max), Min < 1;
← #degs[invites](Min, Max), Max > 2.
```

Informally, this program randomly selects a certain number of John's relatives for invitation. The first line states that *brotherOf* is a subrelation of *relativeOf*, and the next two lines give concrete facts. The disjunctive rule chooses relatives, employing the external predicate *#reach* from Example 3. The next rule declares a generic subrelation inclusion exploiting higher-order atoms.

The constraints ensure that the number of invitees is between 1 and 2, using (for illustration) an external predicate *#degs* from a graph library, where $f_{\#degs}(I, E, Min, Max)$ is 1 iff Min and Max is the minimum and maximum vertex degree of the graph induced by the edges E , respectively. As John's relatives are determined to be Al, Joe, and Mick, P has six answer sets, each of which contains one or two of the facts *invites(john, al)*, *invites(john, joe)*, and *invites(john, mick)*. $\square$

We now state some basic properties of the semantics.

Theorem 1 *The answer-set semantics of HEX programs extends the answer-set semantics of ordinary programs as defined by Gelfond and Lifschitz [1991], as well as the answer-set semantics of HiLog programs as defined by Ross [1994].*

The next property, which is easily proved, expresses that answer sets adhere to the principle of minimality.

Theorem 2 *Every answer set of a HEX program P is a minimal model of P .*

A ground external atom a is called *monotonic relative to P* iff $I \subseteq I' \subseteq HB_P$ and $I \models a$ imply $I' \models a$. For instance, the ground versions of *#reach[edge, a](X)* are all monotonic.

Theorem 3 *Let P be a HEX program without “not” and constraints. If all external atoms in $grnd(P)$ are monotonic relative to P , then P has some answer set. Moreover, if P is disjunction-free, it has a single answer set.*

Notice that this property fails if external atoms can be non-monotonic. Indeed, we can easily model default negation *not p(a)* by an external atom *#not[p](a)*; the HEX program $p(a) \leftarrow \#not[p](a)$ amounts then to the ordinary program $p(a) \leftarrow not\ p(a)$, which has no answer set.

3 Modeling ASP Extensions by External Atoms

By means of external atoms, different important extensions of ASP can be expressed in terms of HEX programs.

3.1 Programs with aggregates

Extending ASP with special *aggregate atoms*, through which the sum, maximum, etc. of a set of numbers can be referenced, is an important issue which has been considered in several recent works (cf., e.g., [Faber et al., 2004]). A non-trivial and challenging problem in this context is giving a natural semantics for aggregates involving recursion. The recent proposal of a semantics by Faber et al. [2004] is an elegant solution of this problem. We show here how it can be easily captured by HEX programs.

An aggregate atom $a(Y, T)$ has the form $f\{S\} \prec T$, where f is an aggregate function (*sum*, *count*, *max*, etc.), $\prec \in \{=, <, \leq, >, \geq\}$, T is a term, and S is an expression $X:\vec{E}(\vec{X}, \vec{Y}, \vec{Z})$, where $\vec{X}$ and $\vec{Y}$ are lists of *local variables*, $\vec{Z}$ is a list of *global variables*, and $\vec{E}$ is a list of atoms whose variables are among $\vec{X}, \vec{Y}, \vec{Z}$.

For example, $\#count\{X : r(X, Z), s(Z, Y)\} \geq T$ is an aggregate atom which is intuitively true if, for given Y and T , at least T different values for X are such that the conjunction $r(X, Z), s(Z, Y)$ holds.

Given $a(Y, T) = f\{S\} \prec T$ as above, an interpretation I , and values y for Y and t for T , f is applied to the set $S(I, y)$ of all values x for X such that $I \models E(x, y, z)$ for some value z for Z . We then have $I \models a(y, t)$ (i.e., $I \models f\{X:E(X, y, Z)\} \prec t$) iff $f(S(I, y)) \prec t$.

Using the above notion of truthhood for $a(y, t)$, Faber et al. [2004] define answer sets of an ordinary program plus aggregates using the reduct fP^I .

We can model an aggregate atom $a(Y, T)$ by an external atom $\#a[Y](T)$ such that for any interpretation I and ground version $\#a[y](t)$ of it, $f_{\#a}(I, y, t) = 1$ iff $I \models a(y, t)$. Note that writing code for evaluating $f_{\#a}(I, y, t)$ is easy.

For any ordinary program P with aggregates, let $\#agg(P)$ be the HEX program which results from P by replacing each aggregate atom $a(Y, T)$ with the respective external atom $\#a[Y](T)$. The following result can then be shown:

Theorem 4 *For any ordinary program P with aggregates, the answer sets of P and $\#agg(P)$ coincide.*

3.2 Description logic programs

The aim of *description logic programs* (or *dl-programs*), due to Eiter et al. [2004], is to combine a rule language under the answer-set semantics with description logics. Informally, a dl-program consists of a description logic (DL) knowledge base L and a generalized normal program P which may contain queries to L , realized by means of special atoms, called *dl-atoms*, appearing in the body of rules. A dl-atom allows for specifying an input from P to L , and thus for a bidirectional flow of information between P and L , and for querying whether a certain DL axiom or its negation logically follows from L . The DL knowledge bases in dl-programs are theories in the description logics *SHIF(D)* and *SHOIN(D)*,

which represent the logical underpinnings of the Web ontology languages OWL Lite and OWL DL, respectively [Bechhofer *et al.*, 2004].

Formally, a *dl-atom* is an expression $dl(X)$ of form

$$DL[S_1 \text{ op}_1 p_1, \dots, S_m \text{ op}_m p_m; Q](X), \quad m \geq 0,$$

where each S_i is a DL concept or role name, op_i a change operator, p_i a unary resp. binary predicate symbol, Q a unary resp. binary predicate, and X a list of terms matching the arity of Q . For space reasons, we confine here to $\text{op}_i = \uplus$ and Q being a possibly negated unary predicate name, for which X is a single term. Intuitively, $S_i \uplus p_i$ increases S_i in L by the extension of p_i . For example, the dl-atom

$$DL[\text{hasColor} \uplus \text{color}; \text{whiteWine}](W)$$

queries a wine ontology if W is known to be a white wine, after augmenting the ontology about wine color (*hasColor*) with facts about *color* from a program P .

An interpretation I of P is a *model* of a ground instance $dl(c)$ of dl-atom $dl(X)$ with respect to DL knowledge base L , denoted $I \models_L dl(c)$, if $L \cup \bigcup_{i=1}^m \{S_i(b) \mid p_i(b) \in I\} \models Q(c)$, where $\models$ is the entailment operator of the given description logic. That is, $I \models_L dl(c)$ iff c belongs to concept Q after augmenting L .

Eiter *et al.* [2004] define answer sets of an ordinary non-disjunctive program P relative to a DL knowledge base L through a reduct sP_L^I , which extends the traditional reduct of Gelfond and Lifschitz [1991]. Assuming that each ground dl-atom $dl(c)$ is monotonic (i.e., $I \models dl(c)$ implies $I' \models dl(c)$, for $I \subseteq I'$; this is the predominant setting), sP_L^I treats negated dl-atoms like negated ordinary atoms. The resulting ground program sP_L^I has a least model, $LM(sP_L^I)$. Then, I is a *strong answer set* of (L, P) iff $I = LM(sP_L^I)$ holds.

We can simulate dl-atoms by external atoms in several ways. A simple one is to use external atoms $\#dl[](X)$ where $f_{\#dl}(I, c) = 1$ iff $I \models_L dl(c)$. Let $\#dl_L(P)$ be the HEX program obtained from a dl-program (L, P) by replacing each dl-atom $dl(X)$ with $\#dl[](X)$. We can then show:

Theorem 5 *Let (L, P) be any dl-program for which all ground dl-atoms are monotonic. Then, the strong answer sets of (L, P) and $\#dl_L(P)$ coincide.*

Note that we can extend the strong answer-set semantics to disjunctive dl-programs by simply extending the embedding $\#dl_L(P)$ to disjunctive programs. This illustrates the use of HEX programs as a framework for defining semantics.

3.3 Programs with monotone cardinality atoms

Marek *et al.* [2004] present an extension of ASP by *monotone cardinality atoms* (*mc-atoms*) kX , where X is a finite set of ground atoms and $k \geq 0$. Such an atom is true in an interpretation I , if $k \geq |X \cap I|$ holds. Note that an ordinary atom A amounts to $1\{A\}$. An *mca-program* is a set of rules

$$H \leftarrow B_1, \dots, B_m, \text{not } B_{m+1}, \dots, \text{not } B_n \quad (3)$$

where H and the B_i 's are mc-atoms. Answer sets (stable models) for an mca-program P are interpretations I which

are *derivable* models of an extended reduct P^I (in the sense of Gelfond and Lifschitz [1991]), which treats negated mc-atoms like negated ordinary atoms. Informally, a model of P^I is derivable, if it can be created from the empty set by iterative rule applications in which the heads of firing rules are nondeterministically satisfied.

We can embed any mca-program P into a HEX program $\#mc(P)$ as follows. Each mc-atom kX is modeled by an external atom $e(kX) = \#k_X[]()$, where $f_{\#k_X}(I) = 1$ iff $k \geq |X \cap I|$. In each rule of form (3), we replace H with a new atom t_H and all B_i with $e(B_i)$, and add the following rules (for $H = k\{A_1, \dots, A_m\}$):

$$\begin{aligned} A_i \vee n_A_i &\leftarrow t_H, & 1 \leq i \leq m, \\ &\leftarrow \text{not } e(H), t_H, \end{aligned}$$

where, globally, n_A is a new atom for each atom A . Informally, these rules simulate the occurrence of the mc-atom in the head. Then, the following correspondence holds.

Theorem 6 *For any finite mca-program P over atoms At , the answer sets of P and $\#mc(P)$ projected to At coincide.*

As shown by Marek *et al.* [2004], ASP extensions similar to mca-programs can be modeled as mca-programs. Hence, these extensions can be similarly embedded into HEX programs.

3.4 Agent programs

Eiter *et al.* [1999] describe logic-based *agent programs*, consisting of rules of the form

$$Op_0 \alpha_0 \leftarrow \chi, [\neg] Op_1 \alpha_1, \dots, [\neg] Op_m \alpha_m,$$

governing an agent's behavior. The Op_i are *deontic modalities*, the α_i are *action atoms*, and χ is a *code-call condition*. The latter is a conjunction of (i) *code-call atoms* of the form $\text{in}(X, f(Y))$ resp. $\text{notin}(X, f(Y))$, which access the data structures of the internal agent state through API functions $f(Y)$ and test whether X is in the result, and (ii) *constraint atoms*. For example, the rule

$$\text{Do dial}(N) \leftarrow \text{in}(N, \text{phone}(P)), O \text{ call}(P)$$

intuitively says that the agent should dial phone number N if she is obliged to call P .

A semantics of agent programs in terms of “reasonable status sets”, which are certain sets of ground formulas $Op \alpha$, is defined by Eiter *et al.* [1999]. They show that the answer sets of a disjunction-free logic program P correspond naturally to the reasonable status sets of a straightforward agent program $AG(P)$. Conversely, code-call atoms as above can be modeled by external atoms $\#in_f[Y](X)$ resp. $\#notin_f[Y](X)$, and deontic modalities by different propositions and suitable rules. In this way, a class of agent programs can be embedded into HEX programs as a host for evaluation.

4 Applications

In this section, we show the usage of HEX programs for different purposes, in which the joint availability of higher-order and external atoms is beneficial. For space reasons, the exposition is necessarily superficial and details will be omitted.

4.1 Semantic Web applications

HEX programs are well-suited as a convenient tool for a variety of tasks related to ontology languages and for Semantic-Web applications in general, since, in contrast to other approaches, they keep decidability but do not lack the possibility of exploiting nondeterminism, performing meta-reasoning, or encoding aggregates and sophisticated constructs through external atoms.

An interesting application scenario where several features of HEX programs come into play is *ontology alignment*. Merging knowledge from different sources in the context of the Semantic Web is a very important task [Calvanese *et al.*, 2001]. To avoid inconsistencies which arise in merging, it is important to diagnose the source of such inconsistencies and to propose a “repaired” version of the merged ontology. In general, given an entailment operator $\models$ and two theories T_1 and T_2 , we want to find some theory $rep(T_1 \cup T_2)$ which, if possible, is consistent (with respect to $\models$). Usually, rep is defined according to some customized criterion, so that to save as much knowledge as possible from T_1 and T_2 . Also, rep can be nondeterministic and admit more than one possible solution.

HEX programs allow to define $\models$ according to a range of possibilities; in the same way, HEX programs are a useful tool for modeling and customizing the rep operator. In order to perform ontology alignment, HEX programs must be able to express tasks such as the following ones:

Importing external theories. This can be achieved, e.g., in the following way:

$$\begin{aligned} triple(X, Y, Z) &\leftarrow \#RDF[uri](X, Y, Z); \\ triple(X, Y, Z) &\leftarrow \#RDF[uri2](X, Y, Z); \\ proposition(P) &\leftarrow triple(P, rdf:type, \\ &\quad rdf:Statement). \end{aligned}$$

We assume here to deal with RDF theories.¹ We take advantage of an external predicate $\#RDF$ intended to extract knowledge from a given URI (Uniform Resource Identifier), in form of a set of “reified” ternary assertions.

Searching in the space of assertions. This task is required in order to choose nondeterministically which propositions have to be included in the merged theory and which not, with statements like

$$pick(P) \vee drop(P) \leftarrow proposition(P).$$

Translating and manipulating reified assertions. E.g., for choosing how to put RDF triples (possibly including OWL assertions) in an easier manipulatable and readable format, and for making selected propositions true, the following rules can be employed:

$$\begin{aligned} (X, Y, Z) &\leftarrow pick(P), triple(P, rdf:subject, X), \\ &\quad triple(P, rdf:predicate, Y), \\ &\quad triple(P, rdf:object, Z); \\ C(X) &\leftarrow (X, rdf:type, C). \end{aligned}$$

¹See <http://www.w3.org/tr/rdf-mt/> for information about RDF.

Filtering propositions. This way, it is possible to customize criteria for selecting which propositions can be dropped and which cannot. For instance, a proposition cannot be dropped if it is an RDFS axiomatic triple:²

$$pick(P) \leftarrow axiomatic(P).$$

Defining ontology semantics. The operator $\models$ can be defined in terms of entailment rules and constraints expressed in the language itself, like in:

$$\begin{aligned} D(X) &\leftarrow (C, rdf:subClassOf, D), C(X); \\ &\leftarrow owl:maxCardinality(C, R, N), C(X), \\ &\quad \#count[R, X](M), M > N, \end{aligned}$$

where the external atom $\#count[R, X](M)$ expresses the aggregate atom $\#count\{Y : R(X, Y)\} = M$. Also, semantics can be defined by means of external reasoners, using constraints like

$$\leftarrow \#inconsistent[pick],$$

where the external predicate $\#inconsistent$ takes for input a set of assertions and establishes through an external reasoner whether the underlying theory is inconsistent.

4.2 Closed world and default reasoning

Reiter’s well-known closed-world assumption (CWA)³ is acknowledged as an important reasoning principle for inferring negative information from a logical knowledge base KB : For a ground atom $p(c)$, conclude $\neg p(c)$ if $KB \not\models p(c)$. Description logic knowledge bases lack this possibility.

Using HEX programs, the CWA may be easily expressed on top of an external KB which can be queried through suitable external atoms. We show this here for a description logic knowledge base L . Assuming that a generic external atom $\#dl_0[C](X)$ for modeling a dl-atom $DL[C](X)$ is available, the CWA principle can be stated as follows:

$$\begin{aligned} C'(X) &\leftarrow not \#dl_0[C](X), concept(C), \\ &\quad cwa(C, C'), o(X), \end{aligned}$$

where $concept(C)$ is a predicate which holds for all concepts, $cwa(C, C')$ states that C' is the complement of C under the CWA, and $o(X)$ is a predicate that holds for all individuals occurring in L . For example, given that

$$L = \{ man \sqsubseteq person, person(lee) \}$$

for concepts man and $person$, the CWA infers $\neg man(lee)$.

As well known, the CWA can become inconsistent. If in the above example, L contains a further axiom

$$person = man \sqcup woman,$$

with the concept $woman$, then the CWA infers $\neg man(lee)$ and $\neg woman(lee)$; this is inconsistent with L .

²In a language enriched with *weak constraints*, we could maximize the set of selected propositions using a constraint of form $:\sim drop(P)$.

³Throughout this section, we refer to Łukasiewicz [1990] for references to closed-world reasoning and circumscription.

We can check inconsistency of the CWA with further rules, though:

$$\begin{aligned} \text{set_false}(C, X) &\leftarrow \text{cwa}(C, C'), C'(X), \\ \text{inconsistent} &\leftarrow \#dl_1[\text{set_false}, \perp](b), \end{aligned}$$

where $\#dl_1[N, C](X)$ effects a check whether L , augmented with all negated facts $\neg c(a)$ such that $N(c, a)$ holds, entails $C(X)$, and $\perp$ is the empty concept (entailment of $\perp(b)$, for any constant b , is tantamount to inconsistency).

Minimal-model reasoning, as under circumscription and the extended closed-world assumption (ECWA), for instance, avoids the problem of CWA inconsistency. We can foster the minimal Herbrand models of L with respect to all concepts and individuals in L elegantly with the following HEX rules:

$$\begin{aligned} \text{set_false}(C, X) &\leftarrow \text{concept}(C), o(X), \text{not } C(X); \\ C(X) &\leftarrow \#dl_1[\text{set_false}, C](X). \end{aligned}$$

Here, the first rule intuitively expresses that if $C(X)$ is not included in an answer set M of P , then it should be set to false. The second rule states that $C(X)$ is in M , if $C(X)$ can be proved in L after setting all atoms in L to false according to M . By the minimality of answer sets, $C(X)$ can only then be in M . Thus, in L no $C(X)$ can be switched to $\neg C(X)$ without raising inconsistency. Hence, M corresponds to a minimal model of L . Applied to our example, we obtain two answer sets (showing here only the interesting atoms):

$$\begin{aligned} M_1 &= \{\text{person}(\text{lee}), \text{woman}(\text{lee}), \\ &\quad \text{set_false}(\text{man}, \text{lee}), \dots\}, \\ M_2 &= \{\text{person}(\text{lee}), \text{man}(\text{lee}), \\ &\quad \text{set_false}(\text{woman}, \text{lee}), \dots\}, \end{aligned}$$

corresponding to the minimal models of L .

Roles in L may be handled similarly. Furthermore, one can easily restrict minimization to a subset of concepts and roles, and accommodate the general setting of ECWA and circumscription, dividing the predicates into minimized, fixed, and varying predicates P , Q , and Z , respectively. On top of minimal models, e.g., reasoning tasks may then be performed.

By *maximizing* rather than minimizing extensions, default reasoning, as in the approach by Poole [1988], on top of a DL knowledge base L may be supported. For example, the rules

$$\begin{aligned} \text{white}(W) &\leftarrow \#dl_1[\text{null}, \text{sparklingWine}](W), \\ &\quad \text{not } n_white(W), \\ n_white(W) &\leftarrow \#dl_2[\text{sparklingWine}, \text{white}, \\ &\quad \text{whiteWine}](W) \end{aligned}$$

on top of a wine ontology L , express that sparkling wines are white by default, where $\#dl_2[C, U, Q](X)$ checks whether L , together with all facts $C(a)$ such that $a \in U$, entails $\neg Q(X)$. Given

$$L = \{\text{sparklingWine}(\text{veuveCliquot}), \\ \text{lambrusco} \sqsubseteq (\text{sparklingWine} \sqcap \neg \text{whiteWine})\},$$

we then can conclude $\text{white}(\text{veuveCliquot})$.

5 Computational Aspects

5.1 Complexity

It appears that higher-order atoms do not add complexity compared to ordinary atoms. Indeed, for finite $\mathcal{C}$, the grounding of an arbitrary HEX program P is, like for an ordinary program, at most exponential in the size of P and $\mathcal{C}$. Since HEX programs with higher-order atoms subsume ordinary programs, we obtain by well-known complexity results for ordinary programs [Dantsin *et al.*, 2001] the following result. Recall that $NEXP$ denotes nondeterministic exponential time, and that for complexity classes C and D , C^D denotes complexity in C with an oracle for a problem in D .

Theorem 7 *Deciding whether a given HEX program P without external atoms has some answer set is $NEXP^{NP}$ -complete in general, and $NEXP$ -complete if P is disjunction-free.*

Classes of programs with lower complexity can be identified under syntactic restrictions, e.g., on predicate arities. Furthermore, if from the customary ASP perspective, P is fixed except for ground facts representing ad-hoc input, the complexity exponentially drops to NP^{NP} resp. NP .

On the other hand, external atoms clearly may be a source of complexity, and without further assumptions even incur undecidability. Viewing the function $f_{\#g}$ associated with an external predicate $\#g \in \mathcal{G}$ as an oracle with complexity in C , we have the following result:

Theorem 8 *Let P be a HEX program, and suppose that for every $\#g \in \mathcal{G}$ the function $f_{\#g}$ has complexity in C . Then, deciding whether P has some answer set is in $NEXP^{NP^C}$, and is in $NEXP^C$ if P is disjunction-free.*

However, there is no complexity increase by external atoms under the following condition on the cardinality of $\mathcal{C}$:

Theorem 9 *Let P be a HEX program. Suppose that for every $\#g \in \mathcal{G}$, the function $f_{\#g}$ is decidable in exponential time in $|\mathcal{C}|$. Then, deciding whether P has some answer set is $NEXP^{NP}$ -complete, and $NEXP$ -complete if P is disjunction-free.*

Informally, the reason is that a possibly exponential-size grounding compensates the exponentiality of external atoms, whose evaluation then becomes polynomial in the size of $\text{grnd}(P)$. The hypothesis of Theorem 9 applies to external atoms modeling aggregate atoms and, under small adjustments, to dl-atoms, if $\models$ is decidable in exponential time. Some complexity results by Faber *et al.* [2004] on ASP with aggregates and by Eiter *et al.* [2004] on interfacing logic programs with the description logic $\mathcal{SHIF}(\mathbf{D})$ therefore follow easily from Theorems 4, 5, and 9.

5.2 Implementation

An experimental working prototype for evaluating HEX programs is available. Several technical issues in an implementation arise, and we can only briefly address them here. In particular, higher-order and external atoms must be handled.

As for higher-order atoms, a polynomial reduction Λ from HEX programs P to ordinary programs $\Lambda(P)$ is possible if P has no external atoms. Indeed, each higher-order atom $Y_0(Y_1, \dots, Y_n)$ in P can be substituted with an ordinary atom $a_n(Y_0, Y_1, \dots, Y_n)$. Since HEX programs conservatively extend ordinary programs (cf. Theorem 1), the answer sets of any HEX program P without external atoms then correspond one-to-one with the answer sets of $\Lambda(P)$. Thus, HEX programs without external atoms can be efficiently evaluated by using an existing ASP solver.

The presence of external atoms makes matters more complex. Λ can still be applied to eliminate higher-order atoms from a HEX program P , and a similar correspondence holds. We may further replace external atoms $\#g[\vec{X}](\vec{Y})$ in $\Lambda(P)$ by ordinary atoms $p_{\#g}(\vec{X}, \vec{Y})$. In the absence of negation as failure and for monotone external atoms, the answer sets of $\Lambda(P)$ can be computed by a bottom-up fixpoint computation (which in case of disjunction is nondeterministic), in which ground atoms $p_{\#g}(\vec{a}, \vec{b})$ are evaluated with the external function $f_{\#g}$.

In the presence of negation as failure, a notion of *e-stratification*, which generalizes the usual notion of stratification and exploits further dependency information supplied for external atoms, can be used to identify a substantial fragment of HEX programs evaluable on the basis of a suitable operational semantics. In the unstratified case, guessing clauses

$$p_{\#g}(\vec{X}, \vec{Y}) \vee \text{not-}p_{\#g}(\vec{X}, \vec{Y}) \leftarrow$$

may be added for generating candidate answer sets of P . For monotone external atoms, the candidates can be verified by a fixpoint computation. For the general case, however, efficient checking methods are needed.

6 Conclusion and Further Work

HEX programs are a natural and powerful evolution of Answer-Set Programming (ASP), which fulfills interoperability needs with other software and supports at the same time abstract problem modeling by higher-order features. These features are needed for a wide range of applications but missing in ASP systems today. In particular, user-defined libraries can be integrated, and thus customization to specific applications is enabled. Our further and ongoing work includes implementation beyond the working prototype, for which suitable algorithms and techniques are currently under development. This and the prototype will be discussed in detail elsewhere. Furthermore, an application in the context of an ongoing project for a personalized Web information system is targeted.

References

- [Bechhofer *et al.*, 2004] S. Bechhofer, F. van Harmelen, J. Hendler, et al. OWL web ontology language reference. <http://www.w3.org/tr/owl-ref/>.
- [Calvanese *et al.*, 2001] D. Calvanese, G. De Giacomo, and M. Lenzerini. A framework for ontology integration. In *Proc. SWWS-2001*, pp. 303–316, 2001.
- [Dantsin *et al.*, 2001] E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity and expressive power of logic programming. *ACM Comp. Surveys*, 33:374–425, 2001.
- [Eiter *et al.*, 1999] T. Eiter, V.S. Subrahmanian, and G. Pick. Heterogeneous active agents, I: Semantics. *Artificial Intelligence*, 108(1-2):179–255, 1999.
- [Eiter *et al.*, 2004] T. Eiter, T. Lukasiewicz, R. Schindlauer, and H. Tompits. Combining Answer Set Programming with description logics for the Semantic Web. In *Proc. KR-2004*, pp. 141–151, 2004.
- [Faber *et al.*, 2004] W. Faber, N. Leone, and G. Pfeifer. Recursive aggregates in disjunctive logic programs: Semantics and complexity. In *Proc. JELIA-2004*, pp. 200–212, 2004.
- [Gelfond and Lifschitz, 1991] M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.
- [Heymans and Vermeir, 2003] S. Heymans and D. Vermeir. Integrating Semantic Web reasoning and Answer Set Programming. In *Proc. ASP-2003*, pp. 194–208, 2003.
- [Kifer *et al.*, 1995] M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *J. ACM*, 42(4):741–843, 1995.
- [Leone *et al.*, 2005] N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello. The DLV system for knowledge representation and reasoning. *ACM Transactions on Computational Logic*, 2005. To appear.
- [Łukaszewicz, 1990] W. Łukaszewicz. *Non-monotonic Reasoning: Formalizations of Commonsense Reasoning*. Ellis Horwood, 1990.
- [Marek *et al.*, 2004] V. Marek, I. Niemelä, and M. Truszczyński. Logic programs with monotone cardinality atoms. In *Proc. LPNMR-2004*, pp. 154–166, 2004.
- [Poole, 1988] D. Poole. A logical framework for default reasoning. *Artificial Intelligence*, 36:27–47, 1988.
- [Rosati, 1999] R. Rosati. Towards expressive KR systems integrating datalog and description logics: Preliminary report. In *Proc. DL-1999*, pp. 160–164, 1999.
- [Ross, 1994] K. A. Ross. On negation in HiLog. *Journal of Logic Programming*, 18(1):27–53, 1994.
- [Simons *et al.*, 2002] P. Simons, I. Niemelä, and T. Soinen. Extending and implementing the stable model semantics. *Artificial Intelligence*, 138:181–234, 2002.
- [Sintek and Decker, 2002] M. Sintek and S. Decker. TRIPLE - A query, inference, and transformation language for the Semantic Web. In *Proc. ISWC-2002*, pp. 364–378, 2002.

On Solution Correspondences in Answer-Set Programming*

Thomas Eiter, Hans Tompits, and Stefan Woltran

Institut für Informationssysteme, Technische Universität Wien,
Favoritenstraße 9–11, A-1040 Vienna, Austria
[eiter,tompits,stefan]@kr.tuwien.ac.at

Abstract

We introduce a general framework for specifying program correspondence under the answer-set semantics. The framework allows to define different kinds of equivalence notions, including previously defined notions like strong and uniform equivalence, in which programs are extended with rules from a given context, and correspondence is determined by means of a binary relation. In particular, refined equivalence notions based on projected answer sets can be defined within this framework, where not all parts of an answer set are of relevance. We study general characterizations of inclusion and equivalence problems, introducing novel semantical structures. Furthermore, we deal with the issue of determining counterexamples for a given correspondence problem, and we analyze the computational complexity of correspondence checking.

1 Introduction

With the availability of efficient implementations of the answer-set semantics [Gelfond and Lifschitz, 1991] for non-monotonic logic programs, *answer-set programming* (ASP) has been recognized as a fruitful paradigm for declarative problem solving. In this approach, a problem is encoded as a logic program such that its models, called *answer sets*, correspond to the solutions of the problem, which can be easily extracted from them. Due to the availability of default negation, ASP has become an important host for solving many AI problems, including planning, diagnosis, information integration, and inheritance reasoning (cf. Gelfond and Leone [2002] for an overview on ASP).

To support engineering of ASP solutions, an important issue is determining equivalence of different encodings, given by two programs. To this end, various notions of equivalence between programs under the answer-set semantics have been studied in the recent past, viz. *strong equivalence* [Lifschitz et al., 2001], *uniform equivalence* [Eiter and Fink, 2003], and relativized notions thereof [Woltran, 2004], as well as *update*

equivalence [Inoue and Sakama, 2004]. Informally, the former notions consider programs P and Q to be equivalent, if $P \cup R$ and $Q \cup R$ have always the same answer sets, where R is a set of rules from a particular collection of rules. Thus, if R is regarded as possible input, equivalent P and Q are guaranteed to compute always the same answer sets. Note that also open logic programs [Bonatti, 2001], in which part of the rules are to be added at runtime, fit within this scheme.

However, none of these works have considered the practically important setting of *projected* answer sets in ASP. Here, not a whole answer set of a program P is of interest, but only its intersection on a subset $B \subseteq U$ of all letters; this includes, in particular, removal of auxiliary letters in computation. For a simple example, consider the programs

$$\begin{aligned} P &= \{ \text{sel}(X) \leftarrow s(X), \text{not out}(X), \\ &\quad \text{out}(X) \vee \text{out}(Y) \leftarrow s(X), s(Y), X \neq Y \} \text{ and} \\ Q &= \{ \text{sel}(X) \leftarrow s(X), \text{not skip}(X), \\ &\quad \text{skip}(X) \leftarrow \text{sel}(X), s(Y), X \neq Y \}. \end{aligned}$$

They should select, by means of *sel*, one element satisfying s in their answer sets. Here, an important issue is whether the programs are equivalent with respect to the “output” predicate *sel*, for all “inputs” s , where s may be defined by rules over predicates from a set A , say, or given by facts.

Another aspect is that, apart from equivalence, other relationships between the sets of answer sets of P and Q might be of interest. A natural example is *inclusion*, which means that each answer set of P is also an answer set of Q . Here Q can be viewed as an approximation of P , which is sound with respect to cautious reasoning from P .

Motivated by these observations, in this paper, we consider solution correspondences in ASP at a generic level. Our main contributions are briefly summarized as follows.

(1) We introduce a general framework for correspondences between the answer sets of programs P and Q , which are augmented by further rules from a *context* C of possible extensions, where the correspondence is determined by a binary relation ρ . Previous notions of equivalence and projected answer sets amount to particular instances of this framework.

(2) We provide characterizations of inclusion correspondence and equivalence between programs under projected answer sets, in terms of novel semantical structures, called *spoilers*, which refute this property, and *certificates*, which

*This work was supported by the Austrian Science Fund under grant P18019, and by the EC via projects FET-2001-37004 WASP, IST-2001-33570 INFOMIX, and IST-2001-33123 CologNeT.

capture the essence of equivalence, similar as SE-models and UE-models do for strong and uniform equivalence, respectively. Based on these characterizations, we present interesting correspondence results on varying projection sets in ASP.

(3) We show how spoilers can be used to construct *counterexamples* to an inclusion resp. equivalence correspondence, consisting of a suitable interpretation M and a rule set R such that M is an answer set of exactly one of $P \cup R$ and $Q \cup R$.

(4) Finally, we determine the computational complexity of correspondence checking for propositional disjunctive programs under projected answer sets. Our main result is that equivalence checking is Π_4^P -complete in general, and thus feasible in polynomial space as compared to a naive guess-and-check procedure which requires exponential space. Furthermore, we show that for restricted settings, the complexity gradually decreases from Π_4^P to coNP.

The results presented in this paper significantly advance the current state of equivalence testing in ASP towards highly relevant settings for practical applications, and provide novel insight into the structure of equivalent programs. Besides the papers quoted above, further related work mostly addresses semantic and complexity characterizations of equivalence [Lin, 2002; Turner, 2003], or describes implementation methods [Eiter *et al.*, 2004; Oikarinen and Janhunen, 2004]. The recent work by Pearce and Valverde [2004] addresses strong equivalence of programs over disjoint alphabets which are synonymous under structurally defined mappings.

The characterizations we present are non-trivial and, as shown by our complexity results, necessarily involved in the sense that no “simple” criteria exist. Together with counterexamples, they provide a basis for powerful program optimization and debugging tools for ASP, which are lacking to date.

2 Preliminaries

We deal with propositional disjunctive logic programs, which are finite sets of rules of form

$a_1 \vee \dots \vee a_l \leftarrow a_{l+1}, \dots, a_m, \text{not } a_{m+1}, \dots, \text{not } a_n, \quad (1)$
 $n \geq m \geq l \geq 0$, where all a_i are propositional atoms and *not* denotes default negation; for $n = l = 1$, we usually identify the rule with the atom a_1 . If all atoms occurring in a program P are from a given set A of atoms, we say that P is a program *over* A . The set of all programs over A is denoted by $\mathcal{P}_A$. We call a rule (resp., program) *normal* iff it contains no disjunction $\vee$. A program is *unary* if it contains only rules of the form $a \leftarrow b$ or $a \leftarrow \text{not } b$; $\mathcal{P}_A^{\text{un}}$ denotes the set of all unary programs over A . For a set of atoms $A = \{a_1, \dots, a_m\}$, *not* A denotes the set $\{\text{not } a_1, \dots, \text{not } a_m\}$. Accordingly, rules of form (1) will also be written as $a_1 \vee \dots \vee a_l \leftarrow B_1, \text{not } B_2$, where $B_1 = \{a_{l+1}, \dots, a_m\}$ and $B_2 = \{a_{m+1}, \dots, \text{not } a_n\}$.

Following Gelfond and Lifschitz [1991], an interpretation I , i.e., a set of atoms, is an *answer set* of a program P iff it is a minimal model of the *reduct* P^I , which results from P by (i) deleting all rules containing default negated atoms *not* a such that $a \in I$ and (ii) deleting all default negated atoms in the remaining rules. The set of all answer sets of a program P is denoted by $\mathcal{AS}(P)$. The relation $I \models P$ between an interpretation I and a program P is defined as usual.

Under the answer-set semantics, two programs P and Q are regarded as (ordinarily) equivalent iff $\mathcal{AS}(P) = \mathcal{AS}(Q)$. The more restrictive forms of *strong equivalence* [Lifschitz *et al.*, 2001] and *uniform equivalence* [Eiter and Fink, 2003] have recently been generalized as follows [Woltran, 2004]: Let P, Q be programs over $\mathcal{U}$, and let $A \subseteq \mathcal{U}$. Then, P and Q are *strongly* (resp., *uniformly*) *equivalent relative to* A iff, for any $R \in \mathcal{P}_A$ (resp., $R \subseteq A$), $\mathcal{AS}(P \cup R) = \mathcal{AS}(Q \cup R)$.

If $A = \mathcal{U}$, strong (resp., uniform) equivalence relative to A reduces to strong (resp., uniform) equivalence simpliciter; if $A = \emptyset$, ordinary equivalence results in either case.

We use the following notation. For an interpretation I and a set S of (pairs of) interpretations, we write $S|_I = \{Y \cap I \mid Y \in S\}$ ($S|_I = \{(X \cap I, Y \cap I) \mid (X, Y) \in S\}$). If $S = \{s\}$, we write $s|_I$ instead of $S|_I$, if convenient.

For any $A \subseteq \mathcal{U}$ and any $Y \subseteq \mathcal{U}$, a pair (X, Y) of interpretations is an *A-SE-interpretation* (over $\mathcal{U}$) iff either $X = Y$ or $X \subset Y|_A$. (X, Y) is an *A-SE-model* of a program P iff

- (i) $Y \models P$;
- (ii) for all $Y' \subset Y$ with $Y'|_A = Y|_A$, $Y' \not\models P^Y$; and
- (iii) $X \subset Y$ implies the existence of an $X' \subseteq Y$ with $X'|_A = X$ such that $X' \models P^Y$ holds.

A pair (X, Y) is *total* iff $X = Y$, and *non-total* otherwise. The set of all *A-SE-models* of P is denoted by $SE^A(P)$.

For $A = \mathcal{U}$, *A-SE-interpretations* (resp., *A-SE-models*) are simply called *SE-interpretations* (resp., *SE-models*), coinciding with the notions defined by Turner [2003], and we write $SE(P)$ instead of $SE^{\mathcal{U}}(P)$.

Proposition 1 ([Woltran, 2004]) *Two programs P and Q are strongly equivalent relative to A iff $SE^A(P) = SE^A(Q)$.*

Example 1 *Consider the following two programs, P_1 and P_2 , which we shall use as a running example:*

$$P_1 = P_0 \cup \{c \vee d \leftarrow a; c \vee d \leftarrow b\},$$

$$P_2 = P_0 \cup \{c \vee d \leftarrow a, b; d \leftarrow b, \text{not } c; c \leftarrow a, \text{not } d\},$$

for $P_0 = \{a \leftarrow c; b \leftarrow c; a \leftarrow d; b \leftarrow d; \leftarrow \text{not } c, \text{not } d\}$.

They have the following SE-models:¹

$$SE(P_1) = \{(\emptyset, abc), (\emptyset, abd), (\emptyset, abcd), (abcd, abcd),$$

$$(abc, abcd), (abd, abcd), (abc, abc), (abd, abd)\},$$

$$SE(P_2) = SE(P_1) \cup \{(b, abc), (a, abd), (b, abcd), (a, abcd)\}.$$

Hence, P_1 and P_2 are not strongly equivalent. On the other hand, $\mathcal{AS}(P_1) = \mathcal{AS}(P_2) = \emptyset$, i.e., P_1 and P_2 are (ordinarily) equivalent. Moreover, P_1 and P_2 are strongly equivalent relative to A iff $A \cap \{a, b\} = \emptyset$. For $A = \{a, b\}$, we get

$$SE^A(P_1) = \{(\emptyset, abc), (\emptyset, abd), (abc, abc), (abd, abd)\},$$

$$SE^A(P_2) = SE^A(P_1) \cup \{(b, abc), (a, abd)\}.$$

3 General Framework

In order to deal with differing notions of program equivalence in a uniform manner, taking in particular the currently existing notions of equivalence, as presented above, as well as

¹We write *abc* instead of $\{a, b, c\}$, *a* instead of $\{a\}$, etc.

equivalence notions based on the projection of answer sets into account, we introduce a general framework for expressing solution correspondences between logic programs. In this framework, we parameterize, on the one hand, the set R of rules to be added to the programs P and Q , and, on the other hand, the relation that has to hold between the collection of answer sets of $P \cup R$ and $Q \cup R$. Concerning the latter parameter, besides equality, other comparison relations like set-inclusion may be used. This leads to the following notion:

Definition 1 By a correspondence frame, or simply frame, $\mathcal{F}$, we understand a triple $(\mathcal{U}, \mathcal{C}, \rho)$, where (i) $\mathcal{U}$ is a set of atoms, called the universe of $\mathcal{F}$, (ii) $\mathcal{C} \subseteq \mathcal{P}_{\mathcal{U}}$, called the context programs of $\mathcal{F}$, or simply the context, and (iii) $\rho \subseteq 2^{2^{\mathcal{U}}} \times 2^{2^{\mathcal{U}}}$.

For all programs $P, Q \in \mathcal{P}_{\mathcal{U}}$, we say that P and Q are $\mathcal{F}$ -corresponding, symbolically $P \simeq_{\mathcal{F}} Q$, iff, for all $R \in \mathcal{C}$, $(\mathcal{AS}(P \cup R), \mathcal{AS}(Q \cup R)) \in \rho$.

Intuitively, in a correspondence frame $\mathcal{F} = (\mathcal{U}, \mathcal{C}, \rho)$, $\mathcal{U}$ determines the general alphabet under consideration, $\mathcal{C}$ determines the kind of rules used for comparison, and ρ is the specific operation used for checking the correspondence of two programs.

It is quite obvious that the equivalence notions discussed above are special cases of $\mathcal{F}$ -correspondence. Indeed, for any universe $\mathcal{U}$ and any $A \subseteq \mathcal{U}$, we have that strong equivalence relative to A coincides with $(\mathcal{U}, \mathcal{P}_A, =)$ -correspondence, and uniform equivalence relative to A coincides with $(\mathcal{U}, 2^A, =)$ -correspondence. Consequently, it holds that (i) strong equivalence coincides with $(\mathcal{U}, \mathcal{P}_{\mathcal{U}}, =)$ -correspondence, (ii) uniform equivalence coincides with $(\mathcal{U}, 2^{\mathcal{U}}, =)$ -correspondence, and (iii) ordinary equivalence coincides with $(\mathcal{U}, \{\emptyset\}, =)$ -correspondence.

In this paper, we are mainly concerned with correspondence frames of form $(\mathcal{U}, \mathcal{P}_A, \subseteq_B)$ and $(\mathcal{U}, \mathcal{P}_A, =_B)$, where $A \subseteq \mathcal{U}$ is some set of atoms, and $\subseteq_B$ and $=_B$ are projections of the standard subset and set-equality relation, respectively, to a set $B \subseteq \mathcal{U}$, defined as follows: for sets $\mathcal{S}, \mathcal{S}'$ of interpretations,

- $\mathcal{S} \subseteq_B \mathcal{S}'$ iff $\mathcal{S}|_B \subseteq \mathcal{S}'|_B$, and
- $\mathcal{S} =_B \mathcal{S}'$ iff $\mathcal{S}|_B = \mathcal{S}'|_B$.

In particular, $=_B$ amounts to answer-set existence if $B = \emptyset$, and to correspondence between answer sets if $B = \mathcal{U}$.

In what follows, if $\mathcal{F}$ is of the form $(\mathcal{U}, \mathcal{C}, =_B)$, we refer to $\mathcal{F}$ also as an *equivalence frame*, and, accordingly, to $\mathcal{F}$ -correspondence also as $\mathcal{F}$ -equivalence. As well, a frame of the form $(\mathcal{U}, \mathcal{C}, \subseteq_B)$ is also referred to as an *inclusion frame*.

For later purposes, we also introduce the following notion: A *correspondence problem*, Π , over $\mathcal{U}$, is a quadruple $(P, Q, \mathcal{C}, \rho)$, where $P, Q \in \mathcal{P}_{\mathcal{U}}$ and $(\mathcal{U}, \mathcal{C}, \rho)$ is a frame. We say that Π holds iff $P \simeq_{(\mathcal{U}, \mathcal{C}, \rho)} Q$ holds. In accord with the above designations, we call Π an *equivalence problem* if $(\mathcal{U}, \mathcal{C}, \rho)$ is an equivalence frame and an *inclusion problem* if $(\mathcal{U}, \mathcal{C}, \rho)$ is an inclusion frame.

For a correspondence problem $\Pi = (P, Q, \mathcal{C}, \rho)$ over $\mathcal{U}$, we usually leave $\mathcal{U}$ implicit, assuming that it consists of all atoms occurring in P , Q , and $\mathcal{C}$.

We next list some basic properties of $\mathcal{F}$ -equivalence.

Proposition 2 Let $(\mathcal{U}, \mathcal{C}, =_B)$ be an equivalence frame and $P, Q \in \mathcal{P}_{\mathcal{U}}$. Then, the following conditions hold:

1. If $P \simeq_{(\mathcal{U}, \mathcal{C}, =_B)} Q$, then $P \simeq_{(\mathcal{U}, \mathcal{C}', =_{B'})} Q$, for all $\mathcal{C}' \subseteq \mathcal{C}$ and all $B' \subseteq B$.
2. $P \simeq_{(\mathcal{U}, \mathcal{C}, =_B)} Q$ iff $P \simeq_{(\mathcal{U}, \mathcal{C}, \subseteq_B)} Q$ and $Q \simeq_{(\mathcal{U}, \mathcal{C}, \subseteq_B)} P$.

We recall an important result due to Woltran [2004], extending an analogous result by Lifschitz *et al.* [2001].

Proposition 3 Consider a frame $(\mathcal{U}, \mathcal{P}_A, =)$, for $A \subseteq \mathcal{U}$, and let $P, Q \in \mathcal{P}_{\mathcal{U}}$. Then, $P \simeq_{(\mathcal{U}, \mathcal{P}_A, =)} Q$ iff $P \simeq_{(\mathcal{U}, \mathcal{P}_A^{un}, =)} Q$.

Example 2 We have already seen that for P_1, P_2 from Example 1, $P_1 \not\simeq_{(\mathcal{U}, \mathcal{P}_A, =)} P_2$ holds, for $A = \{a, b\}$ and $\mathcal{U} = \{a, b, c, d\}$. Hence, by Proposition 3, $P_1 \not\simeq_{(\mathcal{U}, \mathcal{P}_A^{un}, =)} P_2$. This is witnessed by the fact that, e.g., $\mathcal{AS}(P_1 \cup \{a\}) = \{abc, abd\}$ while $\mathcal{AS}(P_2 \cup \{a\}) = \{abc\}$.

The relevance of Proposition 3 is that it allows to drastically reduce the number of required rules for equivalence checking. However, the proposition does not generalize to projections of answer sets.

Theorem 1 For any equivalence frame $(\mathcal{U}, \mathcal{P}_A, =_B)$ with $A, B \subseteq \mathcal{U}$, and any $P, Q \in \mathcal{P}_{\mathcal{U}}$, if $P \simeq_{(\mathcal{U}, \mathcal{P}_A, =_B)} Q$, then $P \simeq_{(\mathcal{U}, \mathcal{P}_A^{un}, =_B)} Q$, but the converse does not hold in general.

Proof. That $P \simeq_{(\mathcal{U}, \mathcal{P}_A, =_B)} Q$ implies $P \simeq_{(\mathcal{U}, \mathcal{P}_A^{un}, =_B)} Q$ is immediate from Part 1 of Proposition 2. Our running example shows the failure of the converse: $P_1 \simeq_{(\mathcal{U}, \mathcal{P}_A^{un}, =_B)} P_2$ holds, where $\mathcal{U} = \{a, b, c, d\}$ and $A = B = \{a, b\}$, but $P_1 \not\simeq_{(\mathcal{U}, \mathcal{P}_A, =_B)} P_2$. The latter holds in view of $\mathcal{AS}(P_1 \cup \{a \vee b \leftarrow\}) = \{abc, abd\}$ but $\mathcal{AS}(P_2 \cup \{a \vee b \leftarrow\}) = \emptyset$, as easily shown. $\square$

The above theorem also holds if the set $\mathcal{P}_A^{un}$ is substituted by the class of all normal programs over A . Indeed, programs P_1 and P_2 of our running example are corresponding with respect to a frame using normal programs over $A = \{a, b\}$ as context, because in each stable model of $P_1 \cup R$ resp. $P_2 \cup R$, both a and b must be true. Thus, rules in R ($\subseteq \mathcal{P}_A$) with negative literals in the body are immaterial.

Theorem 1 indicates that equivalence for projected answer sets is more involved. The same holds for inclusion, since as an easy corollary to Theorem 1, $P \simeq_{(\mathcal{U}, \mathcal{P}_A, \subseteq_B)} Q$ implies $P \simeq_{(\mathcal{U}, \mathcal{P}_A^{un}, \subseteq_B)} Q$, but not vice versa. In fact, the next result shows that, in general, a smallest extension R violating inclusion has exponential size.

Theorem 2 There exists a family of problems $\Pi = (P, Q, \mathcal{P}_A, \subseteq_A)$ such that each $R \subseteq \mathcal{P}_A$ witnessing $\mathcal{AS}(P \cup R)|_A \not\subseteq \mathcal{AS}(Q \cup R)|_A$ is exponential in the size of P and Q .

Proof (Sketch). The idea is to encode a propositional CNF $\phi = \bigwedge_{i=1}^n C_i$ over atoms V , for $C_i = c_{i,1} \vee \dots \vee c_{i,k_i}$, into $\Pi = (P, Q, \mathcal{P}_A, \subseteq_A)$ such that, for any $R \in \mathcal{P}_A$, $\mathcal{AS}(P \cup R)|_A \not\subseteq \mathcal{AS}(Q \cup R)|_A$ and $\neg \tilde{R}$ must include a DNF for ϕ , where $\tilde{R}$ is the result of interpreting R as a classical formula.

Let $\bar{V} = \{\bar{v} \mid v \in V\}$, $V' = \{v' \mid v \in V\}$, $\bar{V}' = \{\bar{v}' \mid v \in V\}$, and $G = \{g_1, \dots, g_n\}$ be sets of new atoms. We define

$$\begin{aligned} P &= \{v \vee \bar{v} \leftarrow; \leftarrow \text{not } v; \leftarrow \text{not } \bar{v} \mid v \in V\} \cup \\ &\quad \{v \leftarrow u, \bar{u}; \bar{v} \leftarrow u, \bar{u} \mid v, u \in V\} \cup \\ &\quad \{v \leftarrow C_i^*; \bar{v} \leftarrow C_i^* \mid v \in V; 1 \leq i \leq n\}, \end{aligned}$$

where $C_i^* = c_{i,1}^*, \dots, c_{i,k_i}^*$, $v^* = \bar{v}$, and $(\neg v)^* = v$, and

$$Q = \{v \vee \bar{v} \leftarrow; v' \leftarrow \bar{v}, \text{not } v'; \bar{v}' \leftarrow v, \text{not } v' \mid v \in V\} \cup \\ \{v' \leftarrow \bar{v}', \bar{v}' \leftarrow \text{not } v', \text{not } \bar{v}' \mid v \in V\} \cup \\ \{v \leftarrow u'; \bar{v} \leftarrow u'; v \leftarrow \bar{u}'; \bar{v} \leftarrow \bar{u}' \mid v, u \in V\} \cup \\ \{g_i \leftarrow \hat{c}_{i,j} \mid i = 1..n, j = 1..k_i\} \cup \{\leftarrow g_1, \dots, g_n\},$$

with $\hat{v} = v'$ and $\hat{\bar{v}} = \bar{v}'$. Let $A = V \cup \bar{V}$. Informally, $P \cup R$ only admits answer sets containing A , and indeed it holds that $(A, A) \in SE(P \cup R)|_A \setminus SE(Q \cup R)|_A$, for some R . For any such R , the SE-models (X, A) with $X \subset A$ must be precisely those where X is a countermodel of ϕ . This means that $\bar{R}^A$ is a CNF for $\neg\phi$, and thus $\neg\bar{R}^A$ amounts to a DNF for ϕ . This proves the claim. Now, as well-known, the smallest DNF for a CNF ϕ can be exponential in ϕ , which proves the result. $\square$

We can similarly construct a family of problems showing that a smallest extension R violating equivalence has exponential size. Hence, a naive guess and check algorithm to disprove inclusion or equivalence needs exponential space. For a better algorithm, we have to develop suitable semantic characterizations, which must take disjunctive extensions into account.

4 Characterizations

In this section, we first present some characteristic structures associated with inclusion and equivalence problems under projections of answer sets, termed *spoilers* and *certificates*. Based on them, we then discuss some interesting invariance results.

We start with some general properties.

Definition 2 A set $\mathcal{S}$ of SE-interpretations is complete iff, for each $(X, Y) \in \mathcal{S}$, also $(Y, Y) \in \mathcal{S}$ as well as $(X, Z) \in \mathcal{S}$, for any $Y \subseteq Z$ with $(Z, Z) \in \mathcal{S}$.

It can be shown that the set $SE(P)$ of all SE-models of a program P is always complete.

The following guarantees that a complete set $\mathcal{S}$ of SE-interpretations can be represented by some program P .

Proposition 4 Let $\mathcal{S}$ be a complete set of SE-interpretations, and let A be a set of atoms. Then, there exists a program $P_{\mathcal{S},A} \in \mathcal{P}_A$ such that $SE(P_{\mathcal{S},A})|_A = \mathcal{S}|_A$.

One possibility to obtain $P_{\mathcal{S},A}$ from $\mathcal{S}$ is as follows: take rules $\leftarrow Y, \text{not } (A \setminus Y)$, for each $Y \subseteq A$ such that $(Y, Y) \notin \mathcal{S}|_A$, and rules $\bigvee_{p \in (Y \setminus X)} p \leftarrow X, \text{not } (A \setminus Y)$, for each $X \subset Y$ such that $(X, Y) \notin \mathcal{S}|_A$ and $(Y, Y) \in \mathcal{S}|_A$.

4.1 Spoilers

The first class of characteristic structures associated with program correspondence we are dealing with are of such a nature that their existence prevents the equivalence of programs under projected answer sets.

We need the following auxiliary notation: Let $\mathcal{S}$ be a set of SE-interpretations and Y, C sets of atoms. Then, $\sigma_Y^C(\mathcal{S}) = \{(X, Z) \in \mathcal{S} \mid Z|_C = Y|_C\}$.

Definition 3 Let $\Pi = (P, Q, \mathcal{P}_A, \subseteq_B)$ be an inclusion problem over $\mathcal{U}$, let $Y \subseteq \mathcal{U}$ be an interpretation, and consider $\mathcal{S} \subseteq \sigma_Y^{A \cup B}(SE^A(Q))$. The pair $(Y, \mathcal{S})$ is a spoiler for Π iff

- (i) $(Y, Y) \in SE^A(P)$,
- (ii) for each $(Z, Z) \in \mathcal{S}$, some non-total $(X, Z) \in \mathcal{S}$ exists,
- (iii) $(Z, Z) \in \mathcal{S}$ iff $(Z, Z) \in \sigma_Y^{A \cup B}(SE^A(Q))$, and
- (iv) $(X, Z) \in \mathcal{S}$ implies $(X, Y) \notin SE^A(P)$.

Intuitively, in a spoiler $(Y, \mathcal{S})$, the interpretation Y is an answer set of $P \cup R$ but not of $Q \cup R$, for some R , which is semantically given by $\mathcal{S}$.

Example 3 For P_1 and P_2 from our running example and $A = \{a, b\}$, $(Y_1, \mathcal{S})$ and $(Y_2, \mathcal{S})$ are the only spoilers for $(P_1, P_2, \mathcal{P}_A, \subseteq_A)$, where $Y_1 = \{abc\}$, $Y_2 = \{abd\}$, and $\mathcal{S} = \{(a, abd), (b, abc), (abc, abc), (abd, abd)\}$, with the latter being a subset of $\sigma_{Y_1}^A(SE^A(P_2)) = \sigma_{Y_2}^A(SE^A(P_2)) = \mathcal{S} \cup \{(\emptyset, abc), (\emptyset, abd)\}$, as required in Definition 3.

The central property of spoilers is as follows:

Theorem 3 Let $\mathcal{F} = (\mathcal{U}, \mathcal{P}_A, \subseteq_B)$ be a frame. Then, for any $P, Q \in \mathcal{P}_\mathcal{U}$, $P \simeq_{\mathcal{F}} Q$ iff there is no spoiler for $(P, Q, \mathcal{P}_A, \subseteq_B)$.

An immediate consequence of this theorem, together with Part 2 of Proposition 2, is the following result:

Corollary 1 Let $\mathcal{F} = (\mathcal{U}, \mathcal{P}_A, =_B)$ be an equivalence frame and $P, Q \in \mathcal{P}_\mathcal{U}$. Then, $P \simeq_{\mathcal{F}} Q$ iff neither $(P, Q, \mathcal{P}_A, \subseteq_B)$ nor $(Q, P, \mathcal{P}_A, \subseteq_B)$ has a spoiler.

As discussed later on, spoilers provide a semantical basis for counterexample generation.

4.2 Certificates

After having introduced structures which *disprove* program correspondence, we now discuss structures which *prove* program correspondence. Roughly speaking, the structures introduced below express the essence of a program P , with respect to program equivalence, in terms of a semantic condition on P alone.

Definition 4 Let C be a set of atoms and $\mathcal{S}$ a set of SE-interpretations. A pair $(\mathcal{X}, Y)$, where $\mathcal{X}$ is a set of interpretations and $Y \subseteq C$, is a C -projection of $\mathcal{S}$ iff there exists some set Z such that (i) $(Z, Z) \in \mathcal{S}$, (ii) $Z|_C = Y|_C$, and (iii) $\mathcal{X} = \{X \mid (X, Z) \in \mathcal{S}, X \subset Z\}$.

For a program P , we call an $(A \cup B)$ -projection of $SE^A(P)$ an (A, B) -certificate of P .

The following lemma can be shown by means of spoilers, and expresses that programs are corresponding with respect to inclusion frames iff their certificates satisfy a certain containment relation.

Lemma 1 Correspondence of $(P, Q, \mathcal{P}_A, \subseteq_B)$ holds, iff for each (A, B) -certificate $(\mathcal{X}, Y)$ of P , an (A, B) -certificate $(\mathcal{X}', Y)$ of Q exists with $\mathcal{X}' \subseteq \mathcal{X}$.

The next result expresses the central property of certificates. Towards its formulation, we require a further concept: An (A, B) -certificate $(\mathcal{X}, Y)$ of a program P is *minimal* iff, for any (A, B) -certificate $(\mathcal{Z}, Y)$ of P , $\mathcal{Z} \subseteq \mathcal{X}$ implies $\mathcal{Z} = \mathcal{X}$.

Theorem 4 Let $\mathcal{F} = (\mathcal{U}, \mathcal{P}_A, =_B)$ be an equivalence frame. Then, for any $P, Q \in \mathcal{P}_\mathcal{U}$, $P \simeq_{\mathcal{F}} Q$ iff the minimal (A, B) -certificates of P and Q coincide.

Note that this result is the pendant to Proposition 1, which deals with a model-theoretic characterization of relativized strong equivalence. Indeed, two programs $P, Q \in \mathcal{P}_{\mathcal{U}}$ are strongly equivalent relative to A iff their minimal $(A, \mathcal{U})$ -certificates coincide. Some further relations between program correspondence and relativized and non-relativized strong equivalence, respectively, are given in the next subsection.

Example 4 In our running example with $A = B = \{a, b\}$, we get that P_1 has a single (A, A) -certificate, $(\{\emptyset\}, \{ab\})$, while P_2 has two (A, A) -certificates, $(\{\emptyset, \{a\}\}, \{ab\})$ and $(\{\emptyset, \{b\}\}, \{ab\})$, all of them minimal. Since they do not coincide, we obtain that P_1 and P_2 are not $(\mathcal{U}, \mathcal{P}_A, =_A)$ -equivalent, as expected.

4.3 Invariance Results

Theorem 4 allows us to derive some interesting invariance results with respect to varying projection sets B .

Theorem 5 Let $\mathcal{U}$ be a set of atoms and $A, B \subseteq \mathcal{U}$. Then, for any $P, Q \in \mathcal{P}_{\mathcal{U}}$, $P \simeq_{(\mathcal{U}, \mathcal{P}_A, =_B)} Q$ iff $P \simeq_{(\mathcal{U}, \mathcal{P}_A, =_{A \cup B})} Q$.

This result follows immediately from Theorem 4 and Definition 4, by observing that (A, B) -certificates of a program P and $(A, A \cup B)$ -certificates of P are actually identical objects, since $(A \cup B)$ -projections of $SE^A(P)$ trivially coincide with $(A \cup (A \cup B))$ -projections of $SE^A(P)$.

Theorem 5 has several interesting consequences.

Corollary 2 For programs P, Q and any set A of atoms, $(P, Q, \mathcal{P}_A, =_{\emptyset})$ holds iff $(P, Q, \mathcal{P}_A, =_A)$ holds.

That is, answer-set existence (which is relevant regarding Boolean properties) relative to additions from $\mathcal{P}_A$ is tantamount to relativized strong equivalence under projection to A .

Corollary 3 Let $\mathcal{U}$ be a set of atoms. Then, for any programs $P, Q \in \mathcal{P}_{\mathcal{U}}$ and any set $B \subseteq \mathcal{U}$ of atoms, $P \simeq_{(\mathcal{U}, \mathcal{P}_{\mathcal{U}}, =_B)} Q$ iff P and Q are strongly equivalent.

This result is quite striking as it shows that strong equivalence corresponds to $(\mathcal{U}, \mathcal{P}_{\mathcal{U}}, =_B)$ -equivalence, for any projection set B . It is derived from the fact that, for any B , the $(\mathcal{U}, B)$ -certificates of a program P are in a one-to-one correspondence to the SE-models of P as follows: $(\{X_1, \dots, X_m\}, Y)$ is a $(\mathcal{U}, B)$ -certificate of P iff $(X_1, Y), \dots, (X_m, Y), (Y, Y)$ are all the SE-models of P with fixed second component Y .

In particular, consistency under answer-set semantics (i.e., if $B = \emptyset$) coincides with strong equivalence. More generally:

Corollary 4 Let $\mathcal{U}$ be a set of atoms. Then, for all programs $P, Q \in \mathcal{P}_{\mathcal{U}}$ and all sets A and B of atoms such that $A \cup B = \mathcal{U}$, $P \simeq_{(\mathcal{U}, \mathcal{P}_A, =_B)} Q$ iff P and Q are strongly equivalent relative to A .

In fact, in this setting, a correspondence between A -SE-models of a program and its (A, B) -certificates is analogously established as above.

5 Counterexamples

Given that a correspondence problem $\Pi = (P, Q, \mathcal{P}_A, \subseteq_B)$ does not hold, it is interesting to know why this is the case. We define a *counterexample* for Π as a pair (R, M) , where

- $R \in \mathcal{P}_A$ such that $\mathcal{AS}(P \cup R) \not\subseteq_B \mathcal{AS}(Q \cup R)$, and
- $M \in \mathcal{AS}(P \cup R)$ and $M|_B \notin \mathcal{AS}(Q \cup R)|_B$.

Furthermore, a counterexample for an equivalence problem $(P, Q, \mathcal{P}_A, =_B)$ is any counterexample for either $(P, Q, \mathcal{P}_A, \subseteq_B)$ or $(Q, P, \mathcal{P}_A, \subseteq_B)$.

Our notion of a spoiler from Definition 3 provides a basis for such counterexamples.

Theorem 6 Suppose $(Y, \mathcal{S})$ is a spoiler for a correspondence problem $\Pi = (P, Q, \mathcal{P}_A, \subseteq_B)$. Then, $(P_{\mathcal{S}, A}, Y)$ is a counterexample for Π , where $P_{\mathcal{S}, A}$ is as in Proposition 4.

This result follows from Theorem 3 and Proposition 4 by the fact that $\mathcal{S}|_A$ is complete for any spoiler $(Y, \mathcal{S})$.

Example 5 For our example and the sketched construction after Proposition 4, we derive counterexamples $(R, \{abc\})$ and $(R, \{abd\})$, where $R = \{a \vee b \leftarrow; \leftarrow \text{not } a; \leftarrow \text{not } b; \leftarrow \text{not } a, \text{not } b\}$; obviously, the last rule is redundant.

Note that if an inclusion problem $(P, Q, \mathcal{P}_A, \subseteq_B)$ fails, some counterexample as in Theorem 6 does exist.

We observe that the programs in the counterexamples of Theorem 6 may contain redundant clauses, as succinctness is not a concern of spoilers. For instance, in our example, $R' = \{a \vee b \leftarrow\}$ would yield a simpler counterexample. In fact, spoilers are not geared towards providing minimal counterexamples with respect to particular syntactic subclasses of contexts.

Towards facilitating special counterexamples, we may extend the notion of a spoiler to pairs $(Y, \mathcal{S})$, where $\mathcal{S} \not\subseteq \sigma_Y^{A \cup B}(SE^A(Q))$ is admitted for complete $\mathcal{S}$, by replacing in Definition 3 the set $\mathcal{S}$ in (ii) with $\mathcal{S} \cap \sigma_Y^B(SE^A(Q))$ and in (iii) and (iv) with $\mathcal{S} \cap \sigma_Y^{A \cup B}(SE^A(Q))$, calling the result an *extended spoiler*.

Example 6 In our running example, $\Pi = (P_1, P_2, \mathcal{P}_A, \subseteq_A)$ has spoilers $(\{abc\}, \mathcal{S})$ and $(\{abd\}, \mathcal{S})$, with $\mathcal{S} = \{(a, abd), (b, abc), (abc, abc), (abd, abd)\}$. Since both $(ab, ab) \notin SE^A(P_2)$ and $(abcd, abcd) \notin SE^A(P_2)$, one can verify that any complete superset $\mathcal{S}'$ of $\mathcal{S}$ not containing any $(\emptyset, Z)$, with $Z|_A = \{ab\}$, yields extended spoilers $(\{abc\}, \mathcal{S}')$ and $(\{abd\}, \mathcal{S}')$. In particular, we may set $\mathcal{S}' = \mathcal{S} \cup \{(a, a), (b, b)\}$. Note that $\mathcal{S}'|_A = \{(a, a), (a, ab), (b, b), (b, ab), (ab, ab)\}$. Now, the simpler program $R' = \{a \vee b \leftarrow\}$ fulfills $SE(R')|_A = \mathcal{S}'|_A$; thus, $(R', \{abc\})$ and $(R', \{abd\})$ are counterexamples for Π .

Theorems 3 and 6 generalize to extended spoilers. More counterexamples can be constructed using such spoilers in general, which may include a counterexample of particular form. In our example, adding $\{(a, a), (b, b)\}$ to an ordinary spoiler allowed us to give a counterexample program which is positive. On the other hand, we can show that no counterexample program which is normal exists (as no $(\emptyset, Z)$ with $Z|_A = \{ab\}$ can be added). An elaboration of this issue remains for further work.

6 Computational Complexity

Our first result is concerned with recognizing whether an interpretation is a discriminating answer set for the extended programs $P \cup R$ and $Q \cup R$, i.e., a “partial” spoiler.

Lemma 2 Given programs P and Q , sets A and B of atoms, and an interpretation Y , deciding whether $(P, Q, \mathcal{P}_A, \subseteq_B)$ has some spoiler whose first component is Y is Π_3^P -complete.

Proof (Sketch). We show Σ_3^P -membership of deciding that no spoiler of form $(Y, \mathcal{S})$ exists. By Definition 3, it suffices to check whether (a) $(Y, Y) \notin SE^A(P)$ or (b) whether there exists a $(Z, Z) \in \sigma_Y^{A \cup B}(SE^A(Q))$ such that each non-total $(X, Z) \in SE^A(Q)$ implies $(X, Y) \in SE^A(P)$. Part (a) is feasible in polynomial time with an NP oracle. For Part (b), note that given Y and Z , checking whether, for each non-total $(X, Z) \in SE^A(Q)$ also $(X, Y) \in SE^A(P)$, is in Π_2^P . Therefore, Part (b) is in Σ_3^P , and the entire test is in Σ_3^P .

The Π_3^P -hardness is shown by a sophisticated reduction from suitable quantified Boolean formulas (QBFs), using machinery from the exponential counterexample construction which was used for showing Theorem 2. $\square$

From this result, we can easily derive the membership part of our main complexity result given below; its hardness part is again shown by an encoding of QBFs.

Theorem 7 Given programs P and Q and sets A and B of atoms, deciding whether $(P, Q, \mathcal{P}_A, \subseteq_B)$ holds is Π_4^P -complete. Moreover, Π_4^P -hardness holds even for $B = \emptyset$, i.e., for answer-set existence.

For the particular case where $B = A$, which constitutes the setting where auxiliary letters are used in logic programs, we obtain in combination with Theorem 5 the same complexity.

Notice that the “partial” spoiler of Lemma 2 avoids a naive guess of a (possibly exponentially large) full spoiler $(Y, \mathcal{S})$ which proves the failure of $(P, Q, \mathcal{P}_A, \subseteq_B)$, at the expense of checking involved conditions on $SE^A(P)$ and $SE^A(Q)$.

The above results for inclusion problems carry over to equivalence problems, since they are polynomially intertranslatable, as seen by Part 2 of Proposition 2 and the following fact:

Proposition 5 $P \simeq_{(U, \mathcal{P}_A, \subseteq_B)} Q$ iff $Q \simeq_{(U, \mathcal{P}_A, =_B)} L_{P,Q}$, where $L_{P,Q} = \{g_P \vee g_Q \leftarrow; \leftarrow g_P, g_Q\} \cup \{H \leftarrow g_R, B \mid R \in \{P, Q\}, H \leftarrow B \in R\}$ and g_P, g_Q are new atoms.

This holds by virtue of Lemma 1 and Theorem 4, and the fact that the (A, B) -certificates of $L_{P,Q}$ are those of P and Q .

If the size of each program in the context $\mathcal{C}$ is polynomially bound by the size of the compared programs P and Q (as is the case for ordinary and uniform equivalence), the complexity is lower. Let us call such problems $(P, Q, \mathcal{C}, \rho)$ *bound*.

Theorem 8 Given programs P and Q , a context $\mathcal{C}$, a set B of atoms, such that $\Pi = (P, Q, \mathcal{C}, \subseteq_B)$ is bound, deciding whether Π holds is Π_3^P -complete. Π_3^P -hardness holds even for $\mathcal{C} = \{\emptyset\}$, i.e., for ordinary equivalence with projection.

For other instances of the framework, the complexity is even lower.

Theorem 9 Given programs P and Q over $\mathcal{U}$ and sets A and B of atoms, deciding whether $(P, Q, \mathcal{P}_A, =_B)$ holds is (i) coNP-complete if $A = \mathcal{U}$, and (ii) Π_2^P -complete if $(A \cup B) = \mathcal{U}$.

This result follows from the invariance results in Section 4.2 and complexity results due to Woltran [2004].

7 Conclusion and Further Work

We have presented a general framework for expressing solution correspondences between nonmonotonic logic programs, and have then developed semantic characterizations of inclusion and equivalence problems under projected answer sets within a context of possible changes. As we have shown, they match the intrinsic complexity of the problem.

Our results provide a semantical basis for developing optimization and debugging techniques, which are lacking at present but vital for further enhancements of ASP as a programming paradigm.

Several issues remain for future work. One is to extend our study to different classes of contexts and compared programs, and to provide suitable semantical and complexity characterizations. Another issue concerns the construction of “good” counterexamples, according to their possible form. Finally, exploring other notions of correspondences than $=_B$ and $\subseteq_B$ in the general framework is an intriguing issue.

References

- [Bonatti, 2001] P.A. Bonatti. Reasoning with open logic programs. In *Proc. LPNMR 2001*, pp. 147–159.
- [Eiter and Fink, 2003] T. Eiter and M. Fink. Uniform equivalence of logic programs under the stable model semantics. In *Proc. ICLP 2003*, pp. 224–238.
- [Eiter et al., 2004] T. Eiter, M. Fink, H. Tompits, and S. Woltran. Simplifying logic programs under uniform and strong equivalence. In *Proc. LPNMR-7*, pp. 87–99.
- [Gelfond and Leone, 2002] M. Gelfond and N. Leone. Logic Programming and Knowledge Representation - The A-Prolog Perspective. *AIJ*, 138(1-2):3–38, 2002.
- [Gelfond and Lifschitz, 1991] M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.
- [Inoue and Sakama, 2004] K. Inoue and C. Sakama. Equivalence of logic programs under updates. In *Proc. JELIA 2004*, pp. 174–186.
- [Lifschitz et al., 2001] V. Lifschitz, D. Pearce, and A. Valverde. Strongly equivalent logic programs. *ACM Trans. Computational Logic*, 2(4):526–541, 2001.
- [Lin, 2002] F. Lin. Reducing strong equivalence of logic programs to entailment in classical propositional logic. In *Proc. KR 2002*, pp. 170–176.
- [Oikarinen and Janhunen, 2004] E. Oikarinen and T. Janhunen. Verifying the equivalence of logic programs in the disjunctive case. In *Proc. LPNMR-7*, pp. 180–193.
- [Pearce and Valverde, 2004] D. Pearce and A. Valverde. Synonymous theories in answer set programming and equilibrium logic. In *Proc. ECAI 2004*, pp. 388–392.
- [Turner, 2003] H. Turner. Strong equivalence made easy: Nested expressions and weight constraints. *Theory & Practice of Logic Programming*, 3(4-5):602–622, 2003.
- [Woltran, 2004] S. Woltran. Characterizations for relativized notions of equivalence in answer set programming. In *Proc. JELIA 2004*, pp. 161–173.

Dual Lookups in Pattern Databases

Ariel Felner

Department of Information
Systems Engineering
Ben-Gurion University
Beer-Sheva, Israel 85104
Email: felner@bgu.ac.il

Uzi Zahavi

Computer Science Dept.
Bar-Ilan University
Ramat-Gan, Israel 92500
Email: zahavi@cs.biu.ac.il

Jonathan Schaeffer and Robert C. Holte

Computing Science Department
University of Alberta
Edmonton, T6G 2E8, Canada
Email: jonathan.@cs.ualberta.ca
Email: holte,@cs.ualberta.ca

Abstract

A pattern database (PDB) is a heuristic function stored as a lookup table. Symmetries of a state space are often used to enable multiple values to be looked up in a PDB for a given state. This paper introduces an additional PDB lookup, called the dual PDB lookup. A dual PDB lookup is always admissible but can return inconsistent values. The paper also presents an extension of the well-known pathmax method so that inconsistencies in heuristic values are propagated in both directions (child-to-parent, and parent-to-child) in the search tree. Experiments show that the addition of dual lookups and bidirectional pathmax propagation can reduce the number of nodes generated by IDA* by over one order of magnitude in the TopSpin puzzle and Rubik's Cube, and by about a factor of two for the sliding tile puzzles.

1 Introduction

Heuristic search algorithms such as A* and IDA* are guided by the cost function $f(n) = g(n) + h(n)$, where $g(n)$ is the actual distance from the initial state to state n and $h(n)$ is a heuristic function estimating the cost from n to a goal state. If $h(s)$ is “admissible” (i.e. is always a lower bound) these algorithms are guaranteed to find optimal paths.

Pattern databases are heuristics in the form of lookup tables. They have proven very useful for defining heuristics for combinatorial puzzles and other problems [Culberson and Schaeffer, 1994; Edelkamp, 2001; Korf and Felner, 2002].

The *domain* of a search space is the set of constants used in representing states. A *subproblem* is an abstraction of the original problem defined by replacing some of these constants by a “don’t care” symbol. A *pattern* is a state of the subproblem. The *pattern space* for a given subproblem is a state space containing all the different patterns connected to one another using the same operators that connect states in the original problem. A *pattern database* (PDB) stores the distance of each pattern to the goal pattern. Typically, a PDB is built by searching backwards, breadth-first, from the goal pattern until the whole pattern space is spanned. Given a state S in the original space, a heuristic value for S , $h(S)$, is computed using a PDB in two steps. First, S is mapped to a pattern. Then, this pattern is looked up in the PDB and the corresponding distance is returned as the value for $h(S)$.

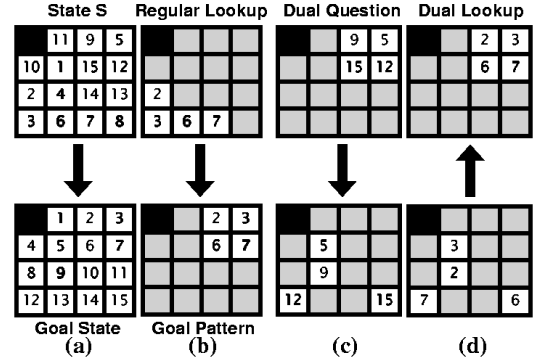


Figure 1: Example of regular and dual lookups

This paper is concerned with the first step, the mapping of a state S to a pattern that can be looked up in a given PDB. The standard mapping, called the regular PDB lookup in this paper, is illustrated in Figure 1(a) and (b) for the 15-puzzle. Patterns are created by ignoring all the tiles except for 2, 3, 6 and 7. Each pattern contains tiles 2, 3, 6 and 7 in a unique combination of positions. The resulting {2-3-6-7}-PDB contains a unique entry for each pattern with the distance from that pattern to the goal pattern (shown in the lower part of Figure 1(b)). Figure 1(b) depicts the regular lookup in this PDB for estimating a distance from a given state S to the goal (Figure 1(a)). State S is mapped to a 2-3-6-7 pattern by ignoring all the tiles other than 2, 3, 6 and 7. Then this pattern’s distance to the goal pattern is looked up in the PDB. To be specific, if the PDB is represented as a 4-dimensional array, PDB , with the array indexes being the locations of tiles 2, 3, 6, and 7 respectively, the regular lookup for state S is $PDB[8][12][13][14]$, because tile 2 is in location 8, tile 3 is in location 12, etc. The value retrieved by a regular PDB lookup for state S is a lower bound (and thus serves as an admissible heuristic) for the distance from S to the goal state in the original space.

It is common practice to exploit special properties of a state space to enable additional lookups to be done in a PDB. [Culberson and Schaeffer, 1998] describe several alternative lookups that can be made in the same PDB based on the physical symmetries of the 15-puzzle. For example, because of the symmetry about the main diagonal, the

PDB built for the goal pattern in Figure 1(b) can also be used to estimate the number of moves required to get tiles 8, 12, 9 and 13 from their current positions in state S to their goal locations. We simply reflect the tiles and their positions about the main diagonal and use the {2-3-6-7}-PDB for retrieving the symmetric values. The idea of reflecting the domain about the main diagonal for having another set of PDBs was also used by [Korf and Felner, 2002; Felner *et al.*, 2004] when solving the 15 and 24 tile puzzles with additive PDBs (see Figure 4 below).

Because all valid, alternative PDB lookups provide lower bounds on the distance from state S to goal, their maximum can be taken as the value for $h(S)$. Of course, there is a trade-off for doing this—each PDB lookup increases the time it takes to compute $h(S)$. Because additional lookups provide diminishing returns in terms of the reduction in the number of nodes generated, it is not always best to use all possible PDB lookups [Cullberson and Schaeffer, 1998]. A number of methods exist for reducing the time needed to compute $h(S)$ by making inferences about some of the values without actually looking them up in a PDB [Holte *et al.*, 2004].

The main contribution of this paper is a new, alternative PDB lookup that is based on properties other than physical symmetries. We call it the dual PDB lookup. The dual lookup is always admissible but, unlike previously considered PDB lookups, it can return values in the search that are inconsistent. Our second contribution is a simple but useful extension of the well-known pathmax method so that inconsistencies in heuristic values are propagated in both directions (child-to-parent, and parent-to-child) in the search tree. Our final contribution is that dual lookups, with bidirectional pathmax propagation, produce state-of-the-art performance for three standard test applications.

2 Dual Lookups in Pattern Databases

The states of many problems, such as the sliding tile puzzles, Rubik’s cube, Towers of Hanoi, etc., are defined by assigning objects (e.g. tiles, cubies, disks) to locations.

To explain the dual PDB lookup, consider again the {2-3-6-7}-PDB for the 15-puzzle defined above. The regular PDB lookup asks the question of what is the cost of getting tiles 2, 3, 6 and 7 from their current locations to their goal locations? In general, the regular lookup focuses on a fixed set of objects (the ones that define the patterns), and bases its lookup on their current locations, which vary from state to state.

In the dual PDB lookup the roles of locations and objects are switched. The dual PDB lookup focuses on a fixed set of locations – the goal locations of the objects in the pattern – and bases its lookup on the objects that occupy those positions, which vary from state to state. In particular, the dual PDB lookup of Figure 1 asks this question: what is the cost of moving the tiles that are currently in the goal location of tiles 2, 3, 6 and 7 to their home locations? For the state S in Figure 1(a), for example, the dual PDB lookup asks, what is the cost of getting tiles 9, 5, 15 and 12 from their current locations (the goal locations of tiles 2, 3, 6 and 7) to their goal locations? See Figure 1(c).

The dual question cannot be answered directly because a

PDB based on tiles 5, 9, 12 and 15 is not available. However, because costs in the 15-puzzle are symmetric and independent of the exact tiles involved, the dual question can be answered using the {2-3-6-7}-PDB. As shown in Figure 1(d), if we replace the names of tiles 5, 9, 12 and 15 with the name of the tile whose goal location they each currently occupy, the 5-9-12-15 goal pattern at the bottom of Figure 1(c) turns into a 2-3-6-7 pattern that can be looked up in the 2-3-6-7 PDB. In particular, the dual lookup would retrieve the value $PDB[9][5][15][12]$, because in state S tile 9 is in tile 2’s goal location, tile 5 is in tile 3’s goal location, etc.

Another example for dual lookups is the (N,K) -TopSpin puzzle which has N tokens arranged in a ring. Any set of K consecutive tokens can be reversed (rotated 180 degrees in the physical puzzle). Our encoding of this puzzle has N operators for each possible reversal/rotation. Figure 2(a) shows the goal state of the (9,4)-TopSpin puzzle. Figure 2(b) shows the result of reversing the tokens at locations 6-9, and Figure 2(c) shows the result of reversing the tokens in locations 4-7 of Figure 2(b).

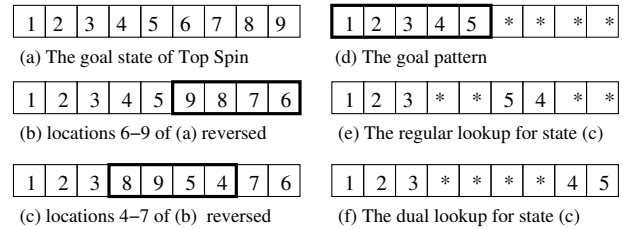


Figure 2: (9,4)-TopSpin states

Suppose patterns for TopSpin are defined by ignoring tokens 6-9. The resulting PDB provides distances to the goal pattern (shown in Figure 2(d)) from all reachable patterns. Consider Figure 2(c). The regular PDB lookup for this state is based on the pattern in Figure 2(e), obtained from (c) by mapping tokens 6-9 to “*”. Since tokens 1-5 are in locations 1, 2, 3, 7 and 6, respectively, the regular lookup would be $PDB[1][2][3][7][6]$. For the dual lookup, we note that the goal locations of tiles 1-5 (as in state (c)) are occupied by tokens 1, 2, 3, 8, 9. The dual lookup, $PDB[1][2][3][8][9]$, gives the cost of moving these tiles to their goal locations. This lookup corresponds to the pattern shown in Figure 2(f).

Dual PDB lookups are possible where there is symmetry between objects and locations of the domain in the sense that each object is located in one location and each location occupies only one object. Example domains where this is true are the TopSpin puzzle (Section 4) and Rubik’s cube (Section 5). A counter example is the towers of Hanoi where there is no symmetry between locations and objects. The 15 puzzle domain is more difficult because the blank has to be handled differently from the other tiles. Nevertheless, even in this seemingly asymmetric domain, the PDBs can be constructed in such a way as to enable the dual symmetry (see Section 6).

3 Bidirectional Pathmax ($bpmx$)

Regular PDB lookups produce consistent heuristic values during search [Holte *et al.*, 1996]. Dual lookups are admissi-

ble, but not necessarily consistent. For our TopSpin example, let h_d be the result of a dual lookup on this PDB. In Figure 2(b), tokens 1-5 are in their goal locations and therefore $h_d(b) = 0$. Figure 2(c) is obtained from (b) by a single move. However, the dual lookup for this state uses the pattern in Figure 2(f), which is two moves away from the goal pattern. Therefore, $h_d(c) = 2$, which is inconsistent with $h_d(b) = 0$ since (c) is only one move from (b).

[Mero, 1984] described two methods of propagating heuristic values between a state and its children to take advantage of inconsistencies. Let P be any state, $\{C_i\}$ the children of P , and $dist(P, C_i)$ the cost of reaching C_i from P . Mero’s first propagation method, now known as pathmax, propagates heuristic values from P to its children: $h(P) - dist(P, C_i)$ is a lower bound on $dist(C_i, Goal)$ and therefore can be used instead of $h(C_i)$ if it is larger. Mero’s second method propagates heuristic values upwards, from the children of P to P . The path from P to the goal must pass through a child of P ¹. Thus, $\min(h(C_i) + dist(P, C_i))$ is a lower bound on $dist(P, Goal)$ and can be used instead of $h(P)$ if it is larger.

Previous work failed to notice that when operators are invertible (and costs symmetric), pathmax allows values to be propagated in both directions and is also applicable in undirected graphs. This might produce a more useful children-to-parent propagation than Mero’s second method.

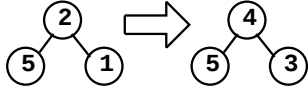


Figure 3: Propagation of values with inconsistent heuristics

The bidirectional pathmax method (*bpmx*) is illustrated in Figure 3. The left side of the figure shows the (inconsistent) heuristic values for a node and its two children. When the left child is generated, its heuristic ($h = 5$) can propagate up to the parent and then down again to the right child. To preserve admissibility, each propagation reduces h by the cost of traversing that path (1 in this example). This results in $h = 4$ for the root and $h = 3$ for the right child. When using IDA*, this bidirectional propagation can cause many nodes to be pruned that would otherwise be expanded. For example, suppose the current IDA* threshold is 2. Without the propagation of h from the left child, both the root node ($f = g + h = 0 + 2 = 2$) and the right child ($f = g + h = 1 + 1 = 2$) would be expanded. Using the propagation just described, the left child will improve the parent’s h value to 4, resulting in a cutoff without even generating the right child.

4 TopSpin

We have implemented the above ideas on the (17,4)-TopSpin puzzle. This domain has $17! = 3.55 \times 10^{14}$ states. We generated a PDB of the leftmost 9 tokens, a pattern space of $17 \times 16 \dots \times 9 = 8.82 \times 10^9$. Since this puzzle is cyclic, we can assume that token number 1 is always in the leftmost

¹This is only true in a directed graph. In an undirected graph the shortest path from P to the goal might pass through the parent of P .

Heuristic	Nodes	Time	<i>bpmx</i>
1r+0d	40,019,429 / 1.0	67.76 / 1.0	0
0r+1d	7,618,805 / 5.3	15.72 / 4.3	0
0r+1d+c	1,397,614 / 28.6	2.93 / 23.1	194,135
2r+0d	6,981,027 / 1.0	21.90 / 1.0	0
1r+1d+c	492,686 / 14.2	1.47 / 14.9	34,725
0r+2d+c	372,414 / 18.7	1.46 / 15.0	29,302
4r+0d	651,080 / 1.0	4.05 / 1.0	0
0r+4d+c	143,177 / 4.6	1.09 / 3.7	4,638
8r+0d	116,208 / 1.0	1.48 / 1.0	0
4r+4d+c	82,606 / 1.4	0.94 / 1.6	1,155
0r+8d+c	74,610 / 1.6	1.12 / 1.3	915
17r+17d+c	27,575	1.34	29

Table 1: Solutions to the (17,4) TopSpin puzzle

position. Thus, for implementation, both numbers above can be divided by 17. Since all the values in the PDB were smaller than 16, each entry needs 4 bits and the PDB needs 259MB.

A PDB of 9 tokens has actually 17 different ways of choosing which tokens are included. A PDB of tokens $[1 \dots 9]$ can also be used as a PDB of $[2 \dots 10]$, $[3 \dots 11]$, etc, with the appropriate mapping of tokens. Thus, a single PDB gives us 17 regular heuristics and 17 dual heuristics. The search was done using IDA*. Many duplicate states can be avoided by forcing two unrelated operators to be applied successively in only one order. For example, the operator that reverses locations (1, 2, 3, 4) is not related to the operator that reverses locations (11, 12, 13, 14). This operator ordering decreased the number of generated nodes by an order of magnitude.

Table 1 presents data for different heuristics and combinations. Each value in the table is an average over a set of 1,000 random permutations. The average solution length for this test set is 14.8. All the experiments reported in this paper were run on a 1.7GHz Pentium 4 PC with 1GB of memory.

The table columns give the number of regular (r) and dual (d) lookups used, the presence of *bpmx* cutoff (c), the number of generated nodes (nodes and ratio to the same number of lookups with only- r result), the time (seconds and ratio), and the number of times that the *bpmx* cutoff occurred.

The results show that a single dual lookup outperforms a regular lookup by a factor of 5.3 in generated nodes (4.3 in running time). This is because the dual lookup frequently “jumps” to different areas of the PDB and has a larger diversity of different heuristic values as will be further explained in Section 7. The *bpmx* cutoff further improves this to a factor of 28.6 in nodes (23.1 in time). The *bpmx* cutoff was applicable 194,135 times, pruning 6,221,191 nodes, and averaging 32 nodes per instance of cutoff. These performance gains are achieved using no additional storage, just by looking at one PDB in different ways.

The table also shows the results of using two lookups and taking their maximum. Compared to two regular PDB lookups, two dual lookups (with *bpmx*) give an 18.7-fold reduction in nodes. This result is better than combining one regular and one dual lookup. When four lookups are used, again the dual-only lookup solution is better than the regular-only lookup solution. We see diminishing returns when more

Heuristic	Nodes	Time
1r+0d	90,930,662 / 1.0	28.18 / 1.0
0r+1d	19,653,386 / 4.6	7.38 / 3.8
0r+1d+c	8,315,116 / 10.9	3.24 / 8.7
2r+0d	12,649,720 / 1.0	4.68 / 1.0
1r+1d+c	2,997,539 / 4.2	1.34 / 3.5
0r+2d+c	5,290,272 / 2.4	2.32 / 2.0
4r+0d	1,053,522 / 1.0	0.64 / 1.0
2r+2d+c	1,667,320 / 0.6	0.90 / 0.7
0r+4d+c	1,053,759 / 1.0	0.67 / 0.9
4r+4d+c	615, 563	0.51
24r+24d+c	362,927	0.90

Table 2: Solutions to Rubik’s cube from one 7-edges PDB

and more lookups are done. Many lookups provide a diversity of heuristic values anyway. Therefore, the improvement factor of any additional lookup (dual or regular) decreases.

Note that our fastest implementation uses 4 regular and 4 dual lookups took 0.94 seconds – 72 times faster than a single regular lookup. Using 17 regular and 17 dual lookups produces the smallest search tree of only 27,575 generated nodes – a factor of 1,451 over a single regular lookup.

We used our fastest implementation (4r+4d+c) to solve larger versions of TopSpin. TopSpin (19, 4) is 18×17 times larger than the (17, 4) variant. We solved 20 instances for TopSpin (19, 4). The average solution length is 17.3, 73 million nodes are generated and the search takes 172 seconds. For TopSpin (20, 4) (19 times larger), we tested on five problems with an average solution length of 20. These problems averaged 2.9 billion nodes and took 7,716 seconds.

5 Rubik’s Cube

[Korf, 1997] solved the $3 \times 3 \times 3$ Rubik’s cube, containing roughly 4×10^{19} different reachable states. There are 20 movable sub-cubes, or *cubies*. They can be divided into eight *corner cubies*, with three faces each, and twelve *edge cubies*, with two faces each. As a first experiment, we built a 7-edge-cubies PDB, the largest that can be stored in 1GB of memory. There are 510,935,040 possible permutations of the 7 edge cubies. At 4 bits per entry, 255MB are needed for this PDB. As with TopSpin, symmetries in the domain mean that there are multiple possible regular and dual lookups.

Table 2 presents results for a number of possible combinations of this setting (the table headings have the same meaning as in Table 1). The start states used were “easy”—100 different states obtained by 14 random moves from the goal configuration (average solution length of 10.66).

The results are similar to the TopSpin experience, albeit slightly lower. Again, the dual lookup and *bpmx* cutoffs result in large reductions in the search effort. However, for this puzzle by the time you hit four lookups, diminishing returns has taken over and the advantage of the dual has dissipated. Our best implementations reduced the number of nodes generated (24r+24d+c) by a factor of 250, and the time (4r+4d+c) by a factor of 55. All this was possible with just one 7-edge-cubies PDB stored in memory.

[Korf, 1997]’s original 1997 Rubik’s cube experiments were repeated, this time with dual PDB lookups. Korf used three PDBs for this domain: one PDB for the 8 corner cubies and two PDBs for two sets of 6 edge cubies. Since a legal move in this domain moves 8 cubies, the only way to combine these 3 PDBs is by taking their maximum. Note that there are 8 corner cubies and all 8 are used by the 8-corner PDB. Thus, performing a dual lookup for this particular PDB is irrelevant. Here, the entire space of corner cubies is in the database and both lookups give the same result.

Results for the same set of 10 random instances used in [Korf, 1997] were obtained. The results for Korf’s set of $8 + 6 + 6$ PDBs were improved by a modest amount by adding the dual lookups for both 6-edge PDBs (from 353 billion nodes to 253 billion). Increasing the edges PDB from 6 to 7 cubies and using a $8 + 7r + 7r + 7d + 7d$ setting reduced the search to 54 billion nodes – an improvement of a factor of 6.4 over Korf’s initial setting. The improvements of adding dual lookups for the 6- and 7-edges PDBs are modest since most of the time the 8-corner PDB has the maximum value; this PDB is larger and contains more cubies than the 6- and 7-edge PDBs. We measured these rates over 10 million random instances. For the $8 + 6r + 6r + 6d + 6d$ setting, the 8-corner PDB had the maximum value in 73.5% of these cases while one of the lookups in the 6-edges cubies was the maximum in only 7.3% of the cases (the rest of the cases are a tie). These numbers were changed to 40.8% and 21.3% respectively for the $8 + 7r + 7r + 7d + 7d$ setting.

6 Sliding-tile Puzzles

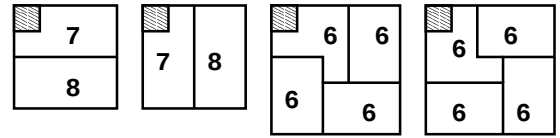


Figure 4: Partitionings and reflections of the tile puzzles

We have also implemented the new ideas for the sliding-tile puzzles. For the 15-puzzle, we used the same 7 – 8 partitioning from [Korf and Felner, 2002] (Figure 4). These PDBs are constructed so that their heuristic values can be *added* [Felner *et al.*, 2004] together and preserve admissibility. The PDBs can be reflected across the diagonal, obtaining another set of 7 – 8 heuristics (also shown in Figure 4).

Dual lookups for this domain are not obvious. While there are 16 similar locations, the 16 tiles are not similar as there are 15 real tiles and one blank. Given the location of the blank, then a horizontal line (or a symmetric vertical line) across the middle of the puzzle divides it into two regions of 8 locations. One region (call it *A*) has 8 locations which are occupied by 8 real tiles, and another region of 8 locations (*B*) which are occupied by 7 real tiles and the blank. Performing a dual lookup for region *A* in the 8-tile PDB is identical to what was done in TopSpin and Rubik’s cube.

A dual lookup in *B* is complicated by the blank. Figure 5 shows four possible blank locations in *B* for a horizontal partitioning. Other locations of the blank (as well as a vertical

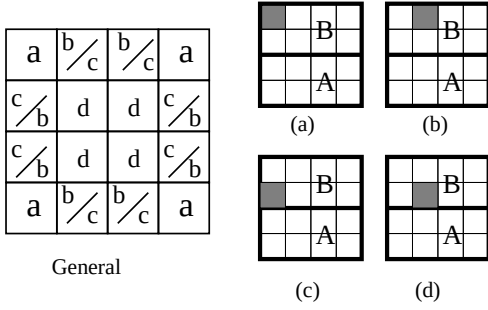


Figure 5: Four different (dual) 7-tile pattern databases

partitioning into two regions) can be easily transformed into one of these 4 configurations (e.g., by reflection). Consider the regular 7-tile PDB of tiles $(1 \dots 7)$ (which corresponds to the 7 real tiles of region B of Figure 5(a)). This PDB answers the question of how to move these real tiles from any possible configuration to their goal configuration. The dual lookup should answer the question of how many moves it takes to distribute the tiles that are currently located in locations $(1 \dots 7)$ to their goal configuration. If these tiles are all real tiles (and the blank is in the corner) then getting this data from the regular PDB is as before. However, when the blank is not in the corner, (the other cases of Figure 5) one of the tiles occupying locations $(1 \dots 7)$ is the blank. Therefore, the answer from a dual lookup of the regular 7-tile PDB will also count moves of the blank while the real tile in the corner will be ignored. This might lose admissibility.

There are two possible ways to solve this problem. The first is to artificially move the blank tile to the nearest corner. In effect, this means reducing the PDB value by one or two (to account for the extra blank moves) to preserve admissibility. Now we can use the regular 7-tile PDB for a dual lookup, at the cost of a weaker heuristic (and, hence, additional search).

A better idea is to add three more 7-tile PDBs for a total of four—one for each of the cases in Figure 5. For each blank scenario we build a regular 7-tile PDB assuming that the blank is located in the relevant location. For example, for region B of Figure 5(b) we build a PDB for the tiles $\{0, 2, 3, 4, 5, 6, 7\}$ and assume that tile 1 is the blank tile. This PDB cannot be consulted as a regular PDB since we assume that tile 1 is the blank. However, for any partitioning where region B corresponds to this case, we can perform a dual lookup in this PDB and retrieve the correct value for the tiles that are currently located in locations $\{0, 2, 3, 4, 5, 6, 7\}$ (or their reflections). Similarly for the other blank locations.

The frame on the left of Figure 5 indicates the relevant PDB for the dual lookup of each possible blank location. In those locations where two PDBs are given, then the right label indicates the PDB to use for a horizontal partition while the left corresponds to a vertical partition. The amount of memory needed is 519KB for the 8-tile PDB and 57.5KB for a 7-tile PDB. Thus the total memory needs (the 8-tile and 4 7-tile PDBs) is 749KB. The three extra PDBs needed to handle all the dual cases correctly represent a small increase of memory.

Table 3 presents results of the different heuristics averaged over the same 1000 instances used in [Korf and Felner, 2002].

Heuristic	Av. H	Nodes	Time
1r	44.75	136,289 / 1.0	0.081 / 1.0
1r+1r*	45.63	36,710 / 3.7	0.034 / 2.4
1d	44.39	278,820 / 0.5	0.157 / 0.5
1d+c	44.39	247,299 / 0.6	0.139 / 0.6
1d+1d*+c	44.40	65,349 / 2.1	0.076 / 1.1
1r+1r*+1d+1d*+c	46.12	18,601 / 7.3	0.022 / 3.7

Table 3: Results for the 15 puzzle

PDBs	Nodes	Time	Memory
1r+1r*	16,413,254,279	32,826	244,140
1r+1r*+1d+1d*+c	6,877,105,604	22,955	976,562

Table 4: 24 puzzle results

The average solution for these instances is 52.55. The first column indicates the heuristic used, with ‘r*’ and ‘d*’ representing the reflected regular and dual PDB lookups. The first row presents the results when only the regular PDB is used, while the second row took the maximum of the regular and reflected PDBs. Note that these two rows are the same results obtained by [Korf and Felner, 2002] but on our current machine. The next three rows present results for different versions of the dual lookup. Note that for this domain the *bpm.x* cutoff yielded a reduction in nodes of only 10% for a single dual lookup. Finally, the last row presents the maximum over the four PDB combinations. Using dual lookups reduces the number of generated nodes by more than a factor of 2 and eliminated one third of the execution time compared to the best results of [Korf and Felner, 2002] (line 2 of Table 3). To our knowledge using the four regular/dual normal/reflected PDB lookups gives the best existing heuristic for this puzzle. Of historical note is that the number of generated nodes is now nearly 30,000 times smaller than when IDA* first solved the 15-puzzle using only Manhattan distance [Korf, 1985].

Similar experiments were performed using the 24-puzzle. The original 6-6-6-6 partitioning from [Korf and Felner, 2002] (Figure 4) needed storage for only two 6-tile PDBs since all the 3×2 rectangles are symmetric. As before, we need additional PDBs to handle the blank. We use 8 6-tile PDBs: one for all the 3×2 rectangles and their duals, but we need 7 6-tile PDBs for the irregular shape in the top left corner (see Figure 4). Each 6-tile PDB needs 122MB and our new system needs 8 times as much memory.

In [Korf and Felner, 2002] 50 random instances of the 24-puzzle were solved. Table 4 presents the average results over the 25 easiest problems of that set (the 25 with the fewest nodes generated). The average solution length for this set is 96.2 moves. Using both regular and dual lookups and their reflections reduces the number of generated nodes by a factor of 2.38 and the time by a factor of 1.43 when compared to the results of [Korf and Felner, 2002] (first line of Table 4).

7 Discussion

Dual PDB lookups double the number of possible PDB lookups. They are effective for all the domains studied but

perform differently on the various domains. There are two phenomena that need explanation. First, the *bpmx* cutoff was much better in TopSpin and Rubik's cube (a factor of up to 5) than in the tile puzzles (only 10%). Second, in TopSpin and Rubik's cube a single dual PDB lookup (even without the *bpmx* cutoff) was better than a single regular PDB by a factor of up to 5.3, while in the tile puzzles it was worse.

To explain this we define the *inconsistency rate* of a heuristic as the average difference between the heuristics of an arbitrary pair of neighboring states. For a consistent heuristic, this rate must not exceed 1 (assuming a uniform edge cost of 1). We measured this rate for 10 million pairs of states for each domain. For the dual lookups the rates were 2.01 for TopSpin, 1.74 for Rubik's cube (7-edge PDB) and only 1.16 for the 15-puzzle.

Values in PDBs are locally correlated. However, in TopSpin 4 tokens change their locations at each move (8 cubes for Rubik's cube). The identity of the tokens queried by the dual lookup can dramatically change between consecutive steps. This is a dramatic "jump" to different place in the PDB. There is no locality of values with such a jump, meaning there is higher chance of getting a radically different heuristic value (better or worse). This causes IDA* to generate less nodes as mistakes of heuristics (such as low estimations) are being corrected fast. This also increases the rate of inconsistency and thus the *bpmx* cutoff is performed frequently.

In the 15-puzzle however, every operator only moves one tile. This means that in two consecutive PDB lookups (regular or dual) most of the indices stay the same – and you get similar values. Furthermore, in most cases of the 15-puzzle, the blank and the tile that exchange their locations belong to the same set of 8 locations indicated by region *B* above. Thus the identity of the 7 and 8 tiles for the dual lookups remain the same in consecutive steps. Only when the blank crosses the partition line (such as moving from position 4 to position 8 in Figure 5c) then the 7-tile set and the 8-tile sets for the dual lookups are changed and there is some chance for a dramatic jump and for inconsistency. However, even in this case the inconsistency rate is low because we *add* heuristics of two different subproblems which together report values for all the 15 tiles. Thus, the identity of the tiles being reported is not changed. Therefore, the diversity and inconsistency rate is low and the *bpmx* cutoff is not performed frequently.

We can conclude that dual heuristics and *bpmx* cutoff are more effective in domains where each operator changes larger parts of the state and the identity of objects being reported in consecutive steps is different.

Another interesting phenomenon is the fact that unlike the other domains, in the 15 puzzle the pure dual PDB lookup was worse than the pure regular lookup and generated almost twice as many nodes. The reason for this is again the location of the blank. Note that while the regular PDB lookup always consults the 8-tile PDB and the 7-tile PDB labeled *a* in Figure 5, the dual PDB might also consult the other 7-tile PDBs (labeled *b*, *c* and *d*). This is because the regular lookup always aims for a region *B* configuration such that the blank is located in a corner (the goal state) while the dual lookup needs to consider other possibilities for region *B*. It turns out that getting the blank to the corner is a harder task and needs

more moves. While the average value over all the entries of the PDB labeled *a* in Figure 5 is 20.91, the average values of the PDBs labeled *b*, *c* and *d* are 20.81, 20.31 and 20.53 respectively. Thus, we expect that the values obtained by the dual lookups will be smaller than those obtained by the regular PDB. Historically, the goal location of the blank is in the corner. However, if we set a goal state such that the blank is in location 4 (as in figure 5.c) then the regular heuristic will always look in the weakest PDB while the dual heuristic will consult the other PDBs as well. We have made such experiments and indeed, the pure dual PDB lookup generated nearly 40% less nodes than the pure regular PDB.

It is important to note that the dual lookups for the tile puzzles are of great importance. TopSpin and Rubik's cube have many symmetries and thus enable many possible regular PDB lookups. In the tile puzzles, however, there is only one symmetry available for the state-of-the-art additive heuristic—the reflection about the main diagonal. Thus, the dual idea doubles the number of possible lookups and achieved a speedup of a factor of 2 over the previous benchmarks.

8 Summary and Conclusions

We presented a new way of using pattern databases. We show that the dual lookup heuristic values are inconsistent, which allows additional opportunities for achieving cut-offs in the search. Results on TopSpin, Rubik's cube, and the sliding tile puzzles confirm the advantages of this new heuristic. To the best of our knowledge we have the best published optimal solvers for all three domains.

Inconsistent heuristics have a negative reputation; something to be avoided. We showed that there is nothing to fear with inconsistent heuristics and there can be real benefits.

References

- [Culberson and Schaeffer, 1994] J. C. Culberson and J. Schaeffer. Efficiently searching the 15-puzzle. Technical report, Department of Computer Science, University of Alberta, 1994.
- [Cullberson and Schaeffer, 1998] J. C. Culberson and J. Schaeffer. Pattern databases. *Computational Intelligence*, 14(3):318–334, 1998.
- [Edelkamp, 2001] S. Edelkamp. Planning with pattern databases. *ECP-01*, 2001.
- [Felner et al., 2004] A. Felner, R. E. Korf, and S. Hanan. Additive pattern database heuristics. *Journal of Artificial Intelligence Research (JAIR)*, 22:279–318, 2004.
- [Holte et al., 1996] R. C. Holte, M. B. Perez, R. M. Zimmer, and A. J. MacDonald. Hierarchical A*: Searching abstraction hierarchies efficiently. *AAAI*, pages 530–535, 1996.
- [Holte et al., 2004] R. Holte, J. Newton, A. Felner, and D. Furcy. Multiple pattern databases. *ICAPS*, pages 122–131, 2004.
- [Korf and Felner, 2002] R. E. Korf and A. Felner. Disjoint pattern database heuristics. *Artificial Intelligence*, 134:9–22, 2002.
- [Korf, 1985] R. E. Korf. Depth-first iterative-deepening: An optimal admissible tree search. *AIJ*, 27:97–109, 1985.
- [Korf, 1997] R. E. Korf. Finding optimal solutions to Rubik's Cube using pattern databases. *AAAI*, pages 700–705, 1997.
- [Mero, 1984] L. Mero. A heuristic search algorithm with modifiable estimate. *Artificial Intelligence*, 23:13–27, 1984.

The Rules of Constraint Modelling

Alan M. Frisch and Chris Jefferson and Bernadette Martínez Hernández

Artificial Intelligence Group, Dept. of Computer Science, Univ. of York, UK

{frisch,caj,berna}@cs.york.ac.uk

Ian Miguel

School of Computer Science, Univ. of St Andrews, UK

ianm@dcs.st-and.ac.uk

Abstract

Many and diverse combinatorial problems have been solved successfully using finite-domain constraint programming. However, to apply constraint programming to a particular domain, the problem must first be *modelled* as a constraint satisfaction or optimisation problem. Since constraints provide a rich language, typically many alternative models exist. Formulating a *good* model therefore requires a great deal of expertise. This paper describes CONJURE, a system that *refines* a specification of a problem in the abstract constraint specification language ESSENCE into a set of alternative constraint models. Refinement is *compositional*: alternative constraint models are generated by composing refinements of the components of the specification. Experimental results demonstrate that CONJURE is able to generate a variety of models for practical problems from their ESSENCE specifications.

1 Introduction

To employ finite-domain constraint programming technology to solve a problem, the problem first must be characterised, or *modelled*, by a set of constraints on decision variables that solutions must satisfy. Modelling can be difficult and requires expertise, thus limiting widespread use of constraint technology. The vast majority of research on constraint modelling presents alternative models to a particular problem and evaluates these alternatives through analysis and/or experiment. The process by which the alternative models are generated is rarely, if ever, discussed. Each constraint programmer must learn the art of modelling by forming generalisations from these studies.

We show that a set of rules can formalise the generation of alternative models. Doing this requires a language in which to express the abstract problem structure of which models are a function. If all modelling choices are to be open to the rules, the language must have a level of abstraction above that at which modelling decisions are made. We have designed such a language and call it ESSENCE.

The rules are embedded in the CONJURE system, which, given an ESSENCE specification, generates models of the type

supported by existing constraint solvers. Hence, we refer to our rules as *refinement rules*. Our current focus is on generating a set of correct models that includes those that a human expert would generate. Future work will focus on generating only *good* models.

2 Challenges and Contributions

A central task, arguably *the* central task, of modelling most combinatorial problems is choosing a representation of complex decision variables. Current finite-domain constraint solvers provide decision variables whose domains contain atomic elements, which we call atomic variables.¹ Yet, combinatorial problems often require finding a more complex combinatorial structure. For example, the Social Golfers Problem (SGP, see [9] problem 10) requires partitioning a set G of golfers in each week of play. Thus, the goal is to find a multiset of partitions of G , i.e. a multiset of sets of sets of G . Modelling the SGP requires deciding how to represent this complex decision variable as a constrained collection of atomic variables.

In concert with choosing a representation of complex decision variables is the task of representing the constraints of the problem. In its natural form, a combinatorial problem imposes constraints on the combinatorial structure that is sought. These constraints must be “translated” so that they are imposed on the representation of the decision variables.

This paper shows how these two central tasks of modelling can be formalised and automated, and in doing so reports two principal contributions. First, we have designed a language, called ESSENCE, that enables combinatorial problems to be stated at a high level of abstraction. This level of abstraction is a consequence of three features: (1) The language supports a wide range of types (including sets, multisets, relations, functions, partitions) and decision variables can be of these types. (2) All types can be nested to arbitrary depth; for example, a decision variable can be of type set, set of sets, set of set of sets, and so forth. (3) Constraints can contain quantifiers that range over decision variables. For example, if a decision variable X is of type set of sets, a constraint can

¹Some finite-domain constraint solvers support variables whose domain elements are finite sets of atomic elements. Though all aspects of our work have considered this, for the sake of simplicity this paper pretends such variables do not exist.

be of the form $\forall x \in X. \phi$. The ESSENCE language is outlined in Sec. 3, though only in enough detail to enable the presentation of our second principal contribution.

Our second major contribution is the formulation and automation of a set of rules that can refine constraints on complex variables in an ESSENCE specification into constraints on atomic variables, which is the level of abstraction provided by existing constraint languages and toolkits. Our formal rules are presented in Sec. 4.

An attempt to formulate such a set of refinement rules confronts two primary difficulties and many secondary ones. Before proceeding it is worth considering the primary ones.

The first difficulty arises because expressions, particularly decision variables of non-atomic type, can usually, if not always, be refined in multiple ways. Furthermore, the refinement of an operator depends on how its operands are refined. For example, in refining the constraint $S_1 = S_2$, where S_1 and S_2 are atomic set variables, the treatment of equality is different for every combination of ways that S_1 and S_2 can be refined. Indeed, it is possible that an operator is inapplicable to certain refinements of its operands.

The second major difficulty arises from arbitrary nesting of types. Refining an operator that is applied to expressions whose types are nested provides the biggest challenge. Consider refining the constraint $A = B$, where A and B are decision variables of some type τ that is nested arbitrarily deep. The generated constraint must involve all components of both A and B . Since there is no bound on the nesting of τ , this complex constraint would have to be generated through recursive rule applications, none of which can look arbitrarily deep into the nesting of τ . Furthermore, we wish to produce refinements in which A and B do not have the same kind of representation. An especially tricky case is when a quantifier ranges over a decision variable of a nested type, such as $\forall x \subseteq A \phi$, where A is as above.

Modelling in constraints involves more than just representing decision variables and problem constraints. Constraint models often contain many symmetries, often enormous numbers of them, which result in redundancies in the search space. Expert modellers are able to identify such symmetries and break them, either by introducing symmetry-breaking constraints or using a symmetry-aware search method. It has been argued that detecting symmetries in a model is as hard as graph isomorphism. We maintain that symmetries enter a model from two sources: either a symmetry is inherent to the combinatorial problem or it is introduced by the modelling process. An automated modelling system ought to identify the symmetry that it introduces into a model. Sec. 5 explains how this can be integrated into our architecture for model generation.

Another technique used by expert modellers is to represent a complex decision variable with multiple representations simultaneously and impose channelling constraints to keep the representations consistent with each other [4; 8; 11]. This sometimes yields more propagation, and therefore reduced search, than a single representation. As will be seen, the rules of CONJURE generate models with multiple representations. Sec. 6 explains how our refinement rules can generate the information that is needed to automatically generate

Given n , put n integer ticks on a ruler of size m such that all inter-tick distances are unique. Minimise m .	
given	n : int
where	$n \geq 0$
letting	$bound$ be 2^n
find	$Ticks$: set (size n) of $0..bound$
minimising	$\max(Ticks)$
such that	$\forall \{i, j\} \subseteq Ticks. \forall \{k, l\} \subseteq Ticks.$ $\{i, j\} \neq \{k, l\} \rightarrow i - j \neq k - l $

Figure 1: ESSENCE specification of the Golomb Ruler problem.

the appropriate channelling constraints.

We conjecture that the good models of a problem can be generated automatically by formulating the problem in ESSENCE and then refining the specification using rules of the kind presented in this paper. We have implemented a program, called CONJURE, that currently refines a subset of ESSENCE, called mini-ESSENCE. This gives us a platform with which we have experimentally tested our conjecture, as reported in in Sec. 7.

3 An Introduction to ESSENCE

This section briefly introduces the abstract specification language ESSENCE. It is an evolving and growing language; version 1.0 is used within this paper. A full specification of the language can be found at www.cs.york.ac.uk/aig/constraints/AutoModel/.

Let us begin by considering the specification of the Golomb Ruler Problem (GRP, problem 6 at www.csplib.org), shown in Fig. 1). A specification is a list of statements, of which there are seven kinds, signalled by the keywords *given*, *where*, *letting*, *find*, *maximising*, *minimising* and *such that*. Statements are composed into specifications according to the regular expression

$(\text{given} | \text{letting} | \text{where})^* \text{find}^+$
 $[\text{minimising} | \text{maximising}] (\text{such that})^*$

Identifiers in CONJURE come in four *categories*: constant, parameter, quantified variable, and decision variable. “Letting” statements declare constant symbols and give their values. “Given” statements declare the problem’s parameters; the values of the parameters are provided to specify the instance of the problem class. Parameter values are not part of the problem specification; as in other modelling languages, they are provided elsewhere. “Where” statements impose restrictions on the parameter values; only parameter values meeting the restrictions specify a problem instance. “Find” statements declare decision variables. A “minimising” or “maximising” statement gives the objective function, if any. Finally, “such that” statements give the problem’s constraints.

The GRP specification begins by declaring n to be a parameter, restricting it to be non-negative and declaring $bound$ to be a constant. Since n is used in the declaration of $bound$, the declaration of $bound$ must come after the declaration of n . Every symbol must be defined before it is used. This restriction prevents cyclical definitions and means that decision variables cannot be used in the definitions of constants and parameters.

The specification language is strongly typed and every expression and subexpression has a type. The type of the decision variable *Ticks* indicates that the goal of the problem is to find a set containing n elements, each of which is an integer in the range 0 to *bound*. The types supported by ESSENCE include the atomic types *int* (integer), *bool* (Boolean) and *l..u* (an integer range type), where *l* and *u* are integer expressions. ESSENCE also provides enumerated types and a new and very useful atomic type: *type* (size α), a type of α unnamed elements.

ESSENCE is the first constraint language to support fully-compositional type constructors. So, for example, a decision variable may be of type integer, set of integer, set of set of integer, and so forth. If τ is a type, α is an integer expression, and $I_1, \dots, I_n$ are expressions of type range then the following are types that appear in this paper:

<i>set</i> (size α) of τ ,	<i>set</i> (maxsize α) of τ	two kinds of set
<i>mset</i> (size α) of τ ,	<i>mset</i> (maxsize α) of τ	two kinds of multiset
<i>matrix</i> [indexed by $I_1, \dots, I_n$] of τ		matrix

Also among the types of ESSENCE are relations between any two types, partial and total functions from any type to any type, partitions of a set of any type and permutations of any type.

Constraints in this specification language are built from the parameters, constants and decision variables using operators commonly found in mathematics and in other constraint specification languages. The language also includes variable binders such as $\forall x$, $\exists x$ and $\sum_x$, where x can range over any specified finite type (e.g., integer range but not integer). The constraint in the GRP can be paraphrased as “For any unordered pair $\{i, j\}$ of ticks and any unordered pair $\{k, l\}$ of ticks, if the two pairs are different then the distance between i and j is not the same as the distance between k and l .” To clarify the notation, the expression $\{i, j\} \subseteq \textit{Ticks}$ means that two distinct elements are drawn from *Ticks* and, without loss of generality, one is called i and the other is called j .

Now consider the specification of the Sonet problem shown in Fig. 2. Notice that *Nodes* is declared to be a range. A subtle point is that the third line of the specification is declaring two parameters. When the *demand* parameter is instantiated to a particular set of sets, the size of the outer set will be known. Hence, the value of m is given indirectly. This declaration also requires the inner sets to have cardinality two. The goal is to find a multiset (representing the rings), each element of which is a set of *Nodes* (representing the nodes on that ring). The objective is to minimise the sum of the number of nodes installed on each ring. The constraint ensures that any pair of nodes that must communicate are installed on a common ring.

The Target Language: ESSENCE'

ESSENCE specifications are refined into a target language called ESSENCE', which is a subset of ESSENCE with a level of abstraction similar to that of existing constraint languages, OPL [19] being the closest. The only types ESSENCE' has are integers, integer ranges, Booleans and matrices; it does not have enumerated types or types of unnamed elements. Binders, such as quantifiers and summations, range only over integer ranges. From this generic constraint language, it is

A Sonet communication network comprises a number of rings, each joining a number of nodes. A node is installed on a ring using an ADM and there is a capacity bound on the number of nodes that can be installed on a ring. Each node can be installed on more than one ring. Communication can be routed between a pair of nodes only if both are installed on a common ring. Given the capacity bound and a specification of which pairs of nodes must communicate, allocate a set of nodes to each ring so that the given communication demands are met. The objective is to minimise the number of ADMs used. (This is a common simplification of the full Sonet problem, as described in [8])

```

given      nrings:int, nnodes:int, capacity:int
where      nrings ≥ 1, nnodes ≥ 1, capacity ≥ 1
letting    Nodes be 1..nnodes
given      demand:set (size m) of set (size 2) of Nodes
find       rings: mset (size nrings) of set (maxsize capacity) of Nodes
minimising  $\sum_{r \in \text{rings}} |r|$ 
such that   $\forall \text{pair} \in \text{demand}. \exists r \in \text{rings}. \text{pair} \subseteq r$ 

```

Figure 2: ESSENCE specification of the Sonet problem.

a short step to an established constraint language, such as OPL, Solver, or Eclipse. To perform this step, we are developing a suite of back-end translators. In future, we also intend to translate to more restricted languages, such as SAT and Pseudo-Boolean formulations.

4 The Architecture of CONJURE

This section discusses the refinement rules CONJURE uses to refine an ESSENCE expression into a set of ESSENCE' expressions. For concision we give only a small subset of all the refinement rules²; the remainder follow a similar pattern.

Refining a Simple ESSENCE Specification

Refinement of an ESSENCE specification begins by refining each constraint and the objective function in turn. Let us begin by considering the single-constraint MicroSonet₁ specification (Fig. 3), where the goal is to fill two rings with the same nodes.

A key consideration in refining MicroSonet₁ is the representation of each *ring*, each of which is a fixed-size set. This paper uses two representations of fixed-size sets. The *explicit representation* is a one-dimensional matrix of *Nodes* indexed by 1..*capacity*. Each element of the matrix corresponds to an element of the set. To represent the set properly, the elements of the matrix are constrained to take distinct values. The *occurrence representation* is a one-dimensional Boolean matrix indexed by *Nodes*, where a *true* entry indicates that the corresponding node is in the set. To represent the set properly, the number of *true* entries must be equal to *capacity* (for counting we treat *true/false* as 1/0). Fig. 3 gives ESSENCE' models using both representations. Note that some symbols of ESSENCE, such as “ $\forall$ ” and “ Σ ”, can also be written textually, such as “forall” and “sum”. In writing ESSENCE' we use the textual version.

The refinement operator, ρ , is a function that maps every ESSENCE expression to a set of ESSENCE' expressions. As explained later, each of these ESSENCE' expressions is tagged with further information necessary to construct the

²See <http://www.cs.york.ac.uk/aig/constraints/AutoModel/> for the complete set.

given	$nnodes: \text{int}, capacity: \text{int}$
where	$nnodes \geq 1, capacity \geq 1$
letting	$Nodes \text{ be } 1..nnodes$
find	$ring_1: \text{set}(\text{size } capacity) \text{ of } Nodes$ $ring_2: \text{set}(\text{size } capacity) \text{ of } Nodes$
such that	$ring_1 = ring_2$

given	$nnodes: \text{int}, capacity: \text{int}$
where	$nnodes \geq 1, capacity \geq 1$
letting	$Nodes \text{ be } 1..nnodes$
find	$ring'_1: \text{matrix}[\text{indexed by } 1..capacity] \text{ of } Nodes$ $ring'_2: \text{matrix}[\text{indexed by } 1..capacity] \text{ of } Nodes$
such that	$\text{forall } (i: 1..capacity) \text{ exists } (j: 1..capacity) \text{ } ring'_1[i] = ring'_2[j]$ $\text{AllDifferent}(ring'_1)$ $\text{AllDifferent}(ring'_2)$

given	$nnodes: \text{int}, capacity: \text{int}$
where	$nnodes \geq 1, capacity \geq 1$
letting	$Nodes \text{ be } 1..nnodes$
find	$ring'_1: \text{matrix}[\text{indexed by } Nodes] \text{ of } \text{bool}$ $ring'_2: \text{matrix}[\text{indexed by } Nodes] \text{ of } \text{bool}$
such that	$\text{forall } (i: Nodes) \text{ } ring'_1[i] = ring'_2[i]$ $\text{sum } (i: Nodes) \text{ } ring'_1[i] = capacity$ $\text{sum } (i: Nodes) \text{ } ring'_2[i] = capacity$

Figure 3: ESSENCE specification and two ESSENCE' models of the MicroSonet₁ problem.

final model. ρ is defined inductively by a set of uniquely-named equations of the form $R \rho(e) = e'_1 \cup e'_2 \cup \dots \cup e'_n$, where R is the name of the equation, e is an ESSENCE expression and each e'_i is a set of ESSENCE' expressions, usually given via set comprehension. For perspicuity, each equation is split into *rules*, written: $R_1 \rho(e) \xrightarrow{\text{ref}} e'_1, R_2 \rho(e) \xrightarrow{\text{ref}} e'_2, \dots, R_n \rho(e) \xrightarrow{\text{ref}} e'_n$.

The SIZEDSETEQUALITY equation is responsible for refining expressions of the form $S_1 = S_2$, where both S_1 and S_2 are expressions of type fixed-size set. Fig. 4 gives two rules of this equation. The complete version of SIZEDSETEQUALITY gives one rule for each combination of a representation of S_1 and a representation of S_2 and a constraint for imposing equality between the them.

To illustrate the operation of a rule, we now discuss how the SIZEDSETEQUALITY1 rule refines $ring_1 = ring_2$. SIZEDSETEQUALITY1 refines S_1 and S_2 into explicit one-dimensional matrices denoted by S'_1 and S'_2 . The SIZEDSETEQUALITY2 rule proceeds similarly. The reader may wonder at the strategy of refining the arguments of the equality constraint inside the equality rule itself. This is explained in the next sub-section.

When a rule needs to introduce a new identifier, such as that denoted by S'_1 , it does so by making use of the `genSymbol` function. This function takes two arguments. The first is either an explicit category (see Sec. 3), or an identifier from which the category information is copied. The second is the type of the new identifier, such as a one-dimensional matrix used for the explicit representation. The `genSymbol` function creates a new identifier of the required category and type that appears nowhere else in the ESSENCE' model being constructed. When, for example, refining $ring_1$ with SIZEDSETEQUALITY1, $S'_1 = \text{genSymbol}(ring_1, \text{matrix}[\text{indexed by } Nodes] \text{ of } 1..n)$, and so S'_1 will denote a unique identifier for a matrix of

decision variables (since $ring_1$ is a decision variable).

Definitions via `genSymbol` and type information (except for the input expression) are given on the right of the long vertical bar. The bar itself has no meaning beyond separating the details of the rule from the types and definitions.

Since SIZEDSETEQUALITY1 chooses to represent both sets explicitly, the expression $e = (\forall i \exists j S'_1[i] = S'_2[j]) \wedge (n_1 = n_2)$ is used to constrain the two explicit matrices to represent the same set (note the introduction of the quantified variables denoted by i and j). Since, in general, it may be the case that e is not yet in ESSENCE', it is refined further and the result, denoted by ϕ , is returned. In this example, $e = \forall i \exists j ring'_1[i] = ring'_2[j] \wedge (capacity = capacity)$ is in ESSENCE', so $\rho(e) = e$. For brevity, the model shown in Fig. 3 omits $capacity = capacity$; a simplifier could easily remove it. This convention is followed in all subsequent models generated by SIZEDSETEQUALITY1.

Recall that the explicit representation of a set (size n) of τ is a matrix of n distinct elements, each of type τ . Thus, when a rule generates this explicit representation, it must introduce into the generated model a constraint that the elements of the matrix are all different. In particular, the SIZEDSETEQUALITY1 rule must introduce two such constraints: $\text{AllDifferent}(S'_1)$ and $\text{AllDifferent}(S'_2)$. And, since the elements of S'_1 and S'_2 have not been refined, ρ must be applied to these two constraints, resulting in the ESSENCE' constraints named χ and ψ in the rule.

Consider where χ and ψ should occur in the generated model. From the ESSENCE' model of Fig. 3 it might appear that χ and ψ should be conjoined to ϕ , returning $\phi \wedge \chi \wedge \psi$. To see that this is incorrect, observe that this treatment would refine $\neg(ring_1 = ring_2)$ to $\neg(\phi \wedge \chi \wedge \psi)$, whereas the desired refinement is $(\neg\phi) \wedge \chi \wedge \psi$. The correct refinement is obtained by returning ϕ as the refinement, tagging the refinement with $\chi \wedge \psi$, and once the ESSENCE' model is generated, adding $\chi \wedge \psi$ to its constraints. Since, in this case, the tag is a constraint, it is labelled by `such that`.

When a rule builds a refinement R , a part P of which has been generated from a recursive call to ρ , the tags of P are, by default, added to the tags of R . Hence, in SIZEDSETEQUALITY1, the tags attached to a refinement of ϕ , χ and ψ are implicitly added to the `such that` tag explicitly given in the rule. As will be seen later, this default can be overridden.

Space precludes giving the full derivation of the explicit ESSENCE' model (Fig. 3). The refinements triggered via SIZEDSETEQUALITY1 in the example are, however, straightforward. Universal quantification over a finite range of integers is in ESSENCE' and can be viewed as a conjunction or simple 'for' loop. Similarly, existential quantification over a finite range of integers can be treated as disjunction. Hence, the first constraint in the explicit ESSENCE' model comes directly from SIZEDSETEQUALITY1. The occurrence model follows similarly.

After refining the constraints and the objective function of an ESSENCE specification, the given `find` and `letting` statements are generated. This is done by scanning the set of constraints and adding an appropriate definition for each unique identifier. `where` statements are refined in the same

SizedSetEquality1 $\rho(S_1 : \text{set}(\text{size } n_1) \text{ of } \tau = S_2 : \text{set}(\text{size } n_2) \text{ of } \tau) \xrightarrow{\text{ref}}$ $\{ \begin{array}{l} \phi \\ \text{such that } \chi \wedge \psi \\ \\ \phi \in \rho((\forall i \exists j S'_1[i] = S'_2[j]) \wedge (n_1 = n_2)) \\ \chi \in \rho(\text{AllDifferent}(S'_1)) \\ \psi \in \rho(\text{AllDifferent}(S'_2)) \end{array} \}$	$\begin{array}{l} S'_1 = \text{genSymbol}(S_1, \text{matrix}[\text{indexed by } 1..n_1] \text{ of } \tau) \\ S'_2 = \text{genSymbol}(S_2, \text{matrix}[\text{indexed by } 1..n_2] \text{ of } \tau) \\ i = \text{genSymbol}(\text{"Quantified variable"}, 1..n_1) \\ j = \text{genSymbol}(\text{"Quantified variable"}, 1..n_2) \end{array}$
SizedSetEquality2 $\rho(S_1 : \text{set}(\text{size } n_1) \text{ of } \tau = S_2 : \text{set}(\text{size } n_2) \text{ of } \tau) \xrightarrow{\text{ref}}$ $\{ \begin{array}{l} \phi \\ \text{such that } \chi \wedge \psi \\ \\ \phi \in \rho((\forall i S'_1[i] = S'_2[i]) \wedge (n_1 = n_2)) \\ \chi \in \rho(\Sigma_i S'_1[i] = n_1) \\ \psi \in \rho(\Sigma_i S'_2[i] = n_2) \end{array} \}$	$\begin{array}{l} \tau \text{ is } [a..b] \text{ or bool} \\ S'_1 = \text{genSymbol}(S_1, \text{matrix}[\text{indexed by } \tau] \text{ of bool}) \\ S'_2 = \text{genSymbol}(S_2, \text{matrix}[\text{indexed by } \tau] \text{ of bool}) \\ i = \text{genSymbol}(\text{"Quantified variable"}, \tau) \end{array}$
ForallRange1 $\rho(\forall i : n_1..n_2 \phi : \text{bool}) \xrightarrow{\text{ref}}$ $\{ \begin{array}{l} \text{forall } (i : n_1..n_2) \phi' \\ \text{such that forall } (i : n_1..n_2) \Gamma \\ \\ (\phi' \text{ with such that } \Gamma) \in \rho(\phi) \end{array} \}$	
ForallSizedSet1 $\rho(\forall i : \tau \in S : \text{set}(\text{size } n) \text{ of } \tau. \phi : \text{bool}) \xrightarrow{\text{ref}}$ $\{ \begin{array}{l} \text{forall } (j : 1..n) \phi' \\ \text{such that } \chi \wedge \text{forall } (j : 1..n) \Gamma \\ \\ (\phi' \text{ with such that } \Gamma) \in \rho(\phi[i \mapsto S'[j]]) \\ \chi \in \rho(\text{AllDifferent}(S')) \end{array} \}$	$\begin{array}{l} j = \text{genSymbol}(i, 1..n) \\ S' = \text{genSymbol}(S, \text{matrix}[\text{indexed by } 1..n] \text{ of } \tau) \end{array}$
SizedSubset1 $\rho(S_1 : \text{set}(\text{size } n_1) \text{ of } \tau \subseteq S_2 : \text{set}(\text{size } n_2) \text{ of } \tau) \xrightarrow{\text{ref}}$ $\{ \begin{array}{l} \phi \\ \text{such that } \chi \wedge \psi \\ \\ \phi \in \rho(\forall i. S'_1[S'_1[i]]) \\ \chi \in \rho(\Sigma_j S'_2[j] = n_2) \\ \psi \in \rho(\text{AllDifferent}(S'_1)) \end{array} \}$	$\begin{array}{l} \tau \text{ is } [a..b] \text{ or bool} \\ S'_1 = \text{genSymbol}(S_1, \text{matrix}[\text{indexed by } 1..n_1] \text{ of } \tau) \\ S'_2 = \text{genSymbol}(S_2, \text{matrix}[\text{indexed by } \tau] \text{ of bool}) \\ i = \text{genSymbol}(\text{"Quantified variable"}, 1..n_1) \\ j = \text{genSymbol}(\text{"Quantified variable"}, \tau) \end{array}$

Figure 4: Example Refinement rules.

way that constraints are.

Refining Nested Types: A Simple Example

To see how expressions of a nested type are refined, consider the *Microsonet₂* specification (Fig. 5), where the goal is to generate two identical sets of rings. The key decision is the representation of *rings₁* and *rings₂*, two sets of sets. Refinement begins with $\rho(\text{rings}_1 = \text{rings}_2)$, requiring **SIZED-SETEQUALITY**. Of the rules given in Fig. 4, only **SIZED-SETEQUALITY1** is applicable, since “set (size *capacity*) of *Nodes*” is neither a range of integers nor Boolean: indexing a matrix by a complex type (here each index would correspond to one of the possible sets of *Nodes*) would often lead to unfeasibly large matrices. Future work will consider relaxing this condition in certain cases.

The justification for refining the arguments of the equality constraint inside the equality rule itself becomes clear in the context of nested types. A seemingly more-natural approach to refining $S_1 = S_2$ would be to refine S_1 and S_2 and constrain the results to be equal. However, given unbounded nesting of types and multiple possible refinements of each type, constraining the results to be equal is not straightforward. Consider the case where S_1 and S_2 are not sets of integers, but sets of sets of ... sets of integers. Having refined S_1 and S_2 there is no fixed constraint on their refinements that enforces equality on S_1 and S_2 . Instead of this approach, our rules are designed to “peel off” a layer of nesting so that the associated constraint/operator can be formed without looking arbitrarily deep inside a refined expression.

Performing **SIZEDSETEQUALITY1** requires refining the

two such that tags, both of which impose that all the elements of a matrix of sets must be different. Since these elements may be of a non-atomic type, this **AllDifferent** constraint must be refined. For space reasons the definition of the **ALLDIFFERENT** refinement rules are omitted.

The main constraint generated by **SIZEDSETEQUALITY1** is $\rho(\forall i \exists j \text{rings}'_1[i] = \text{rings}'_2[j])$. The **FORALLRANGE** equation is now applicable (Fig. 4 gives **FORALLRANGE1**). This rule works by refining ϕ , which contains the free variable i . Each resulting refinement ϕ' and its tag Γ , both of which contain the free variable i , are wrapped in “forall $i : n_1..n_2$.” Refinement continues with $\rho(\exists j \text{rings}'_1[i] = \text{rings}'_2[j])$. **EXISTS RANGE** operates in the same way as **FORALLRANGE**, so we omit the details.

We now have $\rho(\text{rings}'_1[i] = \text{rings}'_2[j])$, for which **SIZEDSETEQUALITY** is used. Both rules are applicable, since $\text{rings}'_1[i]$ and $\text{rings}'_2[j]$ are sets of *Nodes*. The refinement in Fig. 5 uses **SIZEDSETEQUALITY2** to generate an occurrence representation of the inner sets. The rule operates as described earlier, but we highlight how **genSymbol** treats indexed matrices. **genSymbol** respects the index structure of its first argument. Hence, given the indexed one-dimensional array $\text{rings}'_1[i]$, **genSymbol**($\text{rings}'_1[i], \text{matrix} \dots$) creates a two-dimensional Boolean matrix, rings''_1 (see Fig.5), and returns the partially-indexed $\text{rings}''[i]$, which is the refined version of an element of the set of sets.

The remaining refinements are straightforward. Note that rings'_1 , rings'_2 are not in the model. These are intermediate representations which, although more concrete than rings_1

given	$nrings : \text{int}, nnodes : \text{int}, capacity : \text{int}$
where	$nrings \geq 1, nnodes \geq 1, capacity \geq 1$
letting	$Nodes \text{ be } 1..nnodes$
find	$rings_1 : \text{set}(\text{size } nrings) \text{ of } \text{set}(\text{size } capacity) \text{ of } Nodes$ $rings_2 : \text{set}(\text{size } nrings) \text{ of } \text{set}(\text{size } capacity) \text{ of } Nodes$
such that	$rings_1 = rings_2$

given	$nrings : \text{int}, nnodes : \text{int}, capacity : \text{int}$
where	$nrings \geq 1, nnodes \geq 1, capacity \geq 1$
letting	$Nodes \text{ be } 1..nnodes$
find	$rings'' : \text{matrix}[\text{indexed by } 1..nrings, Nodes] \text{ of } \text{bool}$ $rings_2' : \text{matrix}[\text{indexed by } 1..nrings, Nodes] \text{ of } \text{bool}$
such that	$\text{forall } (i : 1..nrings) \text{ exists } (j : 1..nrings)$ $\quad \text{forall } (k : Nodes) rings''[i][k] = rings_2'[j][k]$ $\quad \text{forall } (i : 1..nrings) \text{ sum } (j : Nodes) (rings''[i][j]) = capacity$ $\quad \text{forall } (i : 1..nrings) \text{ sum } (j : Nodes) (rings_2'[i][j]) = capacity$ $\quad \text{forall } (i, j : 1..nrings) i < j \text{ implies}$ $\quad \quad \text{exists } (k : Nodes) rings''[i][k] \neq rings_2'[j][k]$ $\quad \text{forall } (i, j : 1..nrings) i < j \text{ implies}$ $\quad \quad \text{exists } (k : Nodes) rings_2'[i][k] \neq rings_2'[j][k]$

Figure 5: ESSENCE specification and an ESSENCE' model of the MicroSonet₂ problem.

given	$nrings : \text{int}, nnodes : \text{int}, capacity : \text{int}$
where	$nrings \geq 1, nnodes \geq 1, capacity \geq 1$
letting	$Nodes \text{ be } 1..nnodes$
given	$demand : \text{set}(\text{size } m) \text{ of } \text{set}(\text{size } 2) \text{ of } Nodes$
find	$rings : \text{set}(\text{size } nrings) \text{ of } \text{set}(\text{size } capacity) \text{ of } Nodes$
such that	$\forall pair \in demand. \exists r \in rings. pair \subseteq r$

given	$nrings : \text{int}, nnodes : \text{int}, capacity : \text{int}$
where	$nrings \geq 1, nnodes \geq 1, capacity \geq 1$
letting	$Nodes \text{ be } 1..nnodes$
given	$demand'' : \text{matrix}[\text{indexed by } 1..m, 1..2] \text{ of } Nodes$
find	$rings'' : \text{matrix}[\text{indexed by } 1..nrings, Nodes] \text{ of } \text{bool}$
such that	$\text{forall } (i : 1..m) \text{ exists } (j : 1..nrings) \text{ forall } (k : 1..2)$ $\quad rings''[j][demand''[i][k]]$ $\quad \text{forall } (i : 1..nrings) \text{ sum } (j : Nodes) (rings''[i][j]) = capacity$ $\quad \text{forall } (i, j : 1..nrings) i < j \text{ implies}$ $\quad \quad \text{exists } (k : Nodes) rings''[i][k] \neq rings''[j][k]$

Figure 6: ESSENCE specification and an ESSENCE' model of the MicroSonet₃ problem.

and $rings_2$, are matrices of sets, and so not in ESSENCE'.

Refining Nested Types: A Complex Example

Our final example illustrates a difficult case when refining nested types: quantification over nested types. Consider MicroSonet₃ (Fig. 6), a variant of a large piece of SONET, where the goal is to fill a given number of rings such that the inter-node communication demand is met. Refinement begins with $\rho(\forall pair \in demand. \exists r \in rings. pair \subseteq r)$, using the FORALLSIZEDSET1 rule (Fig. 4). The first step is to refine the binding expression, $pair \in demand$, which is, in turn, dependent on the refinement of the given set $demand$. FORALLSIZEDSET1 represents $demand$ explicitly in the same way as SIZEDSETEQUALITY1. The returned expression quantifies over the indices of this matrix.

The quantified expression ϕ (which is $\exists r \in rings. pair \subseteq r$ in the example) is refined in the context of the decision to represent the set S explicitly as the matrix S' . In doing so, $S'[j]$ is substituted for i via $\phi[i \mapsto S'[j]]$ (in the example $demand'[j]$ is substituted for $pair$). As per FORALLRANGE1, the tags associated with this refinement must also be quantified to hold for each possible value of j . Having

highlighted this case, space precludes giving further details of the refinement. One particular rule, SIZEDSUBSET1, which is central to this specification, is presented in Fig. 4. This rule shows two expressions, S_1 and S_2 , of the same type being refined to two different representations. The design of CONJURE means this is no more complicated than the previously presented rules, which produce identical representations.

Refining Nested Operators

None of the refinement examples considered so far have used an ESSENCE specification with nested operators. We have avoided the extremely difficult task of formulating refinement rules that recur on both the type of an expression and its syntactic structure. Each recursive rule either works recursively on syntactic structure and handles only atomic types or it works recursively on nested types and handles only expressions that are syntactically “primitive” (in a sense not defined here). A constraint containing *both* nested types and non-primitive syntactic structure is refined by first rewriting it to an equivalent one containing only primitive syntactic structure. For example, the constraint $A \cap (B \cup C) \subseteq D$ is rewritten to $\exists X, Y (X = B \cup C) \wedge (Y = A \cap X) \wedge (X \subseteq D)$. Each conjunct of the resulting constraint is a primitive constraint that has a dedicated refinement rule. CONJURE performs this rewriting in a preprocessing stage called *flattening*.

5 Some Symmetry Detection is Free

Refinement often introduces symmetries into the models it generates; that is, the generated model contains symmetries that do not correspond to symmetries in the original specification. This always happens when a variable X in the specification is represented by a collection of variables $X'_1, \dots, X'_n$ in the model such that an instantiation of X corresponds to multiple instantiations of $X'_1, \dots, X'_n$. For example, if X is of type $\text{set}(\text{size } 3)$, its explicit representation is a matrix of three variables, $X'[1..3]$. An instantiation of X corresponds to $3!$ instantiations of $X'[1..3]$. The matrix X' has index symmetry [6], which can be broken by the symmetry-breaking constraint $X'[1] \leq X'[2] \leq X'[3]$.

Since certain rules always introduce symmetry, such a rule can tag the refined expression with a description of the symmetry. For example, (though not shown in Fig. 4) SIZEDSETEQUALITY1 tags ϕ with “symmetry $indexsym(\phi, 1)$,” which indicates that ϕ has symmetry in its first index. Once a model has been generated, the `symmetry` tags can be used to add appropriate symmetry-breaking constraints, or to guide a symmetry-aware search method such as SBDS or SBDD.

Detecting symmetry as refinement introduces it is a considerably cheaper operation than symmetry detection on a finished model. In future, we intend to extend our system of `symmetry` tags to include other types of symmetry, such as partial index symmetry and value symmetry.

6 Generating Models with Channelling

As we have seen, ESSENCE expressions have multiple possible refinements. If the same expression appears multiple times in a specification, then, different refinements of it may appear in an ESSENCE' model. Sometimes these concrete

representations have a different name but they are equivalent. In those cases CONJURE automatically unifies them into a single representation. However, CONJURE often produces distinct concrete representations for the same abstract symbol. At first sight, this may appear to be a weakness. On the contrary, some of the most effective models in the literature take this approach, since some representations allow certain constraints to be modelled more easily than others. To maintain consistency among the different representations, *channelling constraints* are used [4].

To facilitate channelling, (though not shown in Fig 4) the refinement rules add *represent* tags to the refinements to record the relationship between ESSENCE identifiers and their representations in the concrete model. For example, when refining a decision variable S of type *set (size n) of int* it is possible for S to be represented by both occurrence and explicit representations in the same model. If that is the case the model will contain the tags “*represent S by $expset(S_1)$* ” and “*represent S by $occset(S_2)$* ,” connecting S with its concrete representations S_1 and S_2 . By following the chain of tags it is possible to connect each concrete object to the original entity it is representing, and from these it is possible to construct channelling constraints.

7 Experimental Results

This paper makes an empirically testable conjecture: it is possible to develop good constraint models for combinatorial problems by specifying the problem in ESSENCE and refining this specification with CONJURE. At this point we can test the conjecture up to certain limits. The first limitation is that we currently are concerned only with the kernel of the model, which lacks channelling constraints and symmetry-breaking constraints and which has not had any transformations applied. The second limitation is that the implementation of CONJURE currently has rules for only certain types: integers, integer ranges, Booleans, bounded and fixed-size sets, bounded and fixed-size multisets, and all compositions of these. The implemented rules can generate ESSENCE’ models with occurrence and explicit representations and with variables whose domains contain sets of integers.

We test this conjecture by identifying problems that can be specified using only the supported types, and checking whether CONJURE generates the kernel of all reasonable models, particularly models appearing in the literature. Most problems in the literature cannot be specified abstractly with the supported types since they involve associating together elements from two or more sets, which requires types such as functions or relations. We have found seven problems in the literature that can be specified abstractly with only the supported types: Schur’s Lemma (SL), Balanced Incomplete Block Design (BIBD), Steiner Triple Systems (STS), SONET (SONET), Social Golfers Problem (SGP), RL Design (RLD), and Balance Code Generation (BCG). Though we know of no published models for RLD and BCG (both come from [5]), we have included them simply to test whether CONJURE can generate reasonable, correct models.

The results of running CONJURE on ESSENCE specifications of these seven problems are summarised in the Table 1.

Problem	#	Choices	Yes	No
BIBD	9	dv: mset of sets (2)	[12]	-
SL	9	dv: mset of sets (2)	[6]	[6]
RLD	9	dv: mset of sets (2)	-	-
STS	27	dv: mset of sets (3)	[12][13]	-
BCG	3	dv: mset of sets (1)	-	-
SONET	27	dv: mset of sets (2) in: set of sets (1)	$4 \times [8]$	$4 \times [8]$
SGP	54	dv: mset of set of sets (2)	[12][13]	[16]

Table 1: Results of running CONJURE on seven problems.

To see how to interpret the table, consider the row labelled SONET. This indicates that CONJURE generated 27 models for this problem. The choices it had to make were how to represent each of the two occurrences of the decision variable (whose type is *set of set*) and the one occurrence of the input parameter (whose type is *set of sets*). Our survey of the literature uncovered four published models (all in [8]) that were among those generated by CONJURE and four (all in [8]) that were not. The entry for SGP indicates that CONJURE generated two of the three models that we found in the literature. Though some models appear in multiple published papers, the table cites only one.

Examining the results of the experiments, we observed that all of the models generated by CONJURE were correct. The kernels of many models found in the literature were generated by CONJURE; that is, the models use the same decision variables and the same problem constraints (as opposed to implied or channelling constraints).

It is more enlightening to consider the models not generated by CONJURE. The four models of SONET not generated by CONJURE arise straightforwardly by considering the decision variable to be a relation between rings and nodes. The rules for refining a relation in this way could be expressed easily in the framework of CONJURE and would enable the generation of these four additional models. This illustrates that there are important issues involved in writing specifications in ESSENCE, which in this case are also intertwined with the detection of symmetry; all of this is outside the scope of this paper.

The missing model of SL could be generated easily from the most natural expression of the problem, which takes the decision variable to be a partition. Again, it would be straightforward to add to CONJURE the necessary rules for partitions.

Finally, the model of SGP not generated by CONJURE uses a novel modelling technique that was created through insight into the structure of the problem. We have not yet considered whether that technique can be generalised and then captured in CONJURE rules.

8 Related Work in Modelling Support

OPL [19], a modelling language similar to ESSENCE’, has been extended by ESRA [7] to provide relation variables and by $\mathcal{F}$ [11] to provide function variables. ESRA currently compiles to only a single representation, but $\mathcal{F}$ can produce alternative models, some with multiple representations and channelling. Other work has extended constraint languages

to support a single representation of both sets [10] and multisets [20]. However our work is unique in providing nested types.

Other languages have been used to specify combinatorial problems. NP-SPEC [3] uses existential second-order logic to express executable specifications of every NP-complete problem. Renker et al. [15] discuss representing abstract specifications in Z, an expressive abstract language that is supported by automatic type-checkers and interactive tools. However, refinement and transformation of the Z specification is manual.

Multi-tac [14] and CACP [2; 1] are systems that perform small-scale transformations of constraint specifications and use heuristics and search to try to improve the search strategy used in the solving of CSPs. These and similar systems are important steps towards automating the solving of CSPs, and complement our work on CONJURE, which at present is only concerned with refining of high-level specifications and not the search strategy to be used, and does not implement transformation rules as of yet.

The area of automatic program refinement is a long-standing and mature area of computer science. KIDS [17] and Specware [18] are among the many domain-independent systems that support the interactive or automatic refinement of formal specifications into *imperative* programs. Our work on CONJURE operates in a simpler setting of transforming specifications into finite-domain constraint models, which are *declarative*. Because our setting is simpler, we can be more ambitious about the extent of automation achieved.

9 Conclusion

This paper has made two major contributions towards realising the possibility of formalising and automating the generation of constraint models. First, the paper has presented a language that enables combinatorial problems to be specified at a level above that at which modelling decisions take place. Second, the paper has shown how a system of formal rules could refine such a specification into model kernels at the level of abstraction supported by existing constraint solvers. We have also outlined an approach to integrating other aspects of model generation—channelling and symmetry breaking—into the existing system.

A rigorous account of model generation would be valuable in several ways. It could make our study of modelling more systematic, revealing gaps in our understanding. It could also guide the study of model selection by identifying the decision points and the set of alternatives available at each. Furthermore, it would be useful in teaching and presenting modelling and in constructing a catalogue of modelling constructs.

The automation of model generation offers further opportunities. CONJURE currently uses its rules to generate all possible models; but the rules could also be used within an interactive system, which, like an interactive theorem prover, allows the user to choose from among the alternatives available at a decision point. Our ultimate dream is that the automation of model generation takes us one step closer to automating the entire modelling process.

Acknowledgements Ian Miguel is supported by a UK-Royal Academy of Engineering/EPSRC Postdoctoral Fellowship. We thank Adam Bakewell for contributing to an early version of this work, and Brahim Hnich and Toby Walsh for useful discussions.

References

- [1] J. Borrett and E.P.K. Tsang. A context for constraint satisfaction problem formulation. *Constraints*, 6:299–327, 2001.
- [2] R. Bradwell, R. Ford, J. Mill, E.P.K. Tsang, and R. Williams. An overview of the CACP project: Modelling and solving constraint satisfaction/optimisation problems with minimal expert intervention. *Wkshp. on Analysis & Visualization of Constraint Programs & Solvers*, 2000.
- [3] M. Cadoli, G. Ianni, L. Palopoli, A. Schaerf, and D. Vasile. NP-SPEC: An executable specification language for solving all problems in NP. *Computer Languages*, 26:165–195, 2000.
- [4] B. M. W. Cheng, K. M. F. Choi, J. Ho-Man Lee, and J. C. K. Wu. Increasing constraint propagation by redundant modeling: An experience report. *Constraints*, 4:167–192, 1999.
- [5] C. J. Colbourn and J. H. Dinitz. *The CRC Handbook of Combinatorial Designs*. CRC, 1995.
- [6] P. Flener, A. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, J. Pearson, and T. Walsh. Breaking row and column symmetries in matrix models. *Proc. 8th Int. Conf. on Princ. & Pract. of CP*, LNCS 2470, 462–476, 2002.
- [7] P. Flener, J. Pearson, M. Agren. Introducing ESRA, a relational language for modelling combinatorial problems. LOPSTR’03: Revised Selected Papers, LNCS 3018, 214–232, 2004.
- [8] A.M. Frisch, B. Hnich, I. Miguel, B.M. Smith, and T. Walsh. Transforming and refining abstract constraint specifications. *Proc. of CSCLP04: Joint Annual Workshop of ERCIM/Colognet on Constraint Solving & Constraint Logic Programming*, 74–88, 2004.
- [9] I. P. Gent and T. Walsh. CSPLib: A benchmark library for constraints. Technical Report APES-09-1999, APES, 1999.
- [10] C. Gervet. Conjunto: Constraint logic programming with finite set domains. *Proc. Int. Symposium in Logic Programming*, 339–358. The MIT press, 1994.
- [11] B. Hnich. *Function Variables for Constraint Programming*. PhD thesis, Dept. Information Science, Uppsala Univ, 2003.
- [12] Z. Kiziltan. *Symmetry Breaking Ordering Constraints*. PhD thesis, Dept. Information Science, Uppsala Univ., 2004.
- [13] V. Lagoon and P. J. Stuckey. Set domain propagation using ROBDDs. *Proc. 10th Int. Conf. on Principles & Practice of Constraint Programming*, 347–361, 2004.
- [14] S. Minton. Integrating heuristics for constraint satisfaction problems: A case study. *Nat. Conf. on AI*, 120–126, 1993.
- [15] G. Renker and A. Ahriz. Building models through formal specification. *Proc. CP-AI-OR-04*, LNCS 3011, 295–401, 2004.
- [16] B. M. Smith. Reducing symmetry in a combinatorial design problem. *Proc. CP-AI-OR*, 351–359, 2001.
- [17] D. R. Smith. KIDS: A knowledge-based software development system. *Automating Software Design*, 483–514. MIT Press, 1991.
- [18] *Specware 4.0 User Manual*. Kestrel Institute, October 1994.
- [19] P. van Hentenryck. *The OPL Optimization Programming Language*. MIT Press, 1999.
- [20] T. Walsh. Consistency and propagation with multiset constraints: A formal viewpoint. *Proc. 9th Int. Conf. on Princ. & Pract. of CP*, LNCS 2833, 724–738, 2003.

Bin-Completion Algorithms for Multicontainer Packing and Covering Problems

Alex S. Fukunaga, Richard E. Korf
Computer Science Department
University of California, Los Angeles
fukunaga@cs.ucla.edu, korf@cs.ucla.edu

Abstract

Bin-completion, a bin-oriented branch-and-bound approach, was recently shown to be promising for the bin packing problem. We propose several improvements to bin-completion that significantly improves search efficiency. We also show the generality of bin-completion for packing and covering problems involving multiple containers, and present bin-completion algorithms for the multiple knapsack, bin covering, and min-cost covering (liquid loading) problems that significantly outperform the previous state of the art. However, we show that for the bin packing problem, bin-completion is not competitive with the state of the art solver.

1 Introduction

Many NP-hard problems involve packing some set of discrete objects into multiple containers (“bins”). In one class of problems, the objective is to *pack* some items into a set of containers without exceeding the containers’ capacities. In a related class of problems, the goal is to *cover* a set of containers by filling them up to at least some minimal level using a set of items. When both the containers and the items are modeled as one-dimensional objects (possibly with an associated cost/value function), we refer collectively to these problems as *one-dimensional, multicontainer packing and covering problems*, or *multicontainer packing problems* in short. Multicontainer packing problems are ubiquitous, and model many AI applications including the allocation and rationing of resources or tasks among a group of agents, as well as operations research problems such as cargo loading and transport.

The most widely studied instance of a problem in this class is the *bin packing problem*: Given a set of items (numbers), and a fixed bin capacity, assign each item to a bin so that the sum of the items assigned to each bin does not exceed the bin capacity. For example, given the set of items 6, 12, 15, 40, 43, 82, and a bin capacity of 100, we can assign 6, 12, and 82 to one bin, and 15, 40, and 43, to another, for a total of two bins. This is an optimal solution to this instance, since the sum of all the items, 198, is greater than 100, and hence at least two bins are required.

In this paper, we consider *complete* algorithms for finding *optimal* solutions to multicontainer packing problems. Multicontainer packing problems are generally strongly NP-hard [Garey and Johnson, 1979] (including all of the problems covered in this paper). Therefore, state of the art, complete algorithms for finding optimal solutions are based on branch-and-bound. In contrast, note that single-container problems such as the classical 0-1 knapsack problem are weakly NP-hard, and pseudopolynomial time dynamic programming algorithms exist (c.f. [Kellerer *et al.*, 2004]).

Previous branch-and-bound algorithms for these problems have tended to be “item-oriented”. For example, the classical algorithm for bin packing is the Martello and Toth algorithm [Martello and Toth, 1990]. Items are considered one at a time. Each node in the search corresponds to a decision regarding the assignment of an item to some non-full container. In contrast, *bin-completion* is a “bin-oriented” branch-and-bound strategy, where a node represents an assignment of a set of items to a single container. In addition to standard branch-and-bound pruning methods, a combinatorial *dominance criterion* is used to prune search.

This paper begins by reviewing the previous work on bin-completion. We then propose several improvements, including a generalization of the nogood pruning method proposed in [Korf, 2003] that significantly improves search efficiency. We demonstrate the generality of the bin-completion approach by presenting new bin-completion algorithms for the multiple knapsack, bin covering, and min-cost covering problems that significantly outperform the state of the art algorithms.

2 Bin-Completion

Bin-completion is a bin-oriented branch-and-bound strategy that exploits dominance properties to reduce the search space. For clarity, we describe the bin-completion algorithm and our extensions in the context of the bin packing problem due to its familiarity and simplicity, although the focus of our empirical work is on other multicontainer problems.

A *feasible set* is a set of items whose sum satisfies the capacity constraint (i.e., does not exceed the bin capacity). We say that a feasible set or bin assignment is *maximal* if no additional item can be added to the set without making the set infeasible. The items are sorted in decreasing order of size. We then generate maximal, feasible sets that include the largest

item. If there is more than one such set, the search may branch at that point. Each node of the search tree, except the root node, represents a complete, feasible assignment of items to a particular bin. The children of the root represent different ways of completing the bin containing the largest item. The nodes at the next level represent different complete, feasible sets that include the largest remaining item, etc. The depth of any branch of the tree is the number of bins in the corresponding solution. Bin-completion is a branch-and-bound algorithm. It starts with an upper bound, such as the best-fit decreasing solution, and applies a lower-bound heuristic function to prune the search. Rather than assigning items one at a time to bins, it branches on the different maximal, feasible sets that can be assigned to each bin.

The key to making bin-completion efficient is the use of a *dominance criterion* on the maximal, feasible sets that allows us to only consider a small subset of them.

Let A and B be two feasible sets. If the elements in B can be partitioned into subsets, and these subsets can be matched to the elements of A such that the sum of the elements in each subset doesn't exceed the corresponding element of A , then set A *dominates* set B . In other words, if the elements of B can be packed into bins whose capacities are the elements of A , then set A dominates set B . For example, let $A = \{20, 30, 40\}$ and let $B = \{5, 10, 10, 15, 15, 25\}$. Partition B into the subsets $\{5, 10\}$, $\{25\}$, and $\{10, 15, 15\}$. Since $5 + 10 \leq 20$, $25 \leq 30$, and $10 + 15 + 15 \leq 40$, set A dominates set B . Given all the feasible sets that contain a common element s , only the undominated sets need to be considered for assignment to the bin containing s . The reason is that if we complete the bin containing s with a dominated set, then we could swap each subset of items in the dominated set with the corresponding element of the dominating set, and get another solution without increasing the total number of bins.

This dominance criterion was proposed by Martello and Toth [Martello and Toth, 1990], and was used in their branch-and-bound algorithm for bin-completion. However, their branch-and-bound algorithm was item-oriented, and they only exploit this dominance property in a limited way. In particular, they take each item j , starting with the largest element, and check if there is a *single* assignment of one or two more elements that dominates all feasible sets containing j . If so, they place j with those elements in the same bin, and apply the reduction to the remaining subproblem. They also use dominance relations to prune some element placements as well. Korf's bin-completion algorithm [Korf, 2002] makes much greater use of the Martello-Toth dominance criterion. In particular, when branching on the completion of any bin, it only considers undominated completions.

Historically, the first bin-completion algorithm we are aware of was proposed by Christofides, Mingozi, and Toth in 1979 for the liquid loading problem, which we call the *min-cost covering problem* (see Section 6) [Christofides *et al.*, 1979]. Although their algorithm performs a bin-oriented branch-and-bound which considers only undominated completions, they use a much weaker dominance criterion (see Section 6). However, subsequent research in branch-and-bound algorithms for multicontainer problems focused on item-oriented approaches, and the bin-completion approach

was apparently not investigated further until recently.

The BISON algorithm for bin packing by Scholl, Klein, and Jurgens [Scholl *et al.*, 1997] is a hybrid algorithm that integrates a suite of complex lower bounding procedures, upper bounding heuristics, and a branch-and-bound algorithm. The branch-and-bound component of BISON is a bin-completion algorithm where each node corresponds to a maximal, feasible bin assignment. A very limited form of the Martello-Toth dominance criterion is applied, as follows. If a maximal, feasible bin assignment has a pair of items that can be replaced by single unassigned item without decreasing the sum of the assignment, then it is dominated, and this node can be pruned.

Korf implemented a bin-completion algorithm for bin packing using the full Martello and Toth dominance criterion, and showed that it significantly outperformed the Martello-Toth item-oriented branch-and-bound algorithm [Korf, 2002]. Further improvements were achieved by using a more efficient algorithm for generating the undominated bin assignments, and nogood pruning (see below) [Korf, 2003].

3 Extensions to Bin-Completion

3.1 Nogood Dominance Pruning

Suppose we have a bin packing instance with the numbers 10, 9, 8, 7, 7, 3, 3, 2, 2, and bin capacity $c=20$. Let B_d represent a bin assignment at depth d . $\{10, 8, 2\}$ and $\{10, 7, 3\}$ are two undominated feasible bin assignments, and thus $B_1 = \{10, 8, 2\}$ and $B_1 = \{10, 7, 3\}$ are two possible assignments of the bin (node) at depth 1.

Korf [Korf, 2003] proposed the following *nogood pruning* technique. After exhausting the subproblem below the assignment $B_1 = \{10, 8, 2\}$, and while exploring the subproblem below the assignment $B_1 = \{10, 7, 3\}$, assume we find a solution that assigns $B_2 = \{9, 8, 2\}$. We can swap the $\{8, 2\}$ from B_2 with the $\{7, 3\}$ from B_1 , resulting in a solution with $B_1 = \{10, 8, 2\}$ and $B_2 = \{9, 7, 3\}$. However, we have already exhausted the subtree below $B_1 = \{10, 8, 2\}$, and therefore, we can prune this branch because it is redundant.

In general, given a node with more than one child, when searching the subtree of any child but the first, we don't need to consider bin assignments that assign to the same bin all items used to complete the current bin in a previously-explored child node, except for the largest element. More precisely, let $\{N_1, N_2, \dots, N_m\}$ be a set of brother nodes in the search tree, and let $\{S_1, S_2, \dots, S_m\}$ be the sets of items used to complete the bin in each node, excluding the first item assigned to the bin, which is common to all the brother nodes. When searching the subtree below node N_i for $i > 1$: for all $j < i$, we exclude any bin assignment B that (1) includes all the items in S_j in the same bin, and (2) swapping S_j from B with the items S_i in N_i results in two feasible bin assignments. By rejecting these bin assignments as redundant, the number of node generations is reduced.

We now propose an extension to this idea that allows pruning even more nodes, which we call *nogood dominance pruning*, or **NDP**. Suppose that after exhausting the subproblem below the assignment $B_1 = \{10, 8, 2\}$, and while exploring the subproblem below the assignment $B_1 = \{10, 7, 3\}$, we consider the assignment $B_2 = \{9, 7, 2\}$. We can swap the

$\{7, 2\}$ from B_2 with the $\{7, 3\}$ from B_1 and end up with a solution with the $B_1 = \{10, 7, 2\}$ and $B_2 = \{9, 7, 3\}$. However, according to the Martello-Toth dominance criterion, $B_1 = \{10, 7, 2\}$ is dominated by $B_1 = \{10, 8, 2\}$, and we have already exhausted the search below the node with the subtree under $B_1 = \{10, 8, 2\}$, so we can prune the search because it is not possible to improve upon the best solution under $B_1 = \{10, 8, 2\}$.

In general, given a node with more than one child, when searching the subtree of any child but the first, we don't need to consider assignments that are dominated by a bin assignment in a previously-explored child node. More precisely, when searching the subtree below node N_i for $i > 1$, we exclude any bin assignments that are dominated by the items in S_j , for $j < i$. Note that an assignment dominates itself. Thus, no bin completion below node N_i can be dominated by the items in S_j , for $j < i$.

Nogood dominance pruning is strictly more powerful than Korf's nogood pruning. Any node pruned by nogood pruning will be pruned by NDP, but not vice versa. Of course, since NDP must detect dominance relationships as opposed to equivalence relationships, NDP will incur more overhead per node compared to nogood pruning. Our current implementation propagates a list of nogood sets along the tree. While generating the undominated completions for a given bin, we check each one to see if it is dominated by any current nogood. If so, we ignore that bin-completion.

Since the size of the nogood list increases with depth, and we compare each bin-completion against each nogood, the per-node overhead of NDP increases with depth. This means that pruning at the bottom of the tree (where pruning has the lowest utility) is more expensive than pruning at the top of the tree (where pruning has the highest utility). A simple strategy which address this issue is *depth-limited NDP*, where NDP pruning is applied only at nodes down to the NDP depth limit L . At nodes below the depth limit, Korf's original nogood pruning technique is applied.

In this section, we have described nogood pruning and NDP in the context of the bin packing problem. Adaptations of the technique for the multiple knapsack and bin covering problem are straightforward, given the dominance criteria used for these other problems.

3.2 Further modifications to the search algorithm

In Korf's algorithm for bin packing, at each node, the undominated feasible bin assignments are generated, and sorted according to decreasing sum. The algorithm branches on the completions according to the sorted order. In other words, a largest-sum-first *value ordering heuristic* is being applied at each node.

An issue with enumerating all undominated completions and applying value ordering is that computing the undominated sets is itself NP-complete. Korf [Korf, 2003] gives an algorithm that generates all and only undominated bin-completions, without the need to generate dominated completions as intermediate results. Let d be the average number of items that fit in a container. The time to generate all undominated feasible assignments of a bin increases with d . This is not an issue for bin packing, where problems with large d

tend to be easily solved using heuristics such as best-fit decreasing. The solution found by the heuristic often equals the lower bound, and therefore no search is required, and hence there is no need to compute undominated completions. However, for the other three problems we considered, we have observed that it is much less likely that heuristics will match the optimistic bound and allow termination without search, and we have found that for instances with high d , the algorithm would either take a very long time to complete, or not terminate within a reasonable time limit because it was spending an inordinate amount of time computing the set of undominated completions at each node. Another significant drawback of completely enumerating the undominated bin assignments at each node is the memory required to store all of them.

An alternative approach is to start to go down the search tree and explore the children of a node without first enumerating and sorting the children. In cases where a good optimistic bound (i.e., lower bound for bin packing, and upper bound for bin covering and multiple knapsack) is available, and it takes relatively little search to find an optimal solution, this approach can result in significant speedups compared to the original scheme of generating all completions before going further down the search tree.

On the other hand, we have observed that the value-ordering strategy used to sort the undominated children of a node has a significant impact, and if we simply traverse the search tree depth-first without first enumerating and sorting the children, we lose the benefits of ordering the search tree according to the value-ordering heuristic. The negative impact of this can be alleviated by a hybrid strategy that generates a small number of children, applies the value-ordering heuristic to these, then recursively calls bin-completion on the remaining subproblem. We call this the *hybrid incremental branching strategy*.

Note that even in the absence of an explicit value ordering strategy, the algorithm used to generate the undominated children (for each problem, we use an adaptation of the algorithm given by Korf [Korf, 2003]) imposes an implicit ordering on the undominated assignments. We are currently evaluating various combinations of hybrid incremental branching and various value ordering strategies, and the choices reported in this paper for value ordering strategies and the number of undominated assignments generated at once in hybrid incremental branching are preliminary.

4 The Multiple Knapsack Problem

The *Multiple Knapsack Problem* (MKP) is a generalization of the classical *0-1 Knapsack Problem*. Consider m bins of capacities $c_1, c_2, \dots, c_m$, and a set of n items that have a *weight* $w_1, \dots, w_n$ and a *profit* $p_1, \dots, p_n$. The goal of the MKP is to assign some subset of the items to each bin such that: (1) each item is assigned to no more than one bin, and (2) the sum of the weights of the items assigned to a bin does not exceed the bin's capacity, and (3) the total profit of the items that are assigned to a bin are maximized. For example, suppose there are two bins with capacity 10 and 7, and 4 items (9,3),(7,3),(6,7),(1,5), where the first element of each pair is the weight of the item and the second element is the profit

of that item. The optimal solution to this MKP instance is to assign (9,3) and (1,5) to the bin with capacity 10, and the item (6,7) to the bin with capacity 7. The MKP has numerous industrial applications. For example, the problem of loading m vehicles with cargo selected from n items in order to maximize the value of the transported items is a MKP instance.

4.1 The Mulknep Algorithm

The state of the art algorithm for the MKP is Pisinger’s Mulknep algorithm [Pisinger, 1999]. Mulknep is a branch-and-bound algorithm, where each node represents a decision as to which bin to place an item into, or to leave it out entirely. There are several well-known upper bounds for the MKP. Recent algorithms, including Mulknep, rely on the solution of the *Surrogate relaxed Multiple Knapsack Problem* (SMKP) instance [Martello and Toth, 1990], which is a single-container 0-1 Knapsack problem where the items are the same as for the original MKP instance, but there is a single container whose capacity is the sum of the capacities of the containers in the original MKP instance. Although the 0-1 Knapsack problem is also NP-complete (weakly NP-complete, since there is a pseudopolynomial algorithm), this upper bound computation is very fast in practice. At each node, Mulknep attempts to *validate* the upper bound for the remaining subproblem by distributing the set of items chosen as the solution for the SMKP into the bins. If such a distribution is possible without violating the bin capacities, then the upper bound has been achieved, and we can backtrack without further search down the current branch. The distribution is done by solving a series of subset-sum problems. First, the smallest knapsack is filled as much as possible with the items from the SMKP solution. Then the second smallest is filled with items from the remainder of the SMKP solution, and so on. Again, while this entails the solution of m subset sum instances, it is fast in practice (takes less time than the upper bound computation using the SMKP). In addition, Mulknep also incorporates techniques for reducing the problem instance at every node, as well as tightening the capacity constraints. See [Pisinger, 1999] for details.

4.2 Bin-Completion for the MKP

We define a dominance criterion for the MKP as follows: In a feasible bin assignment for a bin with capacity c_b , the sum of the item weights is no more than c_b .

Let A and B be two feasible assignments. A dominates B if B can be partitioned into i subsets $B_1, \dots, B_i$ such that each subset B_k is mapped one-to-one (but not necessarily onto) to A_k , an element of A , and for all $k \leq i$, (1) the weight of A_k is greater than or equal the sum of the item weights of the items in B_k , and (2) the profit of item A_k is greater than or equal to the sum of the profits of the items in B_k .

Our bin-completion algorithm for MKP uses the same upper bound and reduction techniques as Mulknep. Our current implementation uses a hybrid incremental branching strategy (see 3.2) that generates two undominated children at a time, then uses a value ordering strategy that sorted them according to descending order of profit sums.

We evaluated our MKP algorithm using the same 4 classes of instances used by Pisinger [Pisinger, 1999]. We consid-

ered: (1) uncorrelated instances: p_j and w_j are uniformly distributed in R , (2) weakly correlated instances: w_j uniformly distributed in $[10,1000]$ and p_j randomly distributed in $[w_j - 99, w_j + 99]$ such that $p_j \geq 1$, (3) strongly correlated instances: w_j uniformly distributed in $[10,1000]$ and $p_j = w_j + 10$, and (4) multiple subset sum instances: w_j uniformly distributed in $[10,1000]$ and $p_j = w_j$. The bin capacities were set as follows: The first $m - 1$ capacities c_i were uniformly distributed in $[0.4 \sum_{j=1}^n w_j/m, 0.6 \sum_{j=1}^n w_j/m]$ for $i = 1, \dots, m - 1$. The last capacity c_m is chosen as $c_m = 0.5 \sum_{j=1}^n w_j - \sum_{i=1}^{m-1} c_i$ to ensure that the sum of the capacities is half of the total weight sum (if the first $m - 1$ capacities exceeded half the total weight, the instance was discarded). For each problem class, 30 instances were generated for various values of m and n . Trivial instances were discarded as in [Pisinger, 1999]. The class of uniform, random instances that require the most search for branch-and-bound solvers appear to be generated when n/m is relatively low [Pisinger, 1999; Martello and Toth, 1990]. Thus, the n/m ratio for the MKP appears to be a critical parameter that determines search difficulty. Pisinger has shown that for problems with high n/m ratio, Mulknep is highly effective, solving very large instances almost instantaneously with little or no search (when n ranged from 200 to 100,000, and m ranged 5 or 10). On the other hand, small n/m ratios between 2-3 result in the hardest problems. We therefore focus on these hard problems.

On each instance, we ran Mulknep, bin-completion, and bin-completion with Nogood Dominance Pruning (BC+NDP). For comparison, we used Pisinger’s Mulknep code, available at his website,¹ compiled using the gcc compiler with -O3 optimization settings. Our bin-completion code was implemented in Common Lisp.² Table 1 shows our results. All experiments were run on a 1.3GHz AMD Athlon. Each algorithm was given a time limit of 300 seconds to solve each instance. The *fail* column indicates the number of instances (out of 30) which could not be solved by the algorithm within the time limit. The *time* and *nodes* column show the total time spent and nodes generated **on the successful runs, excluding the failed runs**. Thus, the most important indicator of performance is the number of failed runs.

Both bin-completion and BC+NDP significantly outperform Mulknep, with the difference in performance becoming more pronounced as problem size was increased. This indicates that bin-completion is asymptotically more efficient than Mulknep for this class of problems. BC + NDP consistently outperforms bin-completion by a significant margin with respect to the number of nodes searched, and significant improvements in success rate and runtimes are observed for the larger problems sets. Similar results were obtained when the items had weights and profits in the range $[10,100]$, as

¹<http://www.diku.dk/~pisinger/>

²All of our bin-completion algorithms described in this paper were compiled using CMUCL version 18e. To serve as a point of reference, we ported Korf’s bin-completion algorithm for bin packing to Common Lisp, and found that CMUCL generated code that ran at approximately half of the speed of Korf’s C code compiled with gcc and -O3 optimization.

(m,n)	Mulknap			Bin-Completion			Bin-Completion + NDP		
	fail	time		fail	time	nodes	fail	time	nodes
Uncorrelated Instances									
10,30	10	851	0	330	7994849	0	116	2059561	
15,45	29	82	27	122	2818240	27	83	1389073	
10,20	8	618	0	<1	1866	0	<1	1038	
20,40	30	-	0	141	4703755	0	20	416409	
25,50	30	-	3	1459	47252632	0	490	9229360	
Weakly Correlated Instances									
10,30	11	841	0	129	2168245	0	99	1544200	
15,45	30	-	23	852	13632608	22	908	13181578	
10,20	4	1334	0	<1	1194	0	<1	794	
20,40	30	-	0	16	634072	0	4	84270	
25,50	30	-	1	734	25605562	0	205	3339290	
Strongly Correlated Instances									
10,30	0	571	0	73	684006	0	62	585643	
15,45	30	-	26	661	3697531	25	711	3816290	
10,20	8	808	0	<1	797	0	<1	557	
20,40	30	-	0	9	359645	0	3	58665	
25,50	30	-	1	424	13008484	1	124	1757203	
Subset Sum Instances									
10,30	0	694	0	15	104240	0	14	93727	
15,45	26	711	4	1788	9452862	3	2017	10200510	
10,20	8	941	0	<1	624	0	<1	462	
20,40	30	-	0	3	126931	0	1	28840	
25,50	30	-	0	137	4613658	0	72	1261483	

Table 1: Multiple Knapsack results. (Times are in seconds).

well as the range [10,10000].

5 The Bin Covering Problem

Suppose we have n items with weights $w_1, \dots, w_n$, and an infinite supply of containers with quota q . The *bin covering* problem, also known as the *dual bin packing* problem, is to pack the items into containers such that the number of containers that contain sets of items whose sum is at least q is **maximized**. That is, the goal is to distribute, or ration the items among as many containers as possible, given that the containers have a specified quota that must be satisfied. Note that the total weight of the items placed in a container can be greater than q .

Bin covering is a model for resource or task allocation among multiple agents where the goal is to maximize the number of agents who fulfill some quota. It is also a model for industrial problems such as: (1) packing peach slices into cans so that each can contains at least its advertised net weight in peaches, and (2) breaking up monopolies into smaller companies, each of which is large enough to be viable [Assman *et al.*, 1984].

5.1 The Labbé, Laporte, and Martello Algorithm

The state of the art complete algorithm for the bin covering problem is by Labbé, Laporte, and Martello [Labbé *et al.*, 1995]. We shall refer to this as the *LLM* algorithm. LLM is a branch-and-bound algorithm. The items are sorted in decreasing order of size. Each node represents a decision as to which bin to put an item into, or to leave it out entirely. At each node, lower bounds based on combinatorial arguments are computed, and the remaining subproblem is reduced using two reduction criteria (see [Labbé *et al.*, 1995]). At the root node, a set of heuristics is applied in order to compute a lower bound. As with the bin packing problem (c.f. [Korf, 2002]), many instances of the bin covering problem require

n	Labbé			Bin-Completion			Bin-Completion + NDP		
	fail	time		fail	time	nodes	fail	time	nodes
10,000 instances per row									
10	0	2	0	2	215	0	2	215	
20	0	4	0	4	595	0	4	548	
40	0	78	0	10	5860	0	10	3888	
60	2	458	0	154	3610644	0	127	2467708	
80	17	823	2	110	3191035	1	88	1135615	
100	28	1210	1	198	6045642	1	79	613463	
50,000 instances per row									
80	n/a	n/a	6	332	7597514	4	362	5763523	
90	n/a	n/a	6	574	15079358	4	493	7320350	
100	n/a	n/a	8	803	20739466	6	585	11600379	
120	n/a	n/a	20	1941	65428014	16	1543	33040246	

Table 2: Bin Covering results. (Times are in seconds).

no search, and can be immediately solved at the root node when the heuristics described in [Labbé *et al.*, 1995] compute a lower bound that equals the upper bound.

5.2 Bin-Completion for Bin Covering

We define a dominance criterion for bin covering as follows. Let A and B be two feasible assignments. A dominates B if B can be partitioned into i subsets $B_1, \dots, B_i$ such that each item $A_k \in A$ is mapped one-to-one (but not necessarily onto) a subset B_k , such that for all $k \leq i$, the sum of the item weights of subset B_k is greater than or equal to the corresponding item A_k (i.e., B_k “covers” A_k).

To see that this dominance criterion gives a valid pruning criterion, suppose a solution to a bin-completion instance assigns B to a bin. If we exchange the items in A for the corresponding subsets of B , then this would result in a valid solution that satisfies at least as many bins.

As with LLM, we apply lower bounding heuristics at the root node, and apply the same upper bounds used in LLM [Labbé *et al.*, 1995] at each node. We use a hybrid incremental branching strategy that generates two undominated children at a time, then using smallest-sums-first value ordering to order them.

We compared our implementation of bin-completion for the bin covering problem with our implementation of the LLM algorithm (both implementations were in Common Lisp and run on a 600MHz Pentium 3). We generated 10,000 random instances for each set, where the items were chosen uniformly in the range [1,9999], and the quota was 10,000. We ran our implementations of LLM, bin-completion, and bin-completion + NDP on each instance, with a time limit of 180 seconds per instance. We also compared the two bin-completion variants on larger instances of 80-120 items, using sets of 50,000 random instances. Table 2 shows our results. Again, we show the number of failed runs that timed out before solving the instance, as well as the time spent and nodes generated, **excluding the failed runs**. Bin-completion significantly outperformed LLM. The increasing gap in performance as problem size increases suggests that bin-completion is asymptotically more efficient than LLM. For the harder problems, bin completion + NDP significantly outperforms bin-completion.

6 Min-Cost Covering

We define the *Min-Cost Covering Problem* (MCCP) as follows. Given a set of m bins with quotas $\{q_1, \dots, q_m\}$, and a set of n items with weights $w_1, \dots, w_n$ and costs $p_1, \dots, p_n$, assign some subset of the items to each bin such that (1) each item is assigned to no more than one bin, (2) the sum of the weights of the items assigned to a bin is at least the bin's capacity (i.e., the bin is covered, as in bin covering), and (3) the total cost of the items that are assigned to the bin are minimized. This problem has also been called the *liquid loading problem* [Christofides *et al.*, 1979], because it was originally motivated by the following industrial application: Consider the disposal or transportation of m different liquids (e.g., chemicals) that cannot be mixed. If we are given n tanks of various sizes, each with some associated cost, the problem is to load the m liquids into some subset of the tanks so as to minimize the total cost. Note that here, the “liquids” correspond to containers, and the “tanks” correspond to the items.

6.1 Christofides, Mingozi, and Toth (CMT) Algorithm

The previous state of the art algorithm for the min-cost covering problem is an early version of bin-completion by Christofides, Mingozi, and Toth [Christofides *et al.*, 1979]. The main difference between their algorithm and the bin-completion algorithm described in Section 2 is their use of a different dominance criterion. The Christofides *et al.* dominance criterion (CMT criterion) is as follows. A set of items A dominates another set B if it is possible to map A one-to-one (but not necessarily into) B by a function π such that $w_j \geq w_{\pi(j)} \forall j \in A$, and $p_j \geq p_{\pi(j)} \forall j \in A$.

Note that the CMT criterion only considers mappings between single elements of A and B , whereas the dominance criteria we have described above for bin packing, MKP, and bin covering considers mappings between elements of A and subsets of B .

Christofides *et al.* empirically evaluated a number of lower bounds for the MCCP. They found that the best bound was computed by solving m single-container MCCP problems, one for each of the bins, where all n items were available for the single-bin MCCP problem. Each single-bin instance was solved using a simple branch-and-bound algorithm. The lower bound is the sum of the optimal values for the m single-container MCCP problems. This is essentially a lower bound derived by relaxing the MCCP constraint that each item can be assigned to at most one bin.

6.2 Bin-Completion for the MCCP

Our new bin-completion algorithm for the MCCP is similar to the Christofides *et al.* algorithm. The major difference is that we use the following dominance criterion. A set of i items A dominates another set B if B can be partitioned into i subsets $B_1, \dots, B_i$ such that each item $A_k \in A$ is mapped one-to-one (but not necessarily onto) to a subset B_k , and for all $k \leq i$, (1) the weight of A_k is less than or equal to the sum of the item weights of the items in B_k , and (2) the profit of item A_k is less than or equal to the sum of the profits of the items in B_k . The CMT dominance criterion is a special case of this criterion, where $|B_k| = 1$ for all k .

We evaluated our bin-completion algorithm and the CMT algorithm using the same classes of problem instances that we used for the MKP (Section 4). For each problem class, 30 nontrivial instances were generated for various values of m and n . The purpose of this experiment was to evaluate the relative impact of each component of bin-completion, that is: (1) the type of dominance criterion used, (2) whether nogood pruning was used, and (3) whether nogood dominance pruning was used.

On each instance, we ran (1) **CMT1**: the CMT algorithm as presented in [Christofides *et al.*, 1979], (2) **CMT2**: the CMT algorithm extended with nogood pruning, (3) **BC1**: bin-completion using our dominance criterion but *without* nogood pruning, (4) **BC2**: bin-completion using our dominance criterion and with nogood pruning, (5) **BC3**: bin-completion using our dominance criterion and NDP. For all algorithms, a hybrid incremental branching strategy that generated 100 children at a time was used, and the children were ordered in increasing order of profit sums.

Table 1 shows our results. All experiments were run on a 2.5GHz Pentium IV. Each algorithm was given a time limit of 60 seconds to solve each instance. The *fail* column indicates the number of instances (out of 30) which could not be solved by the algorithm within the time limit. The *time* and *nodes* column show the total time spent and nodes generated, excluding the failed runs. As shown by the data, each component of bin-completion has a significant impact. Although our dominance criterion requires much more computation per node to compute than the simpler CMT criterion, the performance is significantly improved. The nogood pruning strategy significantly improves performance for both the algorithm based on the CMT dominance criterion, as well as our dominance criterion. Finally, BC3, which uses NGD is results in the best performance. We also tested an item-oriented branch-and-bound algorithm that uses the same lower bounding procedure, but found that it performed even worse than the CMT algorithm on all of the test instances.

7 Bin Packing

We implemented our extended bin-completion algorithm for the bin packing problem, and summarize our results below. For further details, see [Fukunaga, 2005].

Extending Korf's algorithm [Korf, 2003] with NDP and a value ordering based on minimum cardinality (with ties broken according to decreasing sum of item weights) resulted in significant speedups over Korf's algorithm on uniform random problem instances, as well as the class of “uniform” OR-LIB³ instances of 120-1000 items, for the same experiments described in Korf's paper [Korf, 2003]. In addition, extending the algorithm further by applying randomized restarts [Gomes *et al.*, 1998] resulted in significant speedups compared to Korf's algorithm on the “triplet” instances from OR-LIB. For example, our extended algorithm (implemented in Common Lisp and executed on a 1.3GHz Athlon) solves all 80 of the triplet instances from OR-LIB within 1500 seconds combined, whereas Korf's algorithm (implemented in C and executed on a 440MHz Sun Ultrasparc) failed to solve any

³see <http://www.brunel.ac.uk/depts/ma/research/jeb/info.html>

(m,n)	CMT1		CMT2		BC1		BC2		BC3	
	fail	time	fail	time	fail	time	fail	time	fail	time
Uncorrelated Instances										
5,15	0	3	0	2	0	<1	0	<1	0	<1
10,30	30	-	30	-	30	-	8	605	6	567
5,10	0	<1	0	<1	0	<1	0	<1	0	<1
10,20	7	324	0	5	0	168	0	1	0	1
15,30	30	-	27	49	30	-	6	360	1	511
Weakly Correlated Instances										
5,15	0	6	0	2	0	<1	0	<1	0	<1
10,30	30	-	30	-	30	-	28	78	27	77
5,10	0	<1	0	<1	0	<1	0	<1	0	<1
10,20	0	88	0	1	0	4	0	<1	0	<1
15,30	30	-	6	325	25	167	0	115	0	20
20,40	30	-	30	-	30	-	28	72	13	410
Strongly Correlated Instances										
5,15	0	2	0	1	0	<1	0	<1	0	<1
10,30	30	-	30	-	4	361	1	207	0	93
5,10	0	5	0	<1	0	<1	0	<1	0	<1
15,30	17	286	0	95	0	64	0	3	0	1
20,40	30	-	19	330	23	330	6	180	0	58
Subset Sum Instances										
5,15	0	2	0	1	0	<1	0	<1	0	<1
10,30	30	-	30	-	4	362	1	224	0	100
5,10	0	<1	0	<1	0	<1	0	<1	0	<1
10,20	0	5	0	<1	0	<1	0	<1	0	<1
15,30	17	289	0	100	0	65	0	3	0	<1
20,40	30	-	19	258	23	172	6	181	0	63

Table 3: Min-cost covering problem. (Times are in seconds).

of the twenty 501-item instances, and failed to solve 8 of the 249-item instances given a 10 minute time limit per instance. See [Fukunaga, 2005] for further results.

However, our bin-completion solver was not competitive with the state of the art, which is currently a branch-and-price integer linear programming solver by Belov and Scheithauer [Belov and Scheithauer, 2003]. Belov and Scheithauer [Belov and Scheithauer, 2003] provide a new benchmark set of 28 very hard bin packing instances; they solved most of these within seconds, although some took hours. Our current bin-completion code could not solve any of the 28 instances, given 15 minutes per instance. They also report that the largest triplet instances from OR-LIB (Triplet-501) were solved in an average of 33 seconds per instance (660 seconds total) on a 1GHz AMD Athlon XP.

Furthermore, Belov was kind enough to run a set of one hundred, 80-item, uniform, random instances that we generated with items in the range [1,1,000,000] and a bin capacity of 1,000,000 for us. His solver solved all of these instances in less than 1 second each at the root node (i.e., without any search), using rounding heuristics based on the linear programming LP solution, whereas our best bin-completion solver required 534 seconds and searched 75,791,226 nodes.

Recent branch-and-price approaches for bin packing such as the solver by Belov and Scheithauer (see [de Carvalho, 2002] for a survey of others) use a bin-oriented branching strategy, where decisions correspond to the instantiation of one or more complete bins. At each node, a column generation procedure is used to compute the LP lower bound. They derive much of their power from a very accurate LP lower bound based on a cutting stock formulation of bin packing [Gilmore and Gomory, 1961], which has been observed to almost always give the optimal value as the lower bound, and has never been observed to give a value that is more

than one higher than the optimal value (c.f. [Wäscher and Gau, 1996]). In addition, rounding heuristics applied to fractional LP-solutions often yields the optimal, integral solution. The combination of a very tight LP lower bound and good upper bounding procedure result in very little search being performed for almost all problem instances. This domain-specific branch-and-price LP-based approach does not seem to generalize straightforwardly to the MKP and MCCC, in part due to the differences in the granularity of the objective function. While it is possible that a similar branch-and-price approach could be applied to the bin covering problem, we are aware of no such approach in the literature.

For completeness, we have applied a straightforward, integer-linear programming approach to the MKP, MCCC, and bin covering problems. We applied the GNU GLPK integer linear programming solver to the mathematical programming formulations of these problems, but not surprisingly, these performed very poorly in comparison with the domain-specific algorithms described in this paper.

8 Related Work

The multicontainer packing problems we studied in this paper can be formulated as constraint programming problems. In one possible formulation, the variables correspond to bins, and the values of the bins correspond to unique, feasible bin assignments (this is similar to the integer linear programming formulation of the cutting-stock problem [Gilmore and Gomory, 1961]). For example, given items with weights $\{3, 2\}$ let x_1 be the variable representing the first bin with capacity 4. The domain of x_1 is the two singleton sets $\{3\}$ and $\{2\}$.

Given this constraint programming formulation, it can be seen that the pruning techniques based on nogoods described in Section 3 are closely related to techniques proposed for constraint programming. Nogood pruning identifies and prunes nodes by detecting whether the bin assignment for the current node contains a nogood. This is very similar to nogood-based techniques for pruning symmetric solutions proposed by Fahle, Schamberger and Sellmann [Fahle *et al.*, 2001] and Focacci and Milano [Focacci and Milano, 2001],

The NGD technique proposed in this paper is similar to the nogood dominance pruning technique proposed by Focacci and Shaw [Focacci and Shaw, 2002] for constraint programming, who applied their technique to the symmetric and asymmetric traveling salesperson problem with time windows. See [Focacci and Shaw, 2002] for a full description of their technique.

Both methods attempt to prune the search by proving that the current node at depth j , which represents a partial j -variable solution x , is dominated by some previously explored i -variable partial solution (nogood), q .

The main difference between the two methods is the approach used to test for dominance. Focacci and Shaw's method extends q to a j -variable partial solution q' which dominates x . They apply a local search procedure to find the extension q' .

On the other hand, our NGD method starts with a partial, j -variable solution x and tries to transform it to a partial solution x' such that $\bar{x}'_i$, the subset of x' including the first i

variables, is dominated by q . We do this by swapping the values of the i th and j th variables in x to derive x' , and testing whether $\bar{x}'_i$ is dominated by q .

For efficiency, the current implementations of both nogood dominance pruning methods are *weak*, in the sense that if x is dominated by q , the procedures will not necessarily detect the dominance. Focacci and Shaw rely on an incomplete, local search to find the extension q' . We only consider transformations involving two variables, but to fully exploit the dominance criterion, we would need to consider transformations involving all variables $i, i + 1, \dots, j$.

9 Conclusions

This paper makes two main contributions: First, we proposed extensions to bin-completion to improve its performance, including nogood dominance pruning, which generalizes Korf's nogood pruning method. Second, we demonstrated the utility and generality of bin-completion by applying it to three new problems: the bin covering problem, the multiple knapsack problem, and the min-cost covering problem. We proposed new dominance criteria for these problems, and showed that our bin-completion solvers significantly outperform, the previous state of the art on hard instances (in some cases by orders of magnitude with respect to runtime). For these three problems, our nogood dominance pruning technique significantly reduced search effort and runtimes. The comparison of our bin-completion algorithm to the early bin-completion algorithm by Christofides et al. for the MCCP showed that our new dominance criteria resulted in significant performance improvements.

Finally, we revisited the bin packing problem and showed that our extensions significantly improved the performance of bin-completion for some standard benchmarks. However, we found that bin-completion is not competitive with state of the art solvers for bin packing based on the cutting stock problem formulation.

While we have focused on four particular multicontainer problems in this paper, there are many similar problems involving the assignment of objects to multiple containers where similar dominance structures can be exploited. Examples include the generalized assignment problem, multiprocessor scheduling, and the segregated storage problem. Given our results, maximally exploiting powerful dominance criteria in a bin-completion framework appears to be a promising future direction for such multicontainer problems.

10 Acknowledgments

Thanks to Gleb Belov for running his solver on some of our bin packing test instances, and to the anonymous reviewers for helpful comments. This research was supported by NSF under grant No. EIA-0113313.

References

[Assman et al., 1984] S.F. Assman, D.S. Johnson, D.J. Kleitman, and J.Y.T. Leung. On a dual version of the one-dimensional bin-packing problem. *J. Algorithms*, 5:502–525, 1984.

[Belov and Scheithauer, 2003] G. Belov and G. Scheithauer. A branch-and-cut-and-price algorithm for one-dimensional stock

cutting and two-dimensional two-stage cutting. Technical report, Technical University Dresden, 2003. Preprint MATH-NM-03-2003.

[Christofi des et al., 1979] N. Christofi des, A. Mingozzi, and P. Toth. Loading problems. In N. Christofi des, A. Mingozzi, P. Toth, and C. Sandi, editors, *Combinatorial Optimization*. John Wiley & Sons, 1979.

[de Carvalho, 2002] J.M. Val'erio de Carvalho. LP models for bin packing and cutting stock problems. *European Journal of Operational Research*, 141:253–273, 2002.

[Fahle et al., 2001] T. Fahle, S. Schamberger, and M. Sellmann. Symmetry breaking. In *Proc. Int. Conference on Constraint Programming*, pages 93–107, 2001.

[Focacci and Milano, 2001] F. Focacci and M. Milano. Global cut framework for removing symmetries. In *Proc. Int. Conference on Constraint Programming*, pages 77–92, 2001.

[Focacci and Shaw, 2002] F. Focacci and P. Shaw. Pruning sub-optimal search branch branches using local search. In *Proc. CP-AI-OR*, 2002.

[Fukunaga, 2005] A. Fukunaga. *Bin-Completion Algorithms for One Dimensional, Multicontainer Packing Problems*. PhD thesis, UCLA, 2005. forthcoming.

[Garey and Johnson, 1979] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, 1979.

[Gilmore and Gomory, 1961] P.C. Gilmore and R.E. Gomory. A linear programming approach to the cutting stock problem. *Operations Research*, 9:849–859, 1961.

[Gomes et al., 1998] Carla P. Gomes, Bart Selman, and Henry Kautz. Boosting combinatorial search through randomization. In *Proc. AAAI*, pages 431–437, Madison, Wisconsin, 1998.

[Kellerer et al., 2004] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer-Verlag, 2004.

[Korf, 2002] R. Korf. A new algorithm for optimal bin packing. In *Proc. AAAI*, pages 731–736, 2002.

[Korf, 2003] R. Korf. An improved algorithm for optimal bin packing. In *Proc. IJCAI*, pages 1252–1258, 2003.

[Labb'e et al., 1995] M. Labb'e, G. Laporte, and S. Martello. An exact algorithm for the dual bin packing problem. *Operations Research Letters*, 17:9–18, 1995.

[Martello and Toth, 1990] S. Martello and P. Toth. *Knapsack problems: algorithms and computer implementations*. John Wiley & Sons, 1990.

[Pisinger, 1999] D. Pisinger. An exact algorithm for large multiple knapsack problems. *European Journal of Operational Research*, 114:528–541, 1999.

[Scholl et al., 1997] A. Scholl, R. Klein, and C. Jürgens. BISON: A fast hybrid procedure for exactly solving the one-dimensional bin packing problem. *Computers in Operations Research*, 24(7):627–645, 1997.

[Wäscher and Gau, 1996] G. Wäscher and T. Gau. Heuristics for the integer one-dimensional cutting stock problem: A computational study. *OR Spektrum*, 18(3):131–144, 1996.

Limited Discrepancy Beam Search*

David Furcy

University of Wisconsin Oshkosh
Computer Science Department
Oshkosh, WI 54901-8643
furcyd@uwosh.edu

Sven Koenig

University of Southern California
Computer Science Department
Los Angeles, CA 90089-0781
skoenig@usc.edu

Abstract

Beam search reduces the memory consumption of best-first search at the cost of finding longer paths but its memory consumption can still exceed the given memory capacity quickly. We therefore develop BULB (Beam search Using Limited discrepancy Backtracking), a complete memory-bounded search method that is able to solve more problem instances of large search problems than beam search and does so with a reasonable runtime. At the same time, BULB tends to find shorter paths than beam search because it is able to use larger beam widths without running out of memory. We demonstrate these properties of BULB experimentally for three standard benchmark domains.

1 Introduction

Best-first search methods, such as A*, do not scale up to large search problems due to their memory consumption, and linear-space best first search methods [Korf, 1993] have unacceptable runtimes for large search problems. Beam search reduces the memory consumption of best-first search at the cost of finding longer paths. It uses breadth-first search to build its search tree but keeps at most the B states at each level of the search tree with the smallest heuristic values, where the value of the beam width B is set at the beginning of the search. The smaller the beam width, the more states beam search prunes at each step of the search and the less memory it needs to store each level of the search tree. Unfortunately, more pruning typically increases the probability of pruning states on short paths from the start state to a goal state and thus often increases the lengths of the paths found. Excessive pruning can even prevent one from finding any path. Thus, the beam width has to be large. Our experiments show, for example, that beam search with a beam width of 10,000 solves about eighty percent of random problem instances of the 48-Puzzle. The average path length found is on average about one order of magnitude smaller than the one found by variants of WA* [Pearl, 1985], which are alternatives to beam search that also

*The Intelligent Decision-Making Group is partly supported by NSF awards to Sven Koenig under contracts IIS-0098807 and IIS-0350584. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations, agencies, companies or the U.S. government.

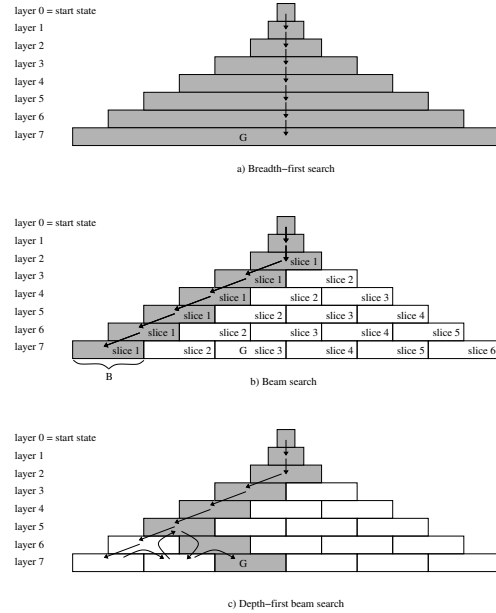


Figure 1: Visualization of Search Methods

reduce the memory consumption of best-first search at the cost of finding longer paths. We develop BULB (Beam search Using Limited discrepancy Backtracking) that is able to solve more problem instances of large search problems than beam search, and does so with a reasonable runtime. At the same time, BULB tends to find shorter paths than beam search because it is able to use larger beam widths without running out of memory. It behaves like beam search until it exhausts the memory capacity without finding a path. It then uses limited discrepancy backtracking to retract its previous pruning decisions. The choice of a good backtracking strategy is important since, for example, beam search with chronological backtracking has unacceptable runtimes.

2 Beam Search

Beam search is any search technique “in which a number of [...] alternatives (the beam) are examined in parallel. [It] is a heuristic technique because heuristic rules are used to discard [prune] non-promising alternatives in order to keep the size

Table 1: Beam Search on the 48-Puzzle

B	Path Length	Generated States	Stored States	Runtime (Seconds)	Problems Solved
1	N/A	N/A	N/A	N/A	0 %
5	11,737.12	147,239	58,680	0.090	100 %
10	36,281.64	904,632	362,799	0.601	100 %
50	25,341.44	3,211,244	1,266,902	2.495	86 %
100	12,129.88	3,079,594	1,212,579	2.296	86 %
500	2,302.86	2,899,765	1,148,559	2.205	74 %
1,000	1,337.95	3,346,004	1,331,451	2.822	84 %
5,000	481.30	5,814,061	2,365,603	5.500	86 %
10,000	440.07	10,569,816	4,312,007	11.307	80 %
50,000	N/A	N/A	N/A	N/A	0 %

of the beam as small as possible” [Bisiani, 1987]. We assume that all actions have a cost of one and study beam-search variants of breadth-first search in this paper. Their objective is to reduce the memory consumption of breadth-first search from exponential to linear in the depth of the search tree, as illustrated by the shaded areas of Figure 1 (a) and (b) for breadth-first search and beam search, respectively. Beam search uses breadth-first search to build its search tree but splits each level of the search tree into slices of at most B states, where B is called the beam width. The number of slices stored in memory is limited to one at each level. When beam search expands a level, it generates all successors of the states at the current level, sorts them in order of increasing heuristic values (from left to right in the figure), splits them into slices of at most B states each, and then extends the beam by storing the first slice only. Beam search terminates when it generates a goal state or runs out of memory.

Table 1 shows experimental results for beam search on the 48-Puzzle with a memory limitation of 6,000,000 states. We created 50 problem instances with random start configurations in which the goal configuration had the blank in the upper left corner. We used the Manhattan distance as heuristic function. (We could have used pattern databases instead [Korf and Taylor, 1996] but did not since we use them later in this paper in the context of 2 additional benchmark domains.) The runtime of beam search was always small since it ran out of memory in seconds. Beam search with a beam width of 1 solved none of the problem instances. This is not surprising since it is similar to greedy search (gradient descent) and thus likely to find rather long paths unless it gets stuck in dead ends because the current state has only successors that are already in memory, in which case it does not find any path at all. Beam search with a beam width of 10 solved all of the problem instances. As the beam width increased, its memory consumption increased, the average path length of the solved problem instances decreased, and the number of solved problem instances eventually decreased. Beam search with beam width 50,000 solved none of the problem instances. This is not surprising since beam search with a beam width of infinity is breadth-first search and thus guaranteed to find shortest paths unless it runs out of memory and then does not find any path at all, which is likely given the exponential memory consumption of breadth-first search. Consider beam search with a large beam width that still solved a substantial number of problem instances, say beam search with a beam width of 10,000 that solved eighty percent of the problem instances. The average path length of the solved problem instances was an order of magnitude smaller than the one reported in [Furcy,

2004] for several variants of WA*.

3 Improving Beam Search

We study how to increase the number of solved problem instances to one hundred percent while reducing the path lengths of the solved problem instances. This cannot be done by varying the beam width since increasing it reduces the number of solved problem instances while decreasing it increases the average path length of the solved problem instances. Rather, we notice that many of the unsolved problem instances are due to misleading heuristic values that prevent states from being included in the beam. For example, the goal state G is put into the third slice of the seventh layer in Figure 1 (b). Beam search thus does not find the goal state since it visits only the first slice of each layer. Our solution to this problem is to backtrack and choose a different slice. Figure 1 (c) shows DB (Depth-first Beam search), our simplest variant of beam search with backtracking. DB behaves like beam search until it exhausts the memory capacity without finding a path. It then uses chronological backtracking to purge existing slices and replace them with others. DB, unfortunately, has unacceptable runtimes, which we explain as follows: Chronological backtracking revisits the most recent decisions first, that is, the decisions close to the bottom of the search tree. This is problematic since the heuristic values are usually the more inaccurate the farther a state is away from the goal state and thus the closer it is to the top of the search tree. Thus, it is important to revisit decisions close to the top of the search tree more quickly. We therefore use limited discrepancy search rather than chronological backtracking to build a more sophisticated variant of beam search with backtracking.

3.1 Original Limited Discrepancy Search

LDS (Limited Discrepancy Search) [Harvey and Ginsberg, 1995] was designed to work on finite binary trees. The successors of a state are sorted in order of increasing heuristic values. Thus, the heuristic values always recommend the left successor over the right one. Choosing the right successor against the recommendation of the heuristic values is called a discrepancy. First, LDS searches the tree greedily, that is, with no discrepancy. If LDS does not find a goal state, then it made at least one wrong decision due to misleading heuristic values. LDS then searches the tree with increasing numbers of allowed discrepancies. Figure 2 contains the pseudo code of LDS. The top-level function *LDS()* repeatedly performs a limited discrepancy search from the start state (Line 4) by calling *LDSprobe()* with an increasing number of allowed discrepancies (Line 6), starting with no discrepancy (Line 2). Unless the current state is a leaf of the tree (Line 9), *LDSprobe()* generates its successors and recursively calls itself on them. If the maximum number of allowed discrepancies is zero, then only the sub-tree below the best successor is visited with no discrepancy allowed (Line 12). Otherwise, the sub-tree under the worst successor is visited with one less discrepancy allowed (since one was just consumed, Line 14), then the sub-tree under the best successor is visited with the same number of allowed discrepancies (since none was con-

```

1. procedure LDS( $s_{start}, h(\cdot)$ ): path length
2.   discrepancies := 0
3.   while (true) do
4.     cost := LDSprobe( $s_{start}, 0, discrepancies, h(\cdot)$ )
5.     if (cost <  $\infty$ ) then return cost
6.     discrepancies := discrepancies + 1
7.   end while
8. procedure LDSprobe(state, depth, discrepancies,  $h(\cdot)$ ): path length
9.   if (state is a leaf) then return  $\infty$ 
10.  else (best, second) := generateSuccessors(state)
11.  if ((best =  $s_{goal}$ ) or (second =  $s_{goal}$ )) then return depth + 1
12.  if (discrepancies = 0) then return LDSprobe(best, depth + 1, 0,  $h(\cdot)$ )
13.  else
14.    cost := LDSprobe(second, depth + 1, discrepancies - 1,  $h(\cdot)$ )
15.    if (cost <  $\infty$ ) then return cost
16.  return LDSprobe(best, depth + 1, discrepancies,  $h(\cdot)$ )

```

Figure 2: Original Limited Discrepancy Search

```

1. procedure GLDS( $s_{start}, h(\cdot)$ ): pathlength
2.   discrepancies := 0; hashtable := { $s_{start}$ }
3.   while (true) do
4.     pathlength := GLDSprobe( $s_{start}, 0, discrepancies, h(\cdot)$ )
5.     if (pathlength <  $\infty$ ) then return pathlength
6.     discrepancies := discrepancies + 1
7.   end while
8. procedure GLDSprobe(state, depth, discrepancies,  $h(\cdot)$ ): path length
9.   SET :=  $\emptyset$ 
10.  for each successor  $s$  of state do
11.    if ( $s = s_{goal}$ ) then return depth + 1
12.    if ( $s \notin hashtable$ ) then SET := SET  $\cup$  { $s$ }
13.  end for
14.  if (SET =  $\emptyset$ ) then return  $\infty$ 
15.  if (hashtable has only one empty slot) then return  $\infty$ 
16.  best := arg min $_{s \in SET}$  {  $h(s)$  }
17.  if (discrepancies = 0) then
18.    hashtable := hashtable  $\cup$  {best}
19.    pathlength := GLDSprobe(best, depth + 1, 0,  $h(\cdot)$ )
20.  else
21.    SET := SET \ {best}
22.    while (SET  $\neq \emptyset$ ) do
23.      state := arg min $_{s \in SET}$  {  $h(s)$  }
24.      SET := SET \ {state}
25.      hashtable := hashtable  $\cup$  {state}
26.      pathlength := GLDSprobe(state, depth + 1, discrepancies - 1,
27.         $h(\cdot)$ )
28.      hashtable := hashtable \ {state}
29.      if (pathlength <  $\infty$ ) then return pathlength
30.    end while
31.    hashtable := hashtable  $\cup$  {best}
32.    pathlength := LDSprobe(best, depth + 1, discrepancies,  $h(\cdot)$ )
33.    hashtable := hashtable \ {best}
34.  return pathlength

```

Figure 3: Generalized Limited Discrepancy Search

sumed at the current level by following the heuristic recommendation, Line 16). LDS terminates when it generates the goal state (Line 11).

3.2 Generalized Limited Discrepancy Search

To apply LDS to beam search, we need to generalize it from binary trees to arbitrary graphs. First, LDS must be able to handle branching factors that are nonuniform and larger than two. Second, LDS must be able to avoid cycles. GLDS (Generalized Limited Discrepancy Search) addresses the first issue by picking a successor s of a given state with a smallest heuristic value $h(s)$. Choosing any other successor is counted as one discrepancy, and the successors are tried from left to right. GLDS addresses the second issue by performing cycle detection with a hash table and not generating successors that are already in the hash table. Figure 3 shows the pseudo code for GLDS. The top-level function GLDS() repeatedly performs generalized limited discrepancy searches from the start state (Line 4) by calling GLDSprobe() with an increasing number of allowed discrepancies (Line 6), starting with no discrepancy (Line 2). GLDSprobe() performs a generalized

```

1. procedure BULB( $s_{start}, h(\cdot), B$ ): path length
2.   discrepancies := 0;  $g(s_{start}) := 0$ ; hashtable := { $s_{start}$ }
3.   while (true) do
4.     pathlength := BULBprobe(0, discrepancies,  $h(\cdot), B$ )
5.     if (pathlength <  $\infty$ ) then return pathlength
6.     discrepancies := discrepancies + 1
7.   end while
8. procedure BULBprobe(depth, discrepancies,  $h(\cdot), B$ ): path length
9.   (SLICE, value, index) := nextSlice(depth, 0,  $h(\cdot), B$ )
10.  if (value  $\geq 0$ ) then return value
11.  if (discrepancies = 0) then
12.    if (SLICE =  $\emptyset$ ) then return  $\infty$ 
13.    pathlength := BULBprobe(depth + 1, 0,  $h(\cdot), B$ )
14.    for each  $s$  in SLICE do hashtable := hashtable \ { $s$ } end for
15.    return pathlength
16.  else
17.    if (SLICE  $\neq \emptyset$ ) then
18.      for each  $s$  in SLICE do hashtable := hashtable \ { $s$ } end for
19.    while (true) do
20.      (SLICE, value, index) := nextSlice(depth, index,  $h(\cdot), B$ )
21.      if (value  $\geq 0$ ) then
22.        if (value <  $\infty$ ) then return value
23.        else break
24.      if (SLICE =  $\emptyset$ ) then continue
25.      pathlength := BULBprobe(depth + 1, discrepancies - 1,  $h(\cdot), B$ )
26.      for each  $s$  in SLICE do hashtable := hashtable \ { $s$ } end for
27.      if (pathlength <  $\infty$ ) then return pathlength
28.    end while
29.    (SLICE, value, index) := nextSlice(depth, 0,  $h(\cdot), B$ )
30.    if (value  $\geq 0$ ) then return value
31.    if (SLICE =  $\emptyset$ ) then return  $\infty$ 
32.    pathlength := BULBprobe(depth + 1, discrepancies,  $h(\cdot), B$ )
33.    for each  $s$  in SLICE do hashtable := hashtable \ { $s$ } end for
34.    return pathlength
35. procedure nextSlice(depth, index,  $h(\cdot), B$ ): (array of states, integer, integer)
36.   currentlayer := { $s \in hashtable \mid g(s) = depth$ }
37.   SUCCS := generateNewSuccessors(currentlayer,  $h(\cdot)$ )
38.   if ((SUCCS =  $\emptyset$ ) or (index = |SUCCS|)) then return ( $\emptyset, \infty, -1$ )
39.   if ( $s_{goal} \in SUCCS$ ) return ( $\emptyset, depth + 1, -1$ )
40.   SLICE :=  $\emptyset$ ;  $i := index$ 
41.   while (( $i < |SUCCS|$ ) and ((|SLICE| <  $B$ ))) do
42.     if (SUCCS[ $i$ ]  $\notin hashtable$ ) then
43.        $g(SUCCS[i]) := depth$ ; SLICE := SLICE  $\cup$  {SUCCS[ $i$ ]}
44.       hashtable := hashtable  $\cup$  {SUCCS[ $i$ ]}
45.       if (hashtable is full) then
46.         for each  $s$  in SLICE do hashtable := hashtable \ { $s$ } end for
47.         return ( $\emptyset, \infty, -1$ )
48.        $i := i + 1$ 
49.   end while
50.   return (SLICE, -1,  $i$ )
51. procedure generateNewSuccessors(stateset,  $h(\cdot)$ ): array of states
52.   index := 0
53.   for each state in stateset do
54.     for each successor  $s$  of state do
55.       if ( $s \notin hashtable$ ) then
56.         SUCCS[index] :=  $s$ ; index := index + 1
57.   end for
58.   end for
59.   Sort states in SUCCS in order of increasing  $h(\cdot)$ -values
60.   return SUCCS

```

Figure 4: BULB

limited discrepancy search from a given state for a given number of allowed discrepancies. First, it generates all successors of the state that are not already in the hash table (Lines 9-13). It backtracks if the goal state is found (Line 11), there are no successors (Line 14), or the hash table is full (Line 15). Otherwise, it identifies the best successor as one with a smallest heuristic value (Line 16). If the number of allowed discrepancies is zero, then GLDSprobe() calls itself on the best successor with no allowed discrepancies (Line 19). Otherwise, GLDSprobe() calls itself repeatedly on the remaining successors with one less allowed discrepancy (Line 26) and then calls itself on the best successor with the same number of allowed discrepancies (Line 31).

3.3 BULB

BULB (Beam search Using Limited discrepancy Backtracking) combines beam search with GLDS. Figure 4 shows the

Table 2: Taxonomy of Search Methods

beam width	type of backtracking		
	none	chronological	limited discrepancy
1	greedy search (gradient descent)	guided depth-first search	limited discrepancy search (LDS/GLDS)
intermediate values	beam search	depth-first beam search (DB)	beam search using limited discrepancy backtracking (BULB)
∞	breadth-first search	breadth-first search	breadth-first search

pseudo code for BULB. The top-level function `BULB()` is basically identical to `GLDS()`. The function `BULBprobe()` performs beam search with generalized limited discrepancy search for a given number of allowed discrepancies. It first generates the first slice of the next level (Line 9). If the slice contains a goal state, the slice is empty, the subtree has been searched exhaustively, or the hash table (which stores the beam) is full, then it aborts (Lines 10 and 12). If the number of allowed discrepancies is zero (Line 11) and the slice is not empty (Line 12), then `BULBProbe()` calls itself with no allowed discrepancies (Line 13), and clears the hash table of the slice (Line 14). Otherwise, `BULBProbe()` clears the hash table of the slice (Line 18), calls itself repeatedly on the remaining slices with one less allowed discrepancy (Line 25) and then calls itself on the best slice with the same number of allowed discrepancies (Line 32). The function `nextSlice()` generates a successor slice for a slice that is already in the hash table at a given depth. It first locates the given slice (Line 36), generates all successors of its states (Line 37), and then locates the slice of the given index within the successors. It does this by inserting successors into both an empty slice (Line 43) and the hash table (Line 44), starting with the successor at the given index (Line 40), until either B successors have been inserted into the slice or the end of the layer has been reached (Line 41). If the hash table is full (Line 45), then it clears the hash table of the incomplete slice (Line 46) and aborts (Line 47). The function `generateNewSuccessors()` generates the successors s of a given set of states that are not already in the hash table and sorts them in order of increasing heuristic values $h(s)$. (The successors can contain duplicates.)

3.4 Properties of BULB

Heuristic search methods that repeatedly fill up and purge memory can be rather complicated [Chakrabarti *et al.*, 1989; Russell, 1992; Kaindl and Khorsand, 1994; Zhou and Hansen, 2002]. In contrast, BULB is relatively simple because it purges contiguous regions of memory and is only an approximation algorithm that does not necessarily find shortest paths. Table 2 shows a taxonomy of search methods. BULB generalizes beam search to beam search with backtracking, limited discrepancy search to beam widths larger than one, and breadth-first search to beam widths smaller than infinity.

- The memory consumption of BULB is $O(Bd)$, where d is the maximum search tree depth. This is achieved by only storing one slice for each level, which requires BULB to re-generate all successors of the states of a slice every time it backtracks. The resulting small memory consumption allows for deeper searches with wider

beams. (Other linear-space search methods often store the siblings of states as well, which makes it unnecessary to re-generate the successors of states but increases the memory consumption substantially.) BULB is a memory-bounded search method and thus continues its search after memory runs out by purging states from memory, resulting in a complete search method. This means that BULB finds a path as long as there is one with a length of the maximum search tree depth or smaller, which approximately equals M/B , where M is the memory capacity measured by the maximal number of states one can store. BULB thus improves on beam search, which is incomplete, and on breadth-first search, which is complete but whose maximum search tree depth approximately equals $\log_b(M)$, where b is the average branching factor of the search tree, and can thus solve only smaller search problems than BULB.

- The runtime of BULB is often small. In fact, BULB frequently finds a path without any backtracking or with only a very limited amount of backtracking. It also eliminates all cycles (loops) and some transpositions (different paths from the start state to a given state), which are often responsible for the large runtimes of depth-first search. BULB, as a generalization of breadth-first search, eliminates all cycles since it never generates states that are already in the hash table. BULB does not make any effort at eliminating transpositions. Nevertheless, BULB, as a generalization of beam search, eliminates some transpositions since it does not re-expand states that are already in its beam.

4 Experimental Evaluation

We now present an experimental study of BULB in three standard benchmark domains: the N-Puzzle, the 4-Peg Towers of Hanoi and the Rubik’s Cube. Note that our figures show graphs only for search methods that were able to solve all random problem instances since we are interested in increasing the number of solved problem instances to one hundred percent. Additional results are reported in [Furcy, 2004].

4.1 N-Puzzle

Our first benchmark domain was the N-Puzzle, as already described in the context of Table 1. Beam search solved all problem instances of small N-Puzzles with a small average path length and did so in fractions of a second. It is therefore not surprising that neither DB nor BULB significantly improved on beam search for N smaller than 48. The situation was different for the 48-Puzzle. DB did not significantly improve on beam search for the 48-Puzzle either. On the other hand, BULB was able to solve all problem instances with a beam width of 10,000 while beam search was only able to solve all problem instances with beam widths of 10 or smaller. BULB was able to find paths of average length 440 with this beam width while beam search was only able to find paths of average length 11,737 with beam widths that allowed it to solve all problem instances (for $B = 5$, which is not shown in Table 1). Thus, BULB was able to reduce

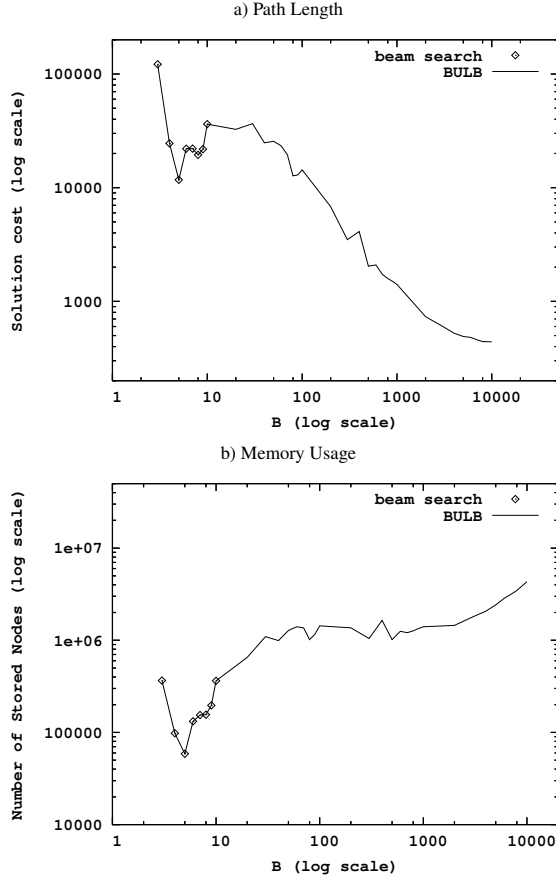


Figure 5: BULB on the 48-Puzzle (B Varies)

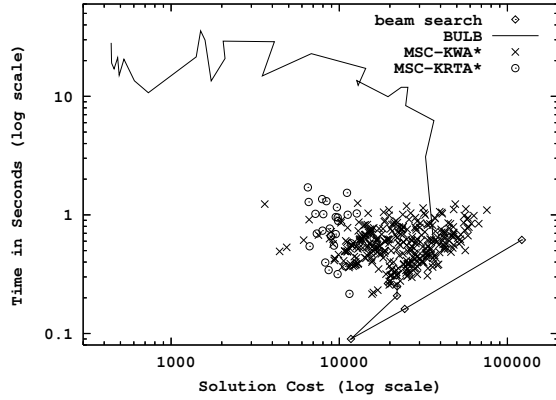


Figure 6: Search Methods on the 48-Puzzle (B Varies)

the path length or, synonymously, solution cost by a factor of about 25. At the same time, the average runtime of BULB was still on the order of 30 seconds on a Pentium 4 PC clocked at a 2.2 GHz. Figure 5 contains detailed data points about BULB. Since BULB generates states in exactly the same order as beam search, the graphs of BULB simply extend the ones of beam search to larger beam widths. For the 80-Puzzle and a memory capacity of 3,000,000 states, there

Table 3: Beam Search on the Towers of Hanoi

B	Path Length	Generated States	Stored States	Runtime (Seconds)	Problems Solved
1	N/A	N/A	N/A	N/A	0 %
5	37,775.12	730,901	188,860	0.306	68 %
10	33,489.26	1,261,982	334,850	0.581	46 %
50	8,468.59	1,619,300	423,103	0.900	68 %
100	4,629.57	1,784,654	462,443	1.012	70 %
500	1,363.59	2,632,408	678,792	1.855	74 %
1,000	831.90	3,196,242	824,784	2.388	58 %
5,000	N/A	N/A	N/A	N/A	0 %

was no beam width that allowed beam search to solve all 50 random problem instances but BULB was able to solve them for a wide range of beam widths. The smallest average runtime of BULB with a beam width that solved all problem instances was about 12 seconds. It was obtained with a beam width of 6 and resulted in an average path length of about 181,000. A larger beam width of 20,000, that still solved all problem instances, increased its average runtime to about 120 seconds but reduced the average path length to 1,130, which is less than 5 times the shortest path length. Figure 6 shows that BULB was also able to improve the average path length of two multi-state commitment search methods for the 48-Puzzle by at least one order of magnitude with an average runtime of only about 20 seconds. These alternatives to beam search are MSC-KWA* [Furcy and Koenig, 2005], a combination of KWA* [Felner *et al.*, 2003] and MSC-WA* [Kitamura *et al.*, 1998], and MSC-KRTA* [Furcy, 2004], a combination of KWA* [Felner *et al.*, 2003], MSC-WA* [Kitamura *et al.*, 1998] and RTA* [Korf, 1990].

4.2 Towers of Hanoi

Our second benchmark domain was the 4-Peg Towers of Hanoi. We created 50 random problem instances with 22 disks in which the goal state had all disks stacked on one peg. We set the memory capacity to 1,000,000 states and used a pattern database similar to that of [Felner *et al.*, 2004] as the heuristic function. Table 3 shows that, similarly to the 48-Puzzle, beam search with large beam widths solved many problem instances, and the average length of the paths found was short. However, there was no beam width that allowed beam search to solve all 50 random problem instances (which is the reason why Figure 7 contains no graphs for beam search) but BULB was able to solve them for a wide range of beam widths. The smallest average runtime of BULB with a beam width that solved all problem instances was about 1.5 seconds. It was obtained with a beam width of 40 and resulted in an average path length of about 10,000. A larger beam width of 1,000, that still solved all problem instances, increased its average runtime to about 7 seconds but reduced the average path length to about 870. Figure 7 contains detailed data points about BULB.

4.3 Rubik's Cube

Our third benchmark domain was the Rubik's Cube. We created 50 random problem instances in which the goal state was the original configuration of the cube. We set the memory capacity to 1,000,000 states and used the pattern databases from [Korf, 1997] as the heuristic function. Beam search was only able to find paths of average length 55.18 with beam widths

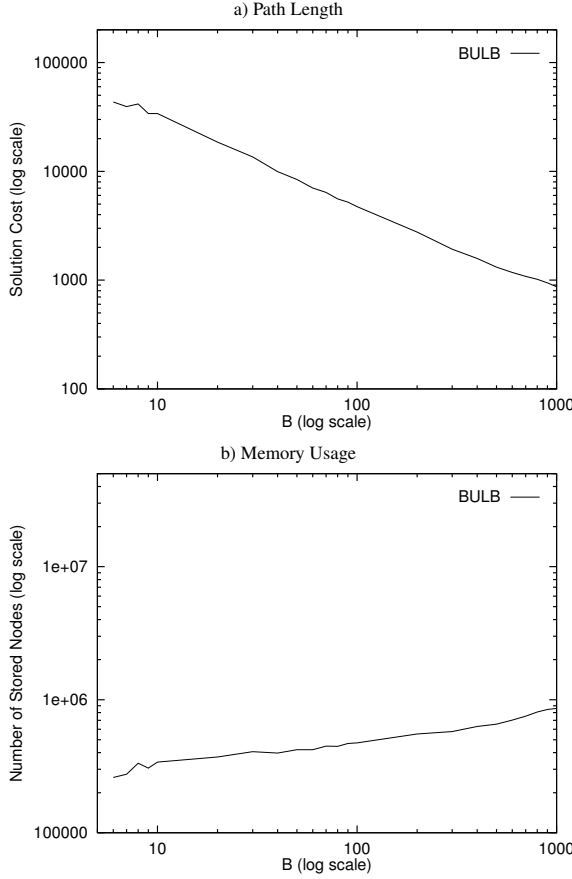


Figure 7: BULB on the Towers of Hanoi (B Varies)

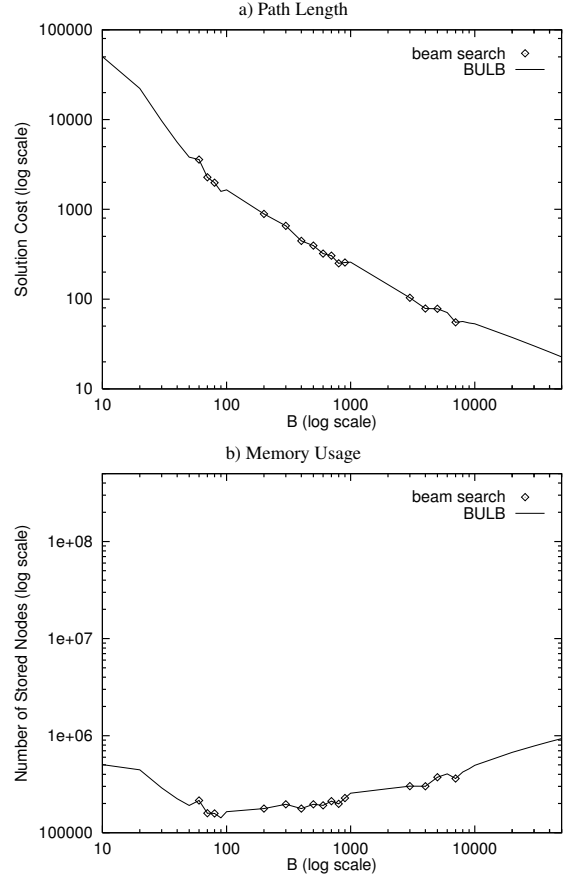


Figure 8: BULB on the Rubik's Cube (B Varies)

Table 4: Beam Search on the Rubik's Cube

B	Path Length	Generated States	Stored States	Runtime (Seconds)	Problems Solved
10	53,909.26	7,146,960	539,084	14.965	38 %
50	3,882.35	2,570,677	194,036	5.224	98 %
100	1,679.76	2,223,466	167,795	4.349	98 %
500	394.84	2,596,065	196,182	5.168	100 %
1,000	259.80	3,398,726	257,058	6.798	98 %
5,000	78.02	4,895,297	373,602	9.977	100 %
10,000	52.33	6,332,050	486,767	13.087	98 %
50,000	21.40	10,848,794	866,741	23.256	10 %
100,000	N/A	N/A	N/A	N/A	0 %

that allowed it to solve all problem instances (for $B = 7,000$, which is not shown in Table 4), which is similar to the average path length found by a recent powerful Rubik's Cube solver based on macro-operators, even though this Rubik's Cube solver uses both a larger number of pattern databases to build the macro-operators and a post-processing step on the paths it finds [Hernádvolgyi, 2001]. Beam search solved all 50 problem instances for several beam widths but BULB was able to solve them for all tested beam widths. BULB with a beam width of 30,000 solved all problem instances and found an average path length of 30.14 with an average runtime of about 40 seconds. This average path length is already smaller than the one of the Rubik's Cube solver mentioned above, even though BULB is a domain-independent search method without any pre- or post-processing and used only about 120

MBytes of memory in our experiments (86 MBytes for the pattern database and 32 MBytes for the hash table). BULB with a beam width of 40,000 found a path of length 25.78 with an average runtime of about 2 minutes. A larger beam width of 50,000, that still solved all problem instances, increased its average runtime to about 7 minutes but reduced the average path length to about 22.74. Figure 8 contains detailed data points about BULB.

5 Related Work

Existing variants of beam search differ from BULB in that they either 1) use no backtracking at all or 2) use chronological backtracking. In the first category, diversity beam search [Shell *et al.*, 1994] deals with imperfect heuristic values by introducing diversity at all levels of the search tree. It differs from BULB in that it is incomplete and requires additional knowledge to measure the level of dissimilarity among states. Divide-and-conquer beam search [Zhou and Hansen, 2004] does not store all of the beam in memory. Instead, it purges some of its slices from memory and uses a divide-and-conquer strategy to reconstruct the path after it finds a goal state, which makes backtracking impossible, at least on the parts of the beam that have been purged from memory. In the second category, band search [Chu and Wah, 1992] is the search method most similar to BULB. It differs

from BULB in that it extends beam search with chronological backtracking and is designed for search trees. It does not detect loops and therefore performs best for small search problems. Complete anytime beam search [Zhang, 1998] does not extend beam search but depth-first search. It uses chronological backtracking (with a beam width of one) while iteratively weakening its pruning rule. Like all depth-first search methods, it performs best on finite trees of shallow depths with large goal densities (such as travelling salesperson problems), which are very different from our benchmark domains.

6 Conclusion

In this paper, we developed BULB (Beam search Using Limited discrepancy Backtracking), a memory-bounded search method that generalizes beam search to beam search with backtracking, limited discrepancy search to beam widths larger than one, and breadth-first search to beam widths smaller than infinity. BULB makes beam search complete (provided that there is sufficient memory to store the beam along a shortest path from the start state to a goal state), tends to find shorter paths than beam search because it is able to use larger beam widths without running out of memory, and can be transformed into an admissible anytime algorithm, for example, by letting it continue its search after it has found a path, resulting in an anytime extension of beam search that is similar in spirit to the anytime extension of WA* described in [Hansen *et al.*, 1997]. BULB outperformed beam search and variants of WA* in our experiments, solved all of our test problems for the 80-Puzzle, and resulted in a state-of-the-art Rubik's Cube solver without any pre- or post-processing, even though it is a domain-independent search method. It is future work to enhance BULB with more complex variants of beam search, for example, variants that change the beam width during the search. It is also future work to enhance BULB with more complex variants of backtracking, for example, variants that give a higher priority to decisions close to the top of the search tree than decisions close to the bottom of the search tree, variants that use depth-bounded discrepancy search [Walsh, 1997] or variants that calculate the discrepancies differently.

References

- [Bisiani, 1987] R. Bisiani. Beam search. In S. Shapiro, editor, *Encyclopedia of Artificial Intelligence*, pages 56–58. Wiley & Sons, 1987.
- [Chakrabarti *et al.*, 1989] P. Chakrabarti, S. Ghose, A. Acharya, and S. de Sarkar. Heuristic search in restricted memory. *Artificial Intelligence*, 41(2):197–221, December 1989.
- [Chu and Wah, 1992] L.-C. Chu and B. Wah. Band search: An efficient alternative to guided best-first search. In *Proceedings of the International Conference on Tools for Artificial Intelligence*, pages 154–161. IEEE Computer Society Press, November 1992.
- [Felner *et al.*, 2003] A. Felner, S. Kraus, and R. Korf. KBFS: K-best-first search. *Annals of Mathematics and Artificial Intelligence*, 39:19–39, 2003.
- [Felner *et al.*, 2004] A. Felner, R. Meshulam, R. Holte, and R. Korf. Compressing pattern databases. In *Proceedings of the National Conference on Artificial Intelligence*, pages 638–643, 2004.
- [Furcy and Koenig, 2005] D. Furcy and S. Koenig. Scaling up WA* with commitment and diversity. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 2005.
- [Furcy, 2004] D. Furcy. *Speeding up the Convergence of Online Heuristic Search and Scaling Up Offline Heuristic Search*. PhD thesis, College of Computing, Georgia Institute of Technology, Atlanta (Georgia), 2004. Available as Technical Report GIT-COGSCI-2004/04.
- [Hansen *et al.*, 1997] E. Hansen, S. Zilberstein, and V. Danilchenko. Anytime heuristic search: First results. Technical Report CMPSCI 97–50, Department of Computer Science, University of Massachusetts, Amherst (Massachusetts), 1997.
- [Harvey and Ginsberg, 1995] W. Harvey and M. Ginsberg. Limited discrepancy search. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 607–615, 1995.
- [Hernádvölgyi, 2001] I. Hernádvölgyi. Searching for macro operators with automatically generated heuristics. In *Proceedings of the Canadian Conference on Artificial Intelligence*, pages 194–203, 2001.
- [Kaindl and Khorsand, 1994] H. Kaindl and A. Khorsand. Memory-bounded bidirectional search. In *Proceedings of the National Conference on Artificial Intelligence*, pages 1359–1364, 1994.
- [Kitamura *et al.*, 1998] Y. Kitamura, M. Yokoo, T. Miyaji, and S. Tatsumi. Multi-state commitment search. In *Proceedings of the International Conference on Tools for Artificial Intelligence*, pages 431–439, 1998.
- [Korf and Taylor, 1996] R. Korf and L. Taylor. Finding optimal solutions to the twenty-four puzzle. In *Proceedings of the National Conference on Artificial Intelligence*, pages 1202–1207, 1996.
- [Korf, 1990] R. Korf. Real-time heuristic search. *Artificial Intelligence*, 42(2-3):189–211, 1990.
- [Korf, 1993] R. Korf. Linear-space best-first search. *Artificial Intelligence*, 62(1):41–78, 1993.
- [Korf, 1997] R. Korf. Finding optimal solutions to Rubik's cube using pattern databases. In *Proceedings of the National Conference on Artificial Intelligence*, pages 700–705, 1997.
- [Pearl, 1985] J. Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, 1985.
- [Russell, 1992] S. Russell. Efficient memory-bounded search methods. In *Proceedings of the European Conference on Artificial Intelligence*, pages 1–5, 1992.
- [Shell *et al.*, 1994] P. Shell, J. Rubio, and G. Barro. Improving search through diversity. In *Proceedings of the National Conference on Artificial Intelligence*, pages 1323–1328, 1994.
- [Walsh, 1997] T. Walsh. Depth-bounded discrepancy search. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1388–1393, 1997.
- [Zhang, 1998] W. Zhang. Complete anytime beam search. In *Proceedings of the National Conference on Artificial Intelligence*, pages 425–430, 1998.
- [Zhou and Hansen, 2002] R. Zhou and E. Hansen. Memory-bounded A* graph search. In *International FLAIRS Conference*, 2002.
- [Zhou and Hansen, 2004] R. Zhou and E. Hansen. Breadth-first heuristic search. In *Proceedings of the International Conference on Automated Planning and Scheduling*, pages 92–100, 2004.

Generalized Amazons is PSPACE-Complete

Timothy Furtak¹, Masashi Kiyomi², Takeaki Uno³, Michael Buro⁴

^{1,4} Department of Computing Science, University of Alberta, Edmonton, Canada. email: {¹furtak,⁴mburo}@cs.ualberta.ca

^{2,3} National Institute of Informatics, Japan. email: ²masashi@grad.nii.ac.jp, ³uno@nii.jp

Abstract

Amazons is a perfect information board game with simple rules and large branching factors. Two players alternately move chess queen-like pieces and block squares on a 10×10 playing field. The player who makes the last move wins. Amazons endgames usually decompose into independent subgames. Therefore, the game is a natural testbed for combinatorial game theory. It was known that determining the winner of simple generalized Amazons endgames is NP-equivalent. This paper presents two proofs for the PSPACE-completeness of the generalized version of the full game.

1 Introduction

Amazons is a two-person perfect information board game that combines elements of chess and Go. Both players start with four pieces called amazons which are symmetrically positioned at the edges of an otherwise empty 10×10 square board. Moves consist of picking an amazon of the player's colour, moving it like a chess queen, and then shooting an arrow straight in one of eight directions from the amazon's destination square to an empty square. This square becomes blocked for the remainder of the game and no amazon or arrow can pass it. Arrows are not allowed to pass amazons either and amazons cannot be captured. Blocking squares is mandatory. The game proceeds in turns and the first player without any legal move loses. Fig. 1 shows two typical Amazons positions.

The game of Amazons is interesting for AI research for at least the following reasons:

- Compared with other popular games its rule set is simple — and yet, it leads to interesting strategic concepts which need to be addressed when designing evaluation functions [Lieberum, 2005].
- The number of legal moves in Amazons positions on the standard board is often in the hundreds or even thousands. This puts pressure on sound backward pruning search techniques such as the alpha-beta algorithm, because the number of positions to be searched even at small depths is astronomical. Amazons can therefore

serve as a testbed for research on selective search and adversarial planning.

- Like Hex, Go-moku, Tic-Tac-Toe, and Othello, Amazons has the property of being monotonically increasing (each move blocks one square), so the underlying structure of the game is a directed acyclic graph, rather than a general graph. This eliminates some potentially complicating variables for scientific research. Also like Hex, Amazons cannot end in a draw.
- Endgames regularly decompose into independent subgames (Fig 1). At this stage, combinatorial game theory can be applied to vastly reduce the search space, and thus speed up good move selection compared with full-board analyses. Hence, Amazons is a suitable test-domain for new ideas in this area. See for instance [Berlekamp, 2000; Müller and Tegs, 2002; Snatzke, 2004].

Establishing the computational complexity of various problems associated with generalized games is also interesting. For example, proving NP- or PSPACE-hardness is convincing evidence that a problem does not possess a simple solution in general (although effective methods may exist for specific instances).

Generalizing the standard Amazons game is straight-

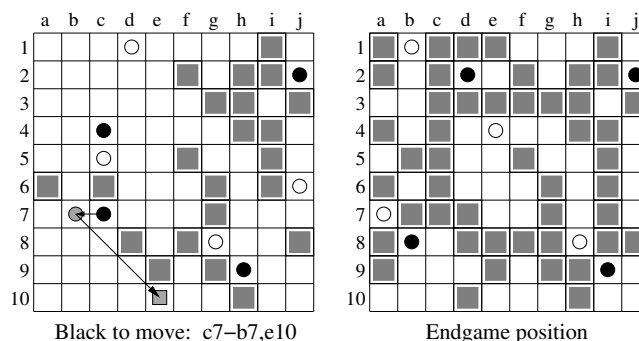


Figure 1: Typical Amazons positions. On the left, the black amazon at c7 is about to move to b7 and block e10 to prevent the white amazon at g8 from entering the SW corner area. On the right, Black can make 24 unopposed moves, whereas White can make only 20, so Black can win regardless of who is to move.

forward: we allow arbitrary square-shaped boards ($n \times n$) with at least one and up to n^2 amazons. It is known that solving generalized Amazons puzzles (i.e. determining whether in a single-player Amazons game the player can make at least k consecutive moves) is NP-complete and determining the winner of simple generalized Amazons endgames — in which players do not interact anymore — is NP-equivalent [Buro, 2000]. In what follows, we present two proofs for the PSPACE-hardness of deciding the winner of generalized¹ Amazons positions. While the first proof is more intuitive, the second one leads to a stronger results, namely that the problem remains PSPACE-complete even if each player has only one amazon.

2 Hex Reduction

In this section we show that determining the winner in Hex positions, which is known to be PSPACE-complete, can be reduced to the related Amazons problem. We begin with the definition of the sets we like to relate to each other:

Definition 1:

- **AMAZONS** is a set of succinct encodings of all generalized Amazons positions with a winning strategy for Black (to move).
- **HEX** is a set of succinct encodings of all Hex positions with a winning strategy for Black (to move).

Theorem 1: $\text{HEX} \leq_p \text{AMAZONS}$.

Corollary 1: AMAZONS is PSPACE-complete.

Proof: [Reisch, 1981] shows that HEX is PSPACE-complete by reducing QBF to HEX. Moreover, $\text{AMAZONS} \in \text{PSPACE}$, which is obvious by the existence of a depth-first minimax search algorithm that decides the winner of a given Amazons position in space polynomial in the length of its encoding. Note, that the number of moves in an Amazons game played on an $n \times n$ board is at most n^2 . $\square$

Proof of Theorem 1: We describe a function f that maps arbitrary code words w into the encoding of an associated Amazons position with the following properties:

1. If w encodes a Hex position, then the first player (Black) in $\text{decode}(w)$ has a Hex winning strategy if and only if the first player in $\text{decode}(f(w))$ (Black) has an Amazons winning strategy.
2. Otherwise, $f(w)$ encodes a fixed Amazons position with a second player (White) winning strategy.
3. There exists a Turing machine F and a polynomial p such that for all w , F started on input w computes $f(w)$ in at most $p(|w|)$ steps.

It follows that $w \in \text{HEX} \Leftrightarrow f(w) \in \text{AMAZONS}$ and $\text{HEX} \leq_p \text{AMAZONS}$. We now describe the transformation f .

From a given $n \times n$ Hex position, each of the n^2 hex tiles are represented by a rectangular room containing one Black

¹We drop “generalized” from now on when the context is clear.

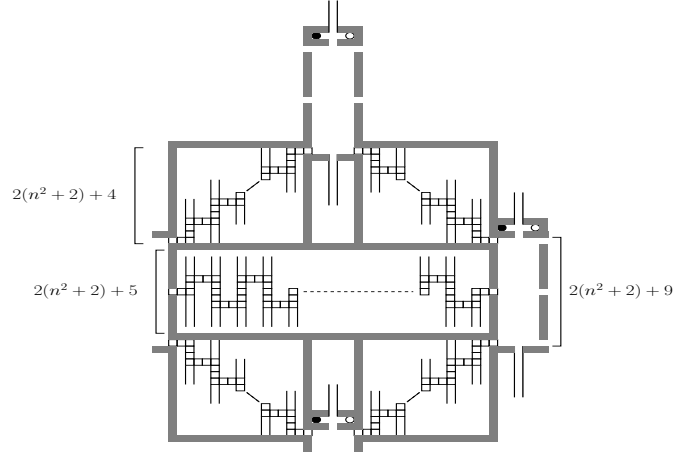


Figure 2: The arrangement of rooms relative to each other and the connecting corridors. There is slack space for arrows in each corridor and at the top and bottom of each room sufficient to allow a large number of amazons to pass through.

and one White amazon (Fig. 2,3). These amazons are placed across from each other so that each is able to contain the other in one move, and so gain control of the room. Occupied tiles on the Hex board are represented by removing the opposing piece. In this way the first stage of the transformed game is effectively a game of Hex where players alternately select rooms, blocking the opponent’s amazons. Indeed, if any player does not take control of a room when such a move is available then the opposing player may take a room “for free,” which in Hex (and similarly here) can only help.

The rooms are connected by corridors which require a certain minimum number of steps to traverse. This traversal time for each corridor is chosen such that no adjacent rooms may affect either player from securing a room. Each of the n^2 rooms may be sealed off from enemy rooms by blocking off the appropriate corridors. As there are at most 6 entrances to each room and each can be blocked in one move, minimum corridor traversal times of $7n^2$ are sufficient.

To force the outcome of the Amazons game to be equivalent to that of the initial Hex game, White must win if and only if he can form a connecting path from top to bottom. This is accomplished by placing an army of $n^2 + 2$ White amazons at the bottom of the board with a path to all of the bottom rooms (Fig. 4). As with adjacent rooms, these amazons are also sufficiently far away that they cannot influence the initial choosing of rooms.

At the top of the board are $2n^2 + 5$ strips of open space. The size of these strips is chosen large enough so that each is larger than the rest of the board when ignoring the other strips. This ensures that the outcome of the game is entirely determined by which player is able to control a majority of the strips. Strips are positioned such that once a White amazon has captured one, it is unable to capture any others. At the top of each strip is a long path connecting to a single Black amazon (Fig. 5). The path is made long enough so that while the amazon may eventually reach the strip, the number of moves required to do so is greater than that needed by White to cap-

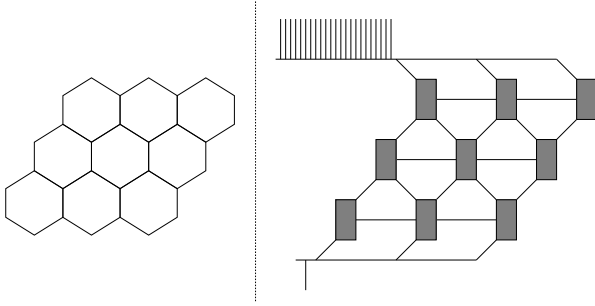


Figure 3: Sample transformation of a 3×3 Hex board.

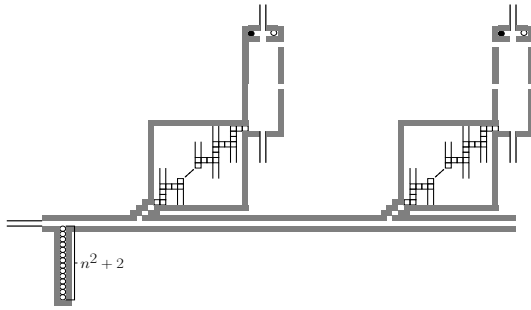


Figure 4: White's army at the bottom of the board. The first amazon can block off paths to enemy rooms.

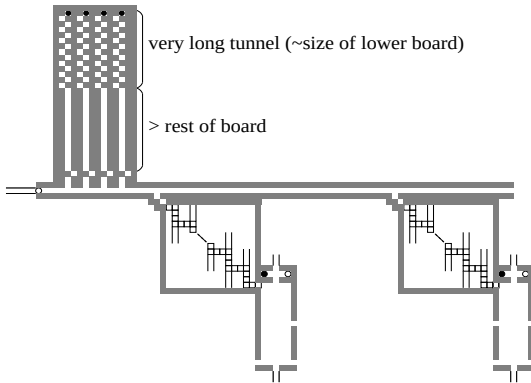


Figure 5: $2n^2 + 5$ large strips of space at the top of the board. White has a gatekeeper to prevent enemy rooms from reaching them. Black has an amazon at the top of each strip that will eventually be able to take control of it if White cannot.

ture as many strips as he can. This is the time needed to decide the control of each room, secure those rooms, and to move the army from the bottom of the board to each of the strips. A very loose upper bound for this distance is the number of free squares in the lower section of the board, everything beneath the strips. In this manner Black is eventually able to take control of any strips that White cannot.

If the first player (White) has a winning strategy in the initial Hex position that player may capture a set of rooms such that a path exists from the bottom of the playfield to the top. By blocking all corridors to hostile rooms he may isolate this

path from enemy influence. Since there are no draws in Hex, a winning strategy for the second player is equivalent to preventing White from forming such a sequence of rooms.

If White can form a connecting path then the one White amazon at the top and the army (of size $n^2 + 2$) at the bottom may be moved one at a time to take control of $n^2 + 3$ strips, a majority allowing White to win the game.

If White cannot form a connecting path then the number of strips he may capture is at most $n^2 + 1$: the number of White amazons on the board not counting the army. Black will then eventually be able to take control of the remaining strips with the amazon at the top of each. This gives Black a majority of free space allowing him to win. The winner of this game is therefore the same as for the original Hex position.

The connecting corridors must allow for the entire army to pass through them. The given structure requires long slack tunnels so that arrows may be fired while moving. These tunnels must be at least as long as the size of the army, $n^2 + 2$. Not counting the boundary walls, the height of the horizontal corridors then is $2(n^2 + 2) + 5$, and the diagonal corridors one less. Ensuring that the corridors are long enough to prevent room interference is a simple matter of adding additional bends in the path, with each bend increasing the travel time by 2. Since each corridor only needs to be $7n^2$ moves long constructing this new board is clearly polynomial.

Each room is a rectangle with slack tunnels running through the middle. This tunnel is easily sufficient with almost twice the space of those in the diagonal corridors. Moving an amazon through a room is straightforward: move to the center column when entering, then move to the exit row, and finally into the next corridor. To enter and exit on the same row simply move straight across. The initial amazon in the room may be considered as part of the army or moved out of the way to the side.

Rooms and corridors may clearly be arranged without conflict (Fig. 2). The entire transformation takes polynomial time in the size of the original Hex board. The strategy to move the army to the top of the board when a connecting series of rooms exists is then trivial. It is also clearly impossible to do this when a connecting series of rooms does not exist. $\square$

3 Geography Reduction

Here we show a stronger result, namely that even if the number of Amazons is bounded by a constant, the problem remains PSPACE-complete. We again start with definitions of the sets we consider:

Definition 2:

- **AMAZONS-2** is a set of succinct encodings of all $n \times n$ Amazons positions with exactly one Black and one White amazon and a winning strategy for Black (to move).
- **GEOGRAPHY-BP3** is a set of succinct encodings of GEOGRAPHY positions — consisting of a directed bipartite planar graph, whose node types are shown in Fig. 7,8,9, and a distinguished start node (Fig. 6) with two leaving edges — for which the first player has a winning strategy.



Figure 6: Start vertex.



Figure 7: Path vertex.

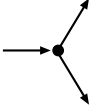


Figure 8: Branch vertex.

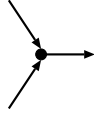


Figure 9: Join vertex.

Theorem 2: $\text{GEOGRAPHY-BP3} \leq_p \text{AMAZONS-2}$.

Corollary 2: AMAZONS-2 and AMAZONS are PSPACE-complete.

Proof: Analog to Proof of Corollary 1. Note that $\text{AMAZONS-2} \leq_p \text{AMAZONS}$. $\square$

Proof of Theorem 2: Geography is a game played by two players on a given directed graph with a marked start node. The players alternately put a marker on any unmarked node from which there exists an edge from the node marked last. The first player who cannot move loses. Note that in case of GEOGRAPHY-BP3 , the graph is bipartite and planar with node types that are shown above. The game always ends in a vertex with two entering edges and one leaving edge at the second visit of that vertex. [Lichtenstein and Sipser, 1980] show that GEOGRAPHY-BP3 is PSPACE-complete.

We present a polynomial-time transformation that maps input words — which are now encodings of planar bipartite graphs with a start marker — into encodings of Amazons positions with two amazons, such that the first player wins the geography game if and only if the first player wins the associated Amazons game. We emulate the Geography game with two amazons — one for each player — and with blocks that fill the board with the necessary structure. The structure has two separate paths for each edge of the given graph. One path (Fig. 10) is for player one and the other path is for player two. The two amazons simultaneously proceed along the paths as quickly as possible, otherwise one player locks up the other player. We construct the structure such that player one winning in generalized geography is equivalent to the amazon of player one being able to trap the amazon of player two. We embed the given directed graph in the board using a pair of paths to represent an edge. To embed the vertices we use two gadgets that force the players to proceed along the paths in pairs and in the same direction as the original directed edges.

Putting a marker on a vertex in GEOGRAPHY-BP3 corresponds to the two amazons going from a vertex gadget to another vertex gadget. Since each vertex gadget can be used

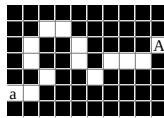


Figure 10: Path example.

only once, vertices cannot be marked twice. We force the two amazons to go along the same paths. If one amazon advances toward a path different from the path chosen by the acting player (the player placing the marker) then the amazon will be locked out by the acting player at some gadget.

We implement a path, which has a one-square width, as blank squares with surrounding blocked squares, as shown in Fig. 10. We assume an amazon is currently at the left entrance a and proceeds to the right exit A . At each player's respective turn, the amazon is moved forward and blocks a square behind it. If the player blocks a square in front of him the amazon becomes trapped. The amazons need to travel at maximum speed, otherwise they will be locked out by the opponent's amazon at some point. In Fig. 10, the amazon needs at least 12 moves to travel from the entrance to the exit. The amazon can also get there in 13 moves. We use gadgets that force the amazons to travel in the minimal numbers of moves (in this case 12). Thus, amazons have to traverse any straight line section of a path in one move. If we want to force an amazon to take several moves, we bend the path an appropriate number of times as shown in Fig. 10. Moreover, it is obvious that we can change the direction a path aims as we like. We now discuss the sub-gadgets shown in Fig. 11 to 14.

Type 1 top speed gadget (Fig. 11). Assume the amazon for player X (denoted X) comes to the bottom left entrance, a , and the amazon for player Y (Y) comes to the bottom right entrance, b . This gadget forces Y to come to the entrance as quickly as possible. That is, if Y comes to the entrance just after X comes to the entrance (we say that they arrive at the entrances at the same time in this situation), then X cannot trap Y and X should exit this gadget at A , and Y should exit at B . If Y comes to this gadget late then X can lock Y out in 3 moves by moving to E , C , then D and blocking F . Thus Y should come to the entrance as quickly as possible. If Y comes to entrance b at the same time X enters, Y can move to F and block D . Thus, Y cannot be trapped. In this case both X and Y can exit this gadget in five moves; X 's exit is A , Y 's exit is B . Note that if we use two paths to connect this gadget to a symmetrical one, we can force both X and Y to move at top speed along those paths.

Type 2 top speed gadget (Fig. 12). If X enters this gadget from a and Y enters from b (resp. d and c), the gadget forces X to proceed to exit A and Y to proceed to exit B (resp. D and C). Otherwise the player being late will be trapped by the other player. This gadget is somewhat complex, so first we will simply refer to its functionality. Assume amazon X comes to the entrance a and amazon Y comes to the entrance b (resp. d and c). This gadget forces Y to come to the entrance

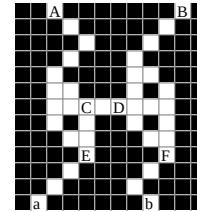


Figure 11: Top speed gadget (type 1).

as quickly as possible. If Y 's entrance is b , this is the same as type 1 top speed gadget, and Y should come to b at the same time that X enters. If Y 's entrance is d , this gadget forces Y to come to d at least before X reaches G , or otherwise Y will be locked out by X . If Y 's entrance is c , then even if Y comes to c at the same time that X enters, X can trap Y . In addition, if X enters from b (resp. c) and Y enters from c (resp. b), Y should come to c at the same time that X enters. Otherwise X can lock Y out. If X 's entrance is a and Y 's entrance is b , X is forced to exit at A and Y is forced to exit at B . Amazons can use the gadget twice, once to enter at a and b and once to enter at d and c . At the first use of the gadget they cannot put blocks to obstruct the next use of the gadget.

Now we detail the amazons' sequence of moves in this gadget. First we will describe the moves when X 's entrance is a and Y 's entrance is b and they come to the entrance at the same time. X moves to N , Y moves to Q ; X then moves to G and blocks I , Y moves to O and blocks H . Then they proceed to their respective exits at top speed. If X blocks a position other than I , Y can then proceed to other exits. This increases the number of moves available to Y , so X should not play this way if he wants to win. Thus we can assume that X blocks I . If Y blocks a position other than H , X can trap Y . Thus Y should block H to prevent being locked out. If X 's entrance is a and Y 's entrance is c and they arrive at the entrance at the same time, X moves to N and Y moves to R . X then moves to G and blocks L , Y moves to P and either blocks F (option 1) or blocks any other position (option 2). In the case of (1), X moves to K and blocks M . In the case of (2), X moves to J and blocks E . In both cases, Y is locked

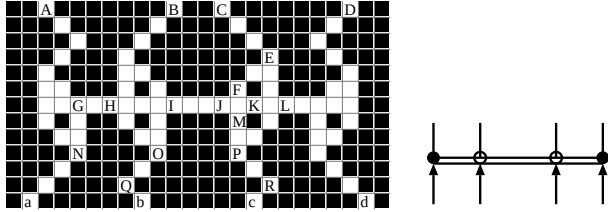


Figure 12: Top speed gadget (type 2).

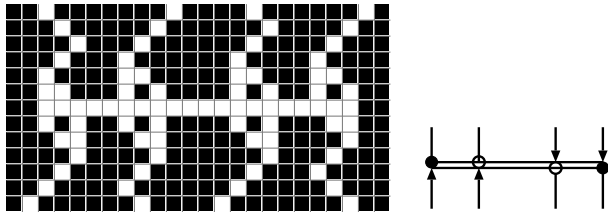


Figure 13: Top speed gadget (type 3).

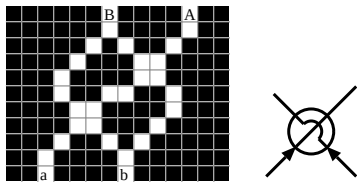


Figure 14: Cross gadget.

out by X . We can check the other properties analogously.

Type 3 top speed gadget (Fig. 13). This gadget is almost identical to the previous gadget. The only difference is the direction of the right hand side paths which have been mirrored.

Cross gadget (Fig. 14). We can cross a path of player X and a path of player Y with this gadget. Assume the entrance for X is a , and the entrance for Y is b . If X and Y come to the gadget at the same time, X is forced to exit the gadget at A and Y is forced to exit this gadget at B . Both X and Y take five moves in that case. If X comes later, then X will be locked out. If Y comes two moves later than X , then Y will be trapped. Similarly, if a player goes to another exit, or tries to obstruct the opponent player, the opponent player can lock the amazon out. In addition, if X comes to the gadget and Y never comes, then if X exits at B , X takes two more moves than if X exits at A . If Y comes to the gadget and X never comes, then if Y exits at A , Y takes two more moves than if Y exits at B . We can easily check these properties.

We now explain the main vertex gadgets corresponding to the four types of vertices in Fig. 6, Fig. 7, Fig. 8 and Fig. 9. It is easy to emulate a vertex in Fig. 7. We do this with two paths, one for each player, on which player one needs one more move to travel from entrance to exit than player two does, effectively reversing which player is next to act. We illustrate gadgets for branching vertices (Fig. 8) and joining vertices (Fig. 9) in Fig. 15 and 16, respectively. We can use the gadget in Fig. 15 for the start vertex (Fig. 6), too. We draw paths for amazon X , who is expected to enter this gadget first, as solid lines and paths for amazon Y as dotted lines. Two amazons always come to the paired entrances at the same time, otherwise an amazon being late will be locked out by the opponent's amazon at the top speed gadget just after the entrance.

Branch gadget (Fig. 15). This gadget corresponds to a vertex with an entering edge and two leaving edges (Fig. 8), from which a player selects one of the leaving edges to follow. Since we can adjust the number of moves amazons require along paths by bending these paths arbitrarily, we set those from X 's entrances to X 's exits and those from Y 's entrances to Y 's exits to be the same length. Because of the cross gadget, X requires two more moves if X wants to take Y 's exit, therefore doing so will result in X being trapped by Y in the type 2 top speed gadget. Thus player X cannot select Y 's path. Y must select the exit in the same direction that X did, otherwise Y is locked out at the type 2 top speed gadget. Note that if X slows down inside the gadget Y can lock X

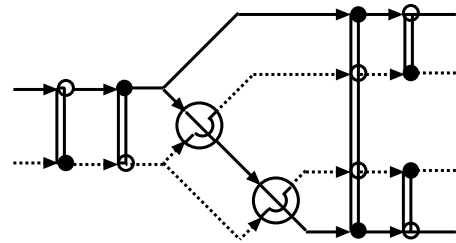


Figure 15: Branch gadget.

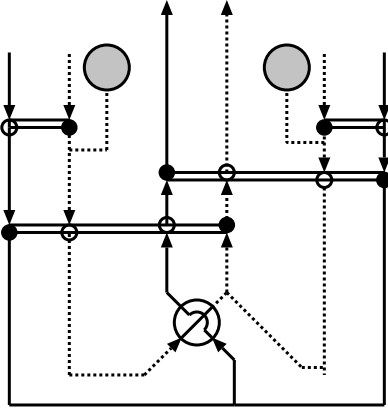


Figure 16: Join gadget.

out in the cross gadget or any of the top speed gadgets. In addition, we can use this gadget for the start vertex (Fig. 6). We place Black and White amazons at the entrance so that the Black amazon would enter the gadget first. The size of this gadget is clearly independent of the input size.

Join gadget (Fig. 16). This gadget corresponds to a vertex with two entering edges and a leaving edge (Fig. 9). Amazons come to this gadget at the left or right entrance. If it is the amazons' first visit to this gadget, the gadget forces the amazons to exit at the top center. If one player changes his way and advances to the entrance not used, the opponent's amazon can trap him in a type 2 top speed gadget in front of the entrance. At the second visit, the amazons cannot exit the gadget, because they can use the left exit of the last top speed gadget only once. Thus, after they pass the first top speed gadget, they have to play Amazons only in the gadget. In this case, Y will win by entering the region denoted by the gray ball, which has larger space than that of the rest of this gadget, and closing the entrance to the ball. Y only has to wait till X fills up the rest of the gadget and becomes locked out by himself. This situation corresponds to that in GEOGRAPHY-BP3 where it is player X 's turn and the adjacent vertices are already marked. The size of this gadget is also independent of the input size.

Using the gadgets described above we can transform any bipartite planar graph — as described in the definition of GEOGRAPHY-BP3 — into an Amazons position such that the player to move wins the Geography game if and only if the first player wins the Amazons game. First we embed the given graph into a grid, taking space polynomially bounded by the input graph's size using a linear time orthogonal graph drawing algorithm (e.g. [Kant, 1996]). Then we replace each grid point which corresponds to a vertex or an edge with a vertex gadget or a path gadget respectively. Each vertex is incident to leaving edges and entering edges at some directions: north, south, west, and east. We can set the entrances and exits of the vertex gadgets equal to the directions of these edges by surrounding them with path gadgets. Further, we sometimes have to use the cross gadget to connect exits to the right entrances in the next gadget. Vertex gadgets and the path gadgets surrounding them are drawn in a board of constant size. An edge corresponding to a grid point can be

also drawn in a constant size, even if it includes a cross gadget. Thus, each grid point is drawn in a board of a constant size, and the size of the Amazons game board is bounded by a polynomial in the size of the input planar graph. To complete the transformation function definition with respect to the three properties stated in the previous proof, input words that do not encode valid graphs are mapped into a fixed Amazons position with two opposing amazons, Black to move and White to win. From the above explanations, it is clear that the run-time of the entire transformation is polynomial in the input length. $\square$

4 Conclusion

We have presented two reductions which imply that determining the winner of generalized Amazons positions is PSPACE-complete. The Hex reduction is appealing because it directly relates two board game complexities. The Geography reduction, however, leads to a stronger result, because the transformation only generates Amazons positions with two opposing amazons, whereas in the Hex reduction, the number of generated amazons cannot be bounded above by a constant.

As this paper was being finished we discovered that Robert Hearn had simultaneously derived similar results [Hearn, 2005]. His reduction shows that generalized Amazons with an unbounded number of amazons is PSPACE-complete.

Acknowledgments

We thank Darse Billings and Robert Hearn for their valuable feedback on a draft paper. Financial support was provided by the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

- [Berlekamp, 2000] E. Berlekamp. Sums of $n \times 2$ amazons. In *Institute of Mathematics, Statistics Lecture Notes – Monograph Series* 35, pages 1–34. Springer, 2000.
- [Buro, 2000] M. Buro. Simple amazons endgames and their connection to hamilton circuits in cubic subgrid graphs. In *Computers and Games Conference*, pages 250–261. Springer, 2000.
- [Hearn, 2005] R.A. Hearn. Amazons is PSPACE-complete. <http://arxiv.org/abs/cs.CC/0502013>, 2005.
- [Kant, 1996] G. Kant. Drawing planar graphs using the canonical ordering. *Algorithmica*, 16(1):4–32, 1996.
- [Lichtenstein and Sipser, 1980] D. Lichtenstein and M. Sipser. Go is polynomial-space hard. *J. of the ACM*, 27:393–401, 1980.
- [Lieberum, 2005] J. Lieberum. An evaluation function for the game of amazons. *Theoretical Computer Science*, accepted, 2005.
- [Müller and Tegős, 2002] M. Müller and T. Tegős. Experiments in computer amazons. In R. Nowakowski, editor, *More Games of No Chance*, pages 243–260. Cambridge University Press, 2002.
- [Reisch, 1981] S. Reisch. Hex ist PSPACE-vollständig. *Acta Informatica*, 15:167–191, 1981.
- [Snatzke, 2004] R.G. Snatzke. *Exhaustive search of databases in the application of Combinatorial Game Theory to the game [of] amazons*. Ph.D. thesis, University of Augsburg, Germany, 2004.

QCSP-Solve: A Solver for Quantified Constraint Satisfaction Problems

Ian P. Gent and Peter Nightingale

School of Computer Science, University of St Andrews, Fife, KY16 9SS, UK

email: {ipg,pn}@dcs.st-and.ac.uk

Kostas Stergiou

Department of Information & Communication Systems Engineering

University of the Aegean, Samos, Greece, email: konsterg@aegean.gr

Abstract

The Quantified Constraint Satisfaction Problem (QCSP) is a generalization of the CSP in which some variables are universally quantified. It has been shown that a solver based on an encoding of QCSP into QBF can outperform the existing direct QCSP approaches by several orders of magnitude. In this paper we introduce an efficient QCSP solver. We show how knowledge learned from the successful encoding of QCSP into QBF can be utilized to enhance the existing QCSP techniques and speed up search by orders of magnitude. We also show how the performance of the solver can be further enhanced by incorporating advanced look-back techniques such as CBJ and solution-directed pruning. Experiments demonstrate that our solver is several orders of magnitude faster than existing direct approaches to QCSP solving, and significantly outperforms approaches based on encoding QCSPs as QBFs.

1 Introduction

The Constraint Satisfaction Problem (CSP) is a very successful paradigm that can be used to model and solve many real-world problems. The CSP has been extended in many ways to deal with problems that contain uncertainty. The Quantified Constraint Satisfaction Problem (QCSP) is an extension in which some of the variables may be universally quantified. For each possible value of such variables, we have to find values for the remaining, existentially quantified, variables so that all the constraints in the problem are satisfied. The QCSP can be used to model PSPACE-complete decision problems from areas such as planning under uncertainty, adversary game playing, and model checking. For example, in game playing we may want to find a winning strategy for all possible moves of the opponent. In a manufacturing problem it may be required that a configuration must be possible for all possible sequences of user choices. Finally, when planning in a safety critical environment, such as a nuclear station, we require that an action is possible for every eventuality.

Interest in QCSPs is growing, following the development of numerous efficient solvers in the closely related area of Quantified Boolean Formulae (QBF or QSAT). However, the

existing direct approaches to solving QCSPs with discrete finite domains, i.e. approaches based on extending CSP techniques, are at an early stage [Bordeaux and Monfroy, 2002; Mamoulis and Stergiou, 2004]. As shown in [Gent *et al.*, 2004], such approaches are significantly outperformed by a QBF solver applied on an encoding of QCSPs into QBFs. Note that, in contrast to QCSPs with finite domains, there is a significant body of work on quantified problems with continuous domains (e.g. [Benhamou and Goualard, 2000; Ratschan, 2002]).

In this paper we introduce an efficient QCSP solver, which we call QCSP-Solve. To develop the solver we first implemented extensions of standard CSP algorithms (FC and MAC) and gradually enhanced them with new capabilities. We started by repeating and analyzing the experiments of [Gent *et al.*, 2004] to discover the features of the QBF solver that account for its effectiveness compared to existing direct approaches. This led us to identify the pure literal rule used by the QBF solver as the main factor contributing to its efficiency. We devised the QCSP analogue of the pure literal rule and incorporated it into our basic solver. As a result we achieved a speed-up of several orders of magnitude. We then continued the development of QCSP-Solve by adding intelligent look-back techniques such as conflict-based backjumping and solution-directed pruning which also offer a significant speed-up. Finally, we implemented a symmetry breaking method based on value interchangeability.

The experimental evaluation of algorithms for QCSPs is difficult for two reasons: first, due to the young age of the area, there is a lack of benchmarks; and second, the generalization of known random generation models from related areas (CSP and QBF) can lead to flawed models. For example in [Gent *et al.*, 2004] it was noted that the generation method used in [Mamoulis and Stergiou, 2004] suffers from a flaw that makes all generated instances insoluble even for small problems sizes. This flaw is also present in other random generation methods. We propose a random generation model that, while creating hard instances, can be used to control the probability of the flaw discovered in [Gent *et al.*, 2004]. For certain parameter settings all generated instances are guaranteed to be unflawed. On problems created using this model, QCSP-Solve is several orders of magnitude faster than the existing QCSP algorithms, and also significantly outperforms the QBF encoding based method of [Gent *et al.*, 2004].

2 Preliminaries

In standard CSPs all variables are existentially quantified. QCSPs are more expressive than CSPs in that they allow universally quantified variables. They enable the formulation of problems where all contingencies must be allowed for. We now give a formal definition.

Definition 1 A *Quantified Constraint Satisfaction Problem* (QCSP) is a formula of the form QC where Q is a sequence of quantifiers $Q_1x_1 \dots Q_nx_n$, where each Q_i quantifies ($\exists$ or $\forall$) a variable x_i and each variable occurs exactly once in the sequence. C is a conjunction of constraints $(c_1 \wedge \dots \wedge c_m)$ where each c_i involves some variables among $x_1, \dots, x_n$.

The semantics of a QCSP QC can be defined recursively as follows. If C is empty then the problem is true. If Q is of the form $\exists x_1 Q_2x_2 \dots Q_nx_n$ then QC is true iff there exists some value $a \in D(x_1)$ such that $Q_2x_2 \dots Q_nx_n C[(x_1, a)]$ ¹ is true. If Q is of the form $\forall x_1 Q_2x_2 \dots Q_nx_n$ then QC is true iff for each value $a \in D(x_1)$, $Q_2x_2 \dots Q_nx_n C[(x_1, a)]$ is true. In this paper we restrict our attention to binary QCSPs². In a binary QCSP, each constraint, denoted by c_{ij} , involves two variables (x_i and x_j) which may be universally or existentially quantified.

As an example consider the following QCSP where Q is a sequence of 7 quantified variables, and C is a conjunction of 9 constraints. This problem will be used in Section 3.4 to demonstrate the various features of QCSP-Solve.

Example 1

$$\exists x_1 \exists x_2 \forall x_3 \forall x_4 \forall x_5 \exists x_6 \exists x_7 (x_1 \neq x_6 \wedge x_1 \neq x_7 \wedge x_2 \neq x_6 \wedge x_3 \neq x_6 \wedge x_3 < x_7 \wedge x_4 \neq x_6 \wedge x_4 \neq x_7 \wedge x_5 \neq x_6 \wedge x_5 < x_7)$$

A special case of a QCSP is Quantified Boolean Formula (QBF). A QBF is of the form QC where Q is defined as above (however in this case the domain of each variable is $\{0, 1\}$). C is a Boolean formula in conjunctive normal form (CNF), a conjunction of clauses where each clause is a disjunction of literals. Each literal is a variable and a sign. The literal is said to be negative if negated and positive otherwise. The semantic definition is the same as for QCSPs. Note that binary QCSPs, unlike 2-QBF (i.e. QBF problems with at most two literals per clause), are not trivial. Despite the restriction to binary constraints, binary QCSPs are still PSPACE-complete [Boerner *et al.*, 2003].

In the rest of the paper we assume that for any constraint c_{ij} , variable x_i is before x_j in the quantification sequence, unless explicitly specified otherwise. We will sometimes refer to universally and existentially quantified variables as *universals* and *existentials* respectively.

3 Description of QCSP-Solve

In this section we describe the basic features of QCSP-Solve. First we discuss preprocessing. Then we analyze the look-ahead and look-back capabilities of QCSP-Solve. In standard

¹ (x_1, a) denotes the assignment of value a to variable x_1 .

²QCSP-Solve can currently handle ternary and binary constraints. We are in the process of extending it to constraints of higher arity.

CSPs look-ahead techniques try to detect dead-ends early by pruning values from future variables, while look-back techniques try to deal with dead-ends in an intelligent way by recording and exploiting the reasons for failures. Note that some of the techniques we will describe may delete values from the domains of universal variables because they may discover that, under the current assignments, these values will definitely lead to a solution. This pruning is different than standard pruning in CSPs.

3.1 Preprocessing

Arc consistency (AC) has been extended to QCSPs in [Bordeaux and Monfroy, 2002] and [Mamoulis and Stergiou, 2004]. QCSP-Solve always applies AC as a preprocessing step. Apart from reducing the problem size by deleting values from the domains of existentials, AC removes from the problem all constraints of the form $\exists x_i \forall x_j, c_{ij}$ and $\forall x_i \forall x_j, c_{ij}$. For the former kind, AC deletes every value of $D(x_i)$ that is not supported by all values of $D(x_j)$. If $D(x_i)$ becomes empty then the algorithm determines insolubility. For the latter kind, if there is a value of $D(x_i)$ that is not supported by all values of $D(x_j)$ then the algorithm determines that the problem is insoluble. After AC has been applied, all such constraints can be safely removed from the problem since they cannot have any further effect. A consequence of this removal is that any universals after the last existential can be ignored, since they participate in no constraints.

3.2 Look-Ahead

In QCSP-Solve we have implemented two basic forms of look-ahead; forward checking (FC) and maintaining arc consistency (MAC). FC (called FC0 in [Mamoulis and Stergiou, 2004] and hereafter) is an extension of standard FC to QCSPs. FC0 is a backtracking-based algorithm that can discover dead-ends early by forward checking the current variable assignment (of an existential or universal) against values of future existentials constrained with the current variable. By slightly modifying the forward checking phase of FC we get an algorithm, called FC1 in [Mamoulis and Stergiou, 2004], which can discover dead-ends earlier than FC0. FC1 has exactly the same behavior as FC0 when the current variable is an existential. If the current variable x_i is a universal then FC1 forward checks each value of x_i against all future variables before assigning a specific value to it. If one of x_i 's values causes a domain wipe-out then FC1 backtracks to the last existential. Otherwise, it proceeds by instantiating the next available value a of $D(x_i)$ and removing all values of future variables that are inconsistent with the assignment (x_i, a) . In this way FC1 can discover dead-ends earlier and avoid fruitless exploration of search tree branches.

The MAC algorithm is also an extension of standard MAC to QCSPs. After each variable assignment, MAC applies AC in the problem using the AC algorithm of [Mamoulis and Stergiou, 2004]. MAC has also been modified in the same way as FC to yield MAC1, an algorithm analogous to FC1. That is, when the current variable x_i is a universal MAC1 applies AC for each instantiation (x_i, a_j) , $j \in \{1, \dots, d\}$ before committing to a particular instantiation. If one of the instantiations causes a domain wipe-out then the algorithm

backtracks. Otherwise, it commits to one of the values and proceeds with the next variable.

In the rest of the paper we will describe how various look-ahead and look-back techniques are combined with an FC-based look-ahead. Most of these techniques can be combined with a MAC-based look-ahead in a very similar way.

The Pure Value Rule

In SAT and QBF a literal l is called *pure* (or *monotone*) if its complementary literal does not appear in any clause. Such literals are important because they can immediately be assigned a value without any need for branching [Cadoli *et al.*, 2002]. This is what the *pure literal rule* does. For example, if an existential literal l only occurs positively, the pure literal rule will set it to true. By repeating and analyzing the experiments of [Gent *et al.*, 2004] we discovered that the pure literal rule has a profound impact on the search effort. When switching it off, the search process was slowed down by orders of magnitude. This immediately gave rise to the following questions: What does the pure literal rule correspond to in QCSPs, and how can we exploit it to prune the search space? To answer these questions, we use the notion of a *pure value*.

Definition 2 A value $a \in D(x_i)$ of a QCSP QC is *pure* iff $\forall Q_j x_j \in Q$, where $x_j \neq x_i$ and $\forall b \in D(x_j)$, the assignments (x_i, a) and (x_j, b) are compatible.

In a way analogous to the pure literal rule in QBF, we have devised and implemented a look-ahead technique, which we call the *pure value (PV) rule*, that detects and exploits pure values to prune the search space. The actions taken are dual for existential and universal pure values. An existential variable with a pure value can be set to that value, while a pure value is removed from the domain of a universal variable. This duality reflects the dual semantics of existential and universal variables. Note that values can become pure dynamically during search because of constraint propagation (see Example 2 in Section 3.4). Therefore, the PV rule is applied both as a preprocessing technique and as a dynamic look-ahead technique during search. The PV rule works as follows.

- If a pure value a of an existential x_i is discovered during preprocessing (*search*), then the assignment (x_i, a) is made and all other values of x_i are permanently (*temporarily*) removed from $D(x_i)$. To check, during search, if a value a of an existential x_i is pure, we only need to check if the assignment (x_i, a) is compatible with all values of future variables. FC (or MAC) guarantee that (x_i, a) is compatible with the values (i.e. the instantiations) of the previous variables.
- If a pure value a of a universal x_i is discovered during preprocessing (*search*), then a is permanently (*temporarily*) removed from $D(x_i)$. To check if a value of a universal is pure, we only need to check against future variables since preprocessing with AC guarantees that there are no constraints between a universal and a previous variable. Note that if all the values of a universal are pure then we can ignore this variable.

Symmetry Breaking

QCSP-Solve utilizes a technique for symmetry breaking based on neighborhood interchangeability. A value a of a

variable x_i is *fully interchangeable* with a value b of x_i , iff every solution which contains the assignment (x_i, a) remains a solution if we substitute b for a , and vice versa [Freuder, 1991]. A value $a \in D(x_i)$ is *neighborhood interchangeable* (NI) with a value $b \in D(x_i)$, iff for each j , such that $c_{ij} \in C$, a and b are compatible with exactly the same values of $D(x_j)$. QCSP-Solve exploits NI to break some symmetries by pruning the domains of universal variables. That is, for each set of NI values we keep one representative and remove the others, either permanently before search, or temporarily during search³. If the algorithm proves that the representative is consistent (i.e. satisfies the QCSP) then so are the rest.

3.3 Look-Back

Various look-back schemes have been developed for CSPs. One of the most successful is conflict-based backjumping [Prosser, 1993]. This algorithm has been successfully combined with FC in CSPs [Prosser, 1993], and a DLL-based procedure in QBF [Giunchiglia *et al.*, 2001]. We describe how CBJ interacts with the FC-based look-ahead of QCSP-Solve.

As in standard CSPs, for each variable x_i we keep a *conflict set*, denoted by $conf_set(x_i)$, which holds the past variables that are responsible for the deletion of values from $D(x_i)$. Initially all conflict sets are empty. When encountering a dead-end, CBJ exploits information kept in conflict sets to backjump to a variable that is (partly) responsible for the dead-end. Conflict sets are updated as follows. If the current variable x_i is existentially quantified and, during forward checking, a value of a future variable x_j is found to be incompatible with the assignment of x_i then x_i is added to $conf_set(x_j)$. If the domain of a future variable x_j is wiped out then the variables in $conf_set(x_j)$ are added to conflict set of the current variable (existential or universal). Backjumping can occur in either of the following two cases:

- If the current variable x_i is existential and there are no more values to be tried for it then QCSP-Solve backjumps to the rightmost variable x_k in Q that belongs to $conf_set(x_i)$. At the same time all variables in $conf_set(x_i)$ (except x_k) are copied to $conf_set(x_k)$ so that no information about conflicts is lost.
- If the current variable x_i is universal and a value is deleted from its domain (because its forward checking results in a domain wipeout) then QCSP-Solve backjumps to the rightmost variable x_k in Q that belongs to $conf_set(x_i)$. Again all variables in $conf_set(x_i)$ (except x_k) are copied to $conf_set(x_k)$.

Solution-Directed Pruning

[Giunchiglia *et al.*, 2001] introduced solution-directed backjumping for QBF. This allows backjumps over universally quantified literals once reaching a leaf node that is a solution. Inspired by this idea, we have implemented a technique that can prune values from universal variables when reaching a solution (i.e. a consistent leaf node). We call this *solution directed pruning* (SDP). SDP is based on the following idea: Assume that x_i is the last universal in Q and

³Experiments have shown that NI during search is an overhead when the PV rule is used.

$q = \{x_{i+1} \dots x_n\} \subset Q$ is the sequence of existentials to the right of x_i . Also, assume that the assignment (x_i, a_i) leads to a solution (i.e. is part of a path to a consistent leaf node) and $\{(x_{i+1}, a_{i+1}) \dots (x_n, a_n)\}$ are the assignments of variables $\{x_{i+1} \dots x_n\}$ along this path. Then any value of x_i that is compatible with all these assignments will definitely also lead to a solution. Such values can be pruned (i.e. ignored) by the search algorithm. Based on this, SDP first computes the values of the last universal x_i that have the above property. All such values are temporarily removed from $D(x_i)$. Now if there are no available values in $D(x_i)$, SDP proceeds with the universal immediately before x_i in Q , say x_j , and checks if its remaining values are compatible with the assignments of all existentials after x_j . This is repeated recursively until a universal is found which has available values left in its domain after SDP has been applied. The algorithm then backtracks to this universal. In this way it is possible to perform solution-directed backjumps.

3.4 The Algorithm of QCSP-Solve

A high level description of QCSP-Solve's algorithm is shown in Figure 1. It takes a QCSP QC and returns TRUE if the problem is satisfiable, and FALSE otherwise⁴. The version of QCSP-Solve shown in Figure 1 is based on FC. A MAC-based version with all the features, except CBJ for the time being, is also currently available. In Figure 1,

- c_var is the current variable.
- $preprocess()$ is a function that preprocesses the problem by applying AC, and computing pure and NI values.
- $compute_PV()$ computes the pure values of c_var during search. If c_var is existential and one of its values (say a) is pure then $compute_PV(c_var)$ sets c_var to a and temporarily removes the rest of $D(c_var)$'s values. If c_var is universal then $compute_PV(c_var)$ temporarily removes all the pure values from $D(c_var)$. Whenever the algorithm backtracks, all values removed by $compute_PV()$ are restored.
- $fc0()$ implements the FC0-type look-ahead. It is called after the current variable (existential or universal) is assigned and forward checks this assignment against all future variables constrained with c_var . If a value of a future variable x_i is deleted then c_var is added to $conf_set(x_i)$. If $D(x_i)$ is wiped out then $\forall x_j, x_j \in conf_set(x_i)$, x_j is added to $conf_set(c_var)$.
- $fc1()$ implements the FC1-type look-ahead. It is called before c_var is assigned (if it is a universal) and forward checks all of $D(c_var)$'s valid values against the future variables constrained with c_var . If the domain of a future variable x_i is wiped out then $\forall x_j, x_j \in conf_set(x_i)$, x_j is added to $conf_set(c_var)$.
- $SDP()$ implements solution directed pruning. $SDP()$ prunes values from universals according to the rule described in Section 3.3 and returns the universal that has values left in its domain after SDP has been applied.

⁴The algorithm can be modified to return solutions in the form of search trees.

```

Boolean QCSP-Solve ( $Q, C$ )
1:  $preprocess(Q, C)$ 
2:  $c\_var \leftarrow$  leftmost variable in the quantification formula
3: while there is no backtrack from the first existential or universal
4:    $compute\_PV(c\_var)$ 
5:   if  $c\_var$  is existential
6:     if no more values in  $D(c\_var)$ 
7:        $c\_var \leftarrow$  rightmost variable in  $conf\_set(c\_var)$ 
8:     else
9:       assign  $c\_var$  with next valid value  $a \in D(c\_var)$ 
10:       $DWO \leftarrow fc(c\_var, a)$ 
11:      if  $DWO = \text{FALSE}$ 
12:        if there are no more unassigned variables
13:          if there are no universals in  $Q$  return TRUE
14:        else
15:           $c\_var \leftarrow SDP(Q)$ 
16:          restore values removed by all variables after  $c\_var$ 
17:        else  $c\_var \leftarrow$  next unassigned variable
18:      else restore values removed by  $c\_var$ 
19:    else //  $c\_var$  is universal //
20:      if no more values in  $D(c\_var)$ 
21:        if  $c\_var$  is the first universal return TRUE
22:      else  $c\_var \leftarrow$  last assigned universal variable
23:      else
24:         $DWO = \text{FALSE}$ 
25:        if no assignment to  $c\_var$  has been tried
26:           $DWO \leftarrow fc1(c\_var)$ 
27:        if  $DWO = \text{FALSE}$ 
28:          assign  $c\_var$  with next available value  $a \in D(c\_var)$ 
29:           $fc(c\_var, a)$ 
30:           $c\_var \leftarrow$  next unassigned variable
31:        else
32:           $c\_var \leftarrow$  rightmost variable in  $conf\_set(c\_var)$ 
33:          restore removed values by all variables after  $c\_var$ 
34:        if there is a backtrack from the first existential return FALSE
35:      return TRUE

```

Figure 1: The algorithm of QCSP-Solve.

QCSP-Solve works as follows. It takes as input a QCSP QC and, after preprocessing the problem (line 1), it proceeds by checking assignments of values to variables until the truth of the QCSP is proved or disproved. Before assigning a value to c_var , QCSP-Solve calls $compute_PV(c_var)$ to compute the pure values of c_var (line 4). If c_var is existential and a dead-end occurs then the algorithm backtracks to the rightmost variable in $conf_set(c_var)$ (lines 6–7). Otherwise, the next valid value of c_var is forward checked against future variables (lines 9–10). If there is no domain wipe-out (DWO) and the algorithm has reached a consistent leaf node (i.e. c_var is the last variable in Q) then it calls $SDP()$ to perform solution-directed pruning (line 13). If QCSP-Solve is not at a leaf node, it proceeds by moving to the next variable (line 15). If there is a DWO, the next value of c_var will be tried in the next iteration of the **while** loop.

If c_var is universal and all of its values have been proved to be consistent (according to the current assignments), then there are two cases. If c_var is the first universal, QCSP-Solve terminates successfully (line 19). Otherwise, it backtracks to the last universal (line 20). Before assigning any value to a universal variable, QCSP-Solve calls $fc1(c_var)$ to perform FC1-type look-ahead (lines 23–24). If there is

a DWO, the algorithm backtracks to the rightmost variable in $conf_set(c_var)$ (line 30). If there is no DWO, or $fc1(c_var)$ has already been called at this level, c_var is assigned with its next available value (line 26), the assignment is forward checked against future variables (line 27), and QCSP-Solve proceeds with the next variable (line 28).

To better understand how the algorithm of QCSP-Solve works, consider the following example.

Example 2 Assume that the domains of the variables in the problem of Example 1 are as follows: $D(x_1) = \{2, 3\}$, $D(x_2) = \{0, 1, 2\}$, $D(x_3) = \{0, 3\}$, $D(x_4) = \{0, 1, 6\}$, $D(x_5) = \{4, 5\}$, $D(x_6) = \{0, 1, 2, 3\}$, $D(x_7) = \{0, 2, 3, 6\}$. Let us trace the execution of QCSP-Solve for a few steps.

1) Preprocessing is applied. There are no arc inconsistent or pure values, so no pruning is performed.⁵ 2) The assignment $(x_1, 2)$ is made. FC reduces $D(x_6)$ and $D(x_7)$ to $\{0, 1, 3\}$ and $\{0, 3, 6\}$ respectively. We now have $conf_set(x_6) = conf_set(x_7) = \{x_1\}$. 3) Now, value 2 of x_2 becomes pure because it is supported by all values in future variables. The PV rule will immediately make the assignment $(x_2, 2)$. 4) FC1 does not wipe out any future domain, so the assignment $(x_3, 0)$ will be made. FC reduces $D(x_6)$ and $D(x_7)$ to $\{1, 3\}$ and $\{3, 6\}$ respectively. 5) Value 0 of x_4 is pure. Therefore, it is removed and the assignment $(x_4, 1)$ is made. FC reduces $D(x_6)$ to $\{3\}$. 6) FC1 does not wipe out any future domain, so the assignment $(x_5, 4)$ will be made. FC reduces $D(x_7)$ to $\{6\}$. 7) x_6 and x_7 are assigned their only available values and a solution is found. 8) Now function SDP() is called (line 13). SDP() discovers that value 5 of the last universal (x_5) is compatible with the assignments of all the existentials after x_5 . Therefore, this value is removed from $D(x_5)$ and a solution-directed backjump to x_4 is performed. 9) The assignment $(x_4, 6)$ is made. FC reduces $D(x_6)$ and $D(x_7)$ to $\{1, 3\}$ and $\{3\}$ respectively. 10) FC1 applied at x_5 wipes out $D(x_7)$ (value 4 of x_5 is incompatible with the only value in $D(x_7)$). Therefore, we have a dead-end. $conf_set(x_7)$ will be added to $conf_set(x_5)$ and the algorithm will backjump to the rightmost variable in $conf_set(x_5)$, which is x_1 .

Figure 2 shows part of the search tree generated by QCSP-Solve and illustrates how subtrees are pruned.

4 Experiments

[Gent *et al.*, 2004] showed that the model for random generation of QCSPs used in [Mamoulis and Stergiou, 2004] can suffer from a local flaw that makes almost all of the generated instances false. In this model there are k alternating quantifiers applied to disjoint sets of variables, with the innermost quantifier being existential. Let us briefly describe the flaw. Suppose we have a series of k universals $x_1, \dots, x_k$ assigned to values $a_1, \dots, a_k$ respectively. If there is an existential x_i later in Q than the k universals and each one of its values is in conflict with one of the values assigned to the universals then the assignment of values $a_1, \dots, a_k$ to variables $x_1, \dots, x_k$ is

⁵Values 4 and 5 of x_5 are NI, but let us ignore this for the sake of the example.

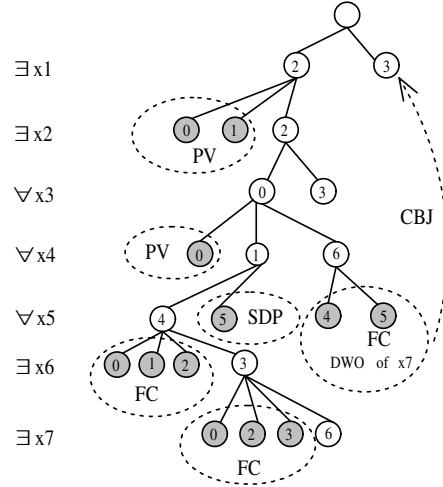


Figure 2: Search tree of Example 2. Dark nodes are pruned by QCSP-Solve. Such nodes together with the feature responsible for their pruning are included in dashed ovals.

inconsistent. This assignment will remain inconsistent irrespective of the assignments to other universals or existentials, and therefore the problem is unsatisfiable. We now propose a generator that can be used to control the probability of flaws.

Variables are quantified in three blocks, a block of existentials followed by a block of universals then another block of existentials. The generator takes 7 parameters: $\langle n, n_\forall, n_{pos}, d, p, q_{\forall\exists}, q_{\exists\exists} \rangle$ where n is the total number of variables, $n_\forall$ is the number of universals, n_{pos} is the position of the first universal in Q , d is the uniform domain size, and p is the number of binary constraints as a fraction of all possible constraints. $q_{\exists\exists}$ is the number of goods in $\exists x_i \exists x_j, c_{ij}$ constraints as a fraction of all possible tuples, and $q_{\forall\exists}$ is a similar quantity for $\forall x_i \exists x_j, c_{ij}$ constraints, described further below. The other two types of constraints that can be removed by preprocessing are not generated.

Since the flaw is a characteristic of $\forall x_i \exists x_j, c_{ij}$ constraints, we restrict these in the following way: we generate a random total bijection from one domain to the other. All tuples not in the bijection are goods. Now $q_{\forall\exists}$ is the fraction of goods from the d tuples in the bijection.

To control the probability p_f of the flaw, we write down an expression for p_f , approximating proportions $p, q_{\forall\exists}, q_{\exists\exists}$ as probabilities. $n_\forall$ is the number of universal variables, and $n_\exists$ is the number of inner existential variables. For each existential assignment (x_i, a) , the probability that it is covered by a universal is $p(1 - q_{\forall\exists})$. If the variable x_i is flawed, then all its values are in conflict with some value of some universal variable. However, each universal variable can only cover one value (since we use a bijection). Therefore (representing existential values using integers) the probability that variable x_i is flawed is given by the following.

$$p(x_i \text{ flawed}) = p(1)p(2|1)p(3|1 \wedge 2) \dots \quad (1)$$

The probability that value a is flawed, given that the previous $a - 1$ values are flawed, is given by formula 2.

$$p(a|1 \dots a-1) = 1 - (1 - p_1(1 - q_{\forall\exists}))^{n_{\forall}-a-1} \quad (2)$$

Substituting equation (2) into equation (1) gives the probability of one variable being flawed.

$$p(x_i \text{ flawed}) = \prod_{i=0}^{d-1} (1 - (1 - p_1(1 - q_{\forall\exists}))^{n_{\forall}-i}) \quad (3)$$

The probability that no existential variables are flawed is given below. This formula is undefined when $d > n_{\forall}$. In this case, $p_f = 1$.

$$p_f = (1 - p(x_i \text{ flawed}))^{n_{\exists}} \quad (4)$$

Experimental Results

Figure 3 presents a comparison of algorithms FC1, FC1+PV, MAC1+PV, and full QCSP-Solve on problems generated according to the model described above. All algorithms apply AC, and NI preprocessing. For each value of $q_{\exists\exists}$ shown in the figures, 100 problem instances were generated and we use the mean average. We include FC1+PV and MAC1+PV in the comparison to illustrate the power of the PV rule. In the problems of Figure 3 the execution of FC1 was stopped at the cut-off limit of 2 hours in more than 50% of the instances. As we can see, QCSP-Solve is many orders of magnitude faster than FC1. The speed-up obtained is largely due to the application of the PV rule. Similar results were obtained with various parameter settings.

Having established that QCSP-Solve is considerably faster than existing direct approaches, we compared it with the state-of-the-art approach of [Gent *et al.*, 2004], using the adapted log encoding with the CSBJ QBF solver. Figure 4 presents indicative results of this comparison. In this case, we used the median because of high outliers. As we can see, QCSP-Solve is significantly faster than CSBJ (more than one order of magnitude), except for very high values of $q_{\exists\exists}$.

5 Conclusion

We introduced QCSP-Solve, an efficient solver for QCSPs. QCSP-Solve incorporates a variety of techniques that are either extensions of techniques used in CSPs and QBF, or are specifically designed for QCSPs. To our knowledge, this is the first time these techniques have been devised and implemented in QCSPs. We also proposed a random generation model that can be used to create instances that are free from the flaw discovered in [Gent *et al.*, 2004]. Experiments showed that QCSP-Solve is several orders of magnitude faster than the existing state-of-the-art direct algorithms for QCSPs, and also significantly outperforms approaches based on encoding QCSPs into QBFs. Current and future work includes extending the solver to handle constraints of any arity, and incorporating other advanced techniques, such as learning.

References

[Benhamou and Goualard, 2000] F. Benhamou and F. Goualard. Universally Quantified Interval Constraints. In *Proceedings of CP-2000*, pages 67–82, 2000.

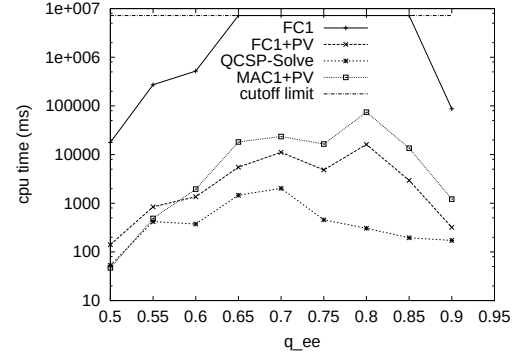


Figure 3: $n = 21$, $n_{\forall} = 7$, $d = 8$, $p = 0.20$, $q_{\forall\exists} = 1/2$.

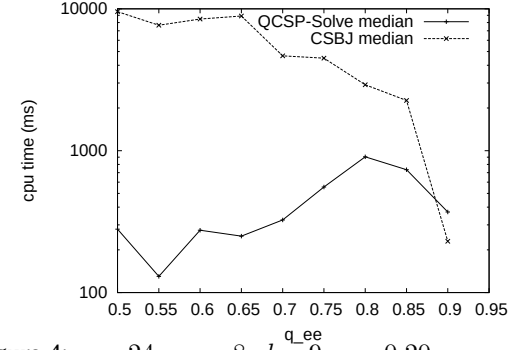


Figure 4: $n = 24$, $n_{\forall} = 8$, $d = 9$, $p = 0.20$, $q_{\forall\exists} = 1/2$.

- [Boerner *et al.*, 2003] F. Boerner, A. Bulatov, P. Jeavons, and A. Krokhin. Quantified Constraints: Algorithms and Complexity. In *Proceedings of CSL-2003*, pages 244–258, 2003.
- [Bordeaux and Monfroy, 2002] L. Bordeaux and E. Monfroy. Beyond NP: Arc-consistency for Quantified Constraints. In *Proceedings of CP-2002*, 2002.
- [Cadoli *et al.*, 2002] M. Cadoli, M. Schaerf, A. Giovanardi, and M. Giovanardi. An Algorithm to Evaluate Quantified Boolean Formulae and its Experimental Evaluation. *Journal of Automated Reasoning*, 28(2):101–142, 2002.
- [Freuder, 1991] E. Freuder. Eliminating Interchangeable Values in Constraint Satisfaction Problems. In *Proceedings of AAAI-91*, pages 227–233, 1991.
- [Gent *et al.*, 2004] I. Gent, P. Nightingale, and A. Rowley. Encoding Quantified CSPs as Quantified Boolean Formulae. In *Proceedings of ECAI-2004*, pages 176–180, 2004.
- [Giunchiglia *et al.*, 2001] E. Giunchiglia, M. Narizzano, and A. Tacchella. Backjumping for Quantified Boolean Logic Satisfiability. In *Proceedings of IJCAI-2001*, pages 275–281, 2001.
- [Mamoulis and Stergiou, 2004] N. Mamoulis and K. Stergiou. Algorithms for Quantified Constraint Satisfaction Problems. In *Proceedings of CP-2004*, pages 752–756, 2004.
- [Prosser, 1993] P. Prosser. Hybrid Algorithms for the Constraint Satisfaction Problem. *Computational Intelligence*, 9(3):268–299, 1993.
- [Ratschan, 2002] S. Ratschan. Quantified Constraints under Perturbations. *Journal of Symbolic Computation*, 33(4):493–505, 2002.

The computational complexity of dominance and consistency in CP-nets

Judy Goldsmith

Dept. of Comp. Sci.
University of Kentucky
Lexington, KY
40506-0046, USA
goldsmith@cs.uky.edu

Jérôme Lang

IRIT
Université Paul Sabatier
31062 Toulouse Cedex
France
lang@irit.fr

Mirosław Truszczyński

Dept. of Comp. Sci.
University of Kentucky
Lexington, KY
40506-0046, USA
mirek@cs.uky.edu

Nic Wilson

Cork Constraint
Computation Centre
University College Cork
Ireland
n.wilson@4c.ucc.ie

Abstract

We investigate the computational complexity of testing dominance and consistency in CP-nets. Up until now, the complexity of dominance has been determined only for restricted classes in which the dependency graph of the CP-net is acyclic. However, there are preferences of interest that define cyclic dependency graphs; these are modeled with general CP-nets. We show here that both dominance and consistency testing for general CP-nets are PSPACE-complete. The reductions used in the proofs are from STRIPS planning, and thus establish strong connections between both areas.

1 Introduction

The problems of eliciting, representing and reasoning with preferences over a multivariable (or, multiattribute) domain arise in many fields such as planning, design, and group decision making. An explicit representation of a preference ordering of elements (we refer to them as *outcomes*) of such multivariable domains is exponentially large in the number of attributes. Therefore, AI researchers have developed languages for representing preference orderings succinctly. The formalism of CP-nets [Boutilier *et al.*, 1999] is among the most popular ones. A CP-net provides a succinct representation of preference ordering on outcomes in terms of local preference statements of the form $p : x_i > x_j$, where x_i, x_j are values of a variable X and p is a logical condition. Informally, a preference statement $p : x_i > x_j$ means that given p , x_i is (strictly) preferred to x_j *ceteris paribus*, that is, *all other things being equal*. The meaning of a CP-net is given by a certain ordering relation (*dominance*) on the set of outcomes, derived from such reading of preference statements. If one outcome dominates another, we say that the dominant one is preferred.

Reasoning about the preference ordering (dominance relation) expressed by a CP-net is far from easy. The key problems include *dominance testing* and *consistency testing*. In the first problem, given a CP-net and two outcomes α and β , we want to decide whether β dominates α . The second problem asks whether there is a dominance cycle in the dominance ordering defined by an input CP-net, that is, whether there is an outcome that dominates (is preferred to) itself.

We study the computational complexity of these two problems. The results obtained so far concerned only restricted classes of CP-nets, all requiring that the graph of variable dependencies implied by preference statements in the CP-net be *acyclic*. Under certain assumptions, the dominance-testing problem is in NP and, under some additional assumptions, even in P [Domshlak and Brafman, 2002; Boutilier *et al.*, 2004a]. Its complexity in the general case has remained until now an open problem. We show that it is in fact PSPACE-complete, even for the propositional case, by exhibiting in Section 4 a PSPACE-hardness proof for dominance testing.

We then turn to consistency testing. While acyclic CP-nets are guaranteed to be consistent, this is not the case with general CP-nets (see [Domshlak and Brafman, 2002; Brafman and Dimopoulos, 2004] for detailed examples and discussions). In Section 5, we show that consistency testing is as hard as dominance testing.

To prove the hardness part of the results, we first establish the PSPACE-hardness of some problems related to propositional STRIPS planning. We then show that these problems can be reduced to CP-net dominance and consistency testing by exploiting connections between actions in STRIPS planning and preference statements in CP-nets.

We assume some familiarity with the complexity class PSPACE (we refer to [Papadimitriou, 1994] for details). In particular, we rely later on the equivalences $\text{NPSPACE} = \text{PSPACE} = \text{coPSPACE}$.

The complexity results in this paper address cyclic CP-nets. Most earlier work has concentrated on the acyclic model. However, we argue that acyclic CP-nets are not sufficiently expressive to capture human preferences on even some simple domains. Consider, for instance, a diner who has to choose either red or white wine, and either fish or meat. Given red wine, they prefer meat, and conversely, given meat they prefer red wine. On the other hand, given white wine, they prefer fish, and conversely, given fish they prefer white wine. This gives a consistent cyclic CP-net, and there is no acyclic CP-net giving rise to the same preferences on outcomes. So, such cyclicity of preference variables does not necessarily lead to a cyclic order on outcomes.

2 Generalized propositional CP-nets

Let $V = \{x_1, \dots, x_n\}$ be a finite set of *variables*. For each variable $x \in V$, we assume a finite *domain* D_x of *values*. An

outcome is an n -tuple $(d_1, \dots, d_n)$ of $D_{x_1} \times \dots \times D_{x_n}$.

In this paper, we focus on *propositional* variables: variables with *binary* domains. Let V be a finite set of propositional variables. For every $x \in V$, we set $D_x = \{x, \neg x\}$ (thus, we overload the notation and write x both for the variable and for one of its values). We refer to x and $\neg x$ as literals. Given a literal l we write $\neg l$ to denote the dual literal to l . The focus on binary variables makes the presentation clearer and has no impact on our complexity results.

A *conditional preference rule* (or, for short, a [preference] rule) over V is an expression $p : l > \neg l$, where l is a literal of some atom $x \in V$ and p is a propositional formula over V that does not involve variable x .

Definition 1 (Generalized CP-net) A generalized CP-net C (for short, a GCP-net) over V is a set of conditional preference rules. For $x \in V$ we define $p_C^+(x)$ and $p_C^-(x)$, usually written just: $p^+(x)$ and $p^-(x)$, as follows: $p_C^+(x)$ is equal to the disjunction of all p such that there exists a rule $p : x > \neg x$ in C ; $p_C^-(x)$ is the disjunction of all p such that there exists a rule $p : \neg x > x$ in C . We define the associated directed graph G_C (the dependency graph) over V to consist of all pairs (y, x) of variables such that y appears in either $p^+(x)$ or $p^-(x)$.

In our complexity results we will also need the following representation of GCP-nets: a GCP-net C is said to be in *conjunctive form* if C only contains rules $p : l > \neg l$ such that p is a (possibly empty) conjunction of literals. In this case all formulas $p^-(x)$, $p^+(x)$ are in disjunctive normal form, that is, a disjunction of conjunctions of literals (including $\top$ – the empty conjunction of literals).

GCP-nets determine a transitive relation on outcomes, interpreted in terms of preference. A preference rule $p : l > \neg l$ represents the statement “given that p holds, l is preferred to $\neg l$ *ceteris paribus*”. Its intended meaning is as follows. If outcome β satisfies p and l , then β is preferred to the outcome α which differs from β only in that it assigns $\neg l$ for variable x . In this situation we say that there is an *improving flip* from α to β sanctioned by the rule $p : l > \neg l$. If $\alpha_0, \dots, \alpha_m$ is a sequence of outcomes with $m \geq 1$ and each next outcome in the sequence is obtained from the previous one by an improving flip, then we say that $\alpha_0, \dots, \alpha_m$ is an *improving sequence* (from α_0 to α_m) for the GCP-net, and that α_m *dominates* α_0 , written $\alpha_0 \prec \alpha_m$.

Finally, A GCP-net is *consistent* if there is no outcome α which is strictly preferred to itself, i.e., such that $\alpha \prec \alpha$.

The main objective of the paper is to establish the complexity of the following two problems concerning the notion of dominance associated with GCP-nets (sometimes under restrictions on the class of input GCP-nets).

Definition 2

GCP-DOMINANCE: given a GCP-net C and two outcomes α and β , decide whether $\alpha \prec \beta$ in C .

GCP-CONSISTENCY: given a GCP-net C , decide whether C is consistent.

There are two properties of GCP-nets that are essential in linking them to the original notion of CP-nets [Boutilier et al., 1999; 2004a].

Definition 3

A GCP-net C over V is *locally consistent* if for every $x \in V$, the formula $p_C^-(x) \wedge p_C^+(x)$ is unsatisfiable. It is *locally complete* if for every $x \in V$, the formula $p_C^-(x) \vee p_C^+(x)$ is a tautology.

Definition 4 (Propositional CP-net) A CP-net over the set V of (propositional) variables is a *locally consistent and locally complete GCP-net* over V .

Problems CP-DOMINANCE and CP-CONSISTENCY are defined analogously to Definition 2.

This definition of a CP-net differs from the one given in [Boutilier et al., 2004a], which uses explicit conditional preference tables. Our representation is often more compact, but it is easy to verify that it is equivalent in that it gives rise to the same definition of dominance.

When defining a decision problem, it is critical to specify the way to represent its input instances, as the representation may affect the complexity of the problem. Unless stated otherwise, we assume that GCP-nets (and so, also CP-nets) are represented as a set of preference rules, as described in Definition 1. Therefore, the size of a GCP-net is given by the total size of the formulas $p^-(x)$, $p^+(x)$, $x \in V$.

3 Propositional STRIPS planning

In this section we derive some technical results on propositional STRIPS planning which form the basis of our complexity results in Sections 4 and 5. We establish the complexity of plan existence problems for propositional STRIPS planning, under restrictions on input instances that make the problem useful in the studies of dominance and consistency in GCP-nets.

Let V be a finite set of variables. A *state* over V is a complete and consistent set of literals over V (which we often view as the conjunction of its members). A state is therefore equivalent to an *outcome*, defined in a CP-nets context.

Definition 5 (Propositional STRIPS planning) A propositional STRIPS instance is a tuple $\langle V, \alpha_0, \gamma, ACT \rangle$, where

1. V is a finite set of propositional variables;
2. α_0 is a state (over V), called the initial state;
3. γ is a state called the goal;
4. ACT is a finite set of actions, where each action $a \in ACT$ is described by a consistent conjunction of literals $pre(a)$ (a precondition) and a consistent conjunction of literals $post(a)$ (a postcondition, or effect).

An action a is *executable* in a state α if $\alpha \models pre(a)$. The effect of a in state α is the state α' containing the same literals as α for all variables not mentioned in $post(a)$, and the literals of $post(a)$ otherwise. We assume that an action can be applied to any state, but that it has no effect if its preconditions do not hold (this assumption has no effect on complexity).

The PROPOSITIONAL STRIPS PLAN EXISTENCE problem, or STRIPS PLAN for short, is to decide whether for a given propositional STRIPS instance $\langle V, \alpha_0, \gamma, ACT \rangle$ there is a *successful plan*, that is, sequence of actions leading from the initial state α_0 to a state satisfying the goal γ .

A plan is *irreducible* if its every action changes the state.

We assume, without loss of generality, that for any action a , no literal in $post(a)$ also appears in $pre(a)$; otherwise we can omit the literal from $post(a)$ without changing the effect of the action; (if $post(a)$ then becomes an empty conjunction, the action a can be omitted from ACT as it has no effect).

We have the following result, proved in [Bylander, 1994] :

Proposition 1 ([Bylander, 1994])

STRIPS PLAN is PSPACE-complete.

Typically, propositional STRIPS instances do not require that goals be complete. We restrict consideration to complete goals. This restriction has no effect on the complexity: the plan existence problem remains PSPACE-complete under the goal-completeness restriction [Lang, 2004].

3.1 Acyclic STRIPS

Definition 6 (Acyclic sets of actions) A set of actions ACT is acyclic if there is no state α such that $\langle V, \alpha, \alpha, ACT \rangle$ has a non-empty irreducible plan (informally, if there are no non-trivial directed cycles in the space of states induced by ACT).

We will now consider the following two problems:

1. ACYCLIC STRIPS PLAN: given a propositional STRIPS instance $\langle V, \alpha_0, \gamma, ACT \rangle$ such that ACT is acyclic and $\alpha_0 \neq \gamma$, decide whether there is a plan for $\langle V, \alpha_0, \gamma, ACT \rangle$.
2. ACTION-SET ACYCLICITY: given a set ACT of actions, decide whether ACT is acyclic.

We will show that both problems are PSPACE-complete.

Proposition 2

ACYCLIC STRIPS PLAN is PSPACE-complete.

Proof: Membership in PSPACE is evident as the problem is a restriction of STRIPS PLAN. To prove PSPACE-hardness, we exhibit a polynomial-time reduction from STRIPS PLAN. Let $PE = \langle V, \alpha_0, \gamma, ACT \rangle$ be an instance of STRIPS PLAN. The idea behind the reduction is to introduce a *counter*, so that each time an action is executed, the counter is incremented. The counter may count up to 2^n , where $n = |V|$, making use of n additional variables. The counter is initialized to 0. Once it reaches $2^n - 1$ it can no longer be incremented and no action can be executed. Hence, the set of actions in the resulting instance of STRIPS PLAN is acyclic.

To describe the reduction, we write V as $\{x_1, \dots, x_n\}$. We define $PE' = \langle V', \alpha'_0, \gamma', ACT' \rangle$ as follows:

- $V' = \{x_1, \dots, x_n, z_1, \dots, z_n\}$, where z_i are new variables we will use to implement the counter;
- $\alpha'_0 = \alpha_0 \wedge \neg z_1 \wedge \dots \wedge \neg z_n$;
- $\gamma' = \gamma \wedge z_1 \wedge \dots \wedge z_n$;
- for each action $a \in ACT$, we include in ACT' n actions a^i , $1 \leq i \leq n$, such that
 $pre(a^i) = pre(a) \wedge \neg z_i \wedge z_{i+1} \wedge \dots \wedge z_n$
 $post(a^i) = post(a) \wedge z_i \wedge \neg z_{i+1} \wedge \dots \wedge \neg z_n$
- Furthermore, we include in ACT' n actions b_i , $1 \leq i \leq n$, such that
 $pre(b_i) = \neg z_i \wedge z_{i+1} \wedge \dots \wedge z_n$
 $post(b_i) = z_i \wedge \neg z_{i+1} \wedge \dots \wedge \neg z_n$

We will denote states over V' by pairs (α, k) , where α is a state over V and k is an integer, $0 \leq k \leq 2^n - 1$. We view k as a compact representation of a state over variables $z_1, \dots, z_n$: assuming that the binary representation of k is $d_1 \dots d_n$ (with d_n being the least significant digit), k represents the state which contains z_i if $d_i = 1$ and $\neg z_i$, otherwise.

PE' is acyclic, since executing any action in ACT' increments the counter k and no action can be executed once the counter has reached the value $2^n - 1$.

We claim that there is a plan for PE if and only if there is a plan for PE' . First, assume that there is a plan in PE . Let π be a shortest plan in PE and let m be its length. We have $m \leq 2^n - 1$, since no state along π repeats (otherwise, shorter plans than π for PE would exist). Let $\alpha_0, \alpha_1, \dots, \alpha_m = \gamma$ be the sequence of states obtained by executing π . Let a be the action used in the transition from α_k to α_{k+1} . Since $k < 2^n - 1$, there is exactly one i , $1 \leq i \leq n$, such that the action a^i applies at the state (α_k, k) over V' . Replacing a with a^i in π yields a plan that when started at $(\alpha_0, 0)$ leads to $(\alpha_m, m) = (\gamma, m)$. Appending that plan with appropriate actions b_i to increment the counter to $2^n - 1$ yields a plan for PE' . Conversely, if τ is a plan for PE' , the plan obtained from τ by removing all actions of the form b_j and replacing each action a^i with a is a plan for PE .

Thus, the claim and the assertion follow. ■

Proposition 3

ACTION-SET ACYCLICITY is PSPACE-complete.

Proof: The argument for the membership in PSPACE is standard. To prove PSPACE-hardness, we proceed as follows. Let $PE = \langle V, \alpha_0, \gamma, ACT \rangle$ be a STRIPS instance such that ACT is acyclic and $\alpha_0 \neq \gamma$. Let a be a new action defined by $pre(a) = \gamma$ and $post(a) = \alpha_0$. It is easy to see that $ACT \cup \{a\}$ is not acyclic if and only if there exists a plan for PE . Thus, the PSPACE-hardness of the *complement* of the ACTION-SET ACYCLICITY problem follows from Proposition 2. Since PSPACE = coPSPACE, this suffices to prove the hardness part of the assertion. ■

3.2 Mapping STRIPS plans to single-effect STRIPS plans

Versions of the STRIPS PLAN and ACYCLIC STRIPS PLAN problems that are important for us allow only single-effect actions (actions with exactly one literal in their postconditions) in input propositional STRIPS instances. We refer to these restrictions as SE STRIPS PLAN and ACYCLIC SE STRIPS PLAN.

To prove PSPACE-hardness of both problems, we describe a mapping from STRIPS instances to single-effect STRIPS instances.

Consider an instance $PE = \langle V, \alpha_0, \gamma, ACT \rangle$ of the STRIPS PLAN problem (where ACT is not necessarily acyclic). For each action $a \in ACT$ we introduce a *new* variable x_a . We set $X = \bigwedge_{a \in ACT} \neg x_a$. That is, X is the conjunction of negative literals of all the additional variables. In addition, for each $a \in ACT$ we set $X_a = x_a \wedge \bigwedge_{b \in ACT - \{a\}} \neg x_b$. We now define an instance $PE' = \langle V', \alpha'_0, \gamma', S(ACT) \rangle$ of the SE STRIPS PLAN problem as follows:

- Set of variables: $V' = V \cup \{x_a : a \in ACT\}$;
- Initial state: $\alpha'_0 = \alpha_0 \wedge X$;
- Goal state: $\gamma' = \gamma \wedge X$;
- Set of actions: $S(ACT) = \{a^i : a \in ACT, i = 1, \dots, 2|post(a)| + 1\}$.
Let $a \in ACT$ and $post(a) = l_1 \wedge \dots \wedge l_q$. For $i = 1, \dots, q$, we define:
 $pre(a^i) = pre(a) \wedge X \wedge \neg l_i$; $post(a^i) = x_a$;
 $pre(a^{q+i}) = X_a$; $post(a^{q+i}) = l_i$.
We also define:
 $pre(a^{2q+1}) = X_a \wedge l_1 \wedge \dots \wedge l_q$; $post(a^{2q+1}) = \neg x_a$.

Let π be a sequence of actions in ACT . We define $S(\pi)$ to be the sequence of actions in $S(ACT)$ obtained by replacing each action a in π by $a^1, \dots, a^{2q+1}$ (where $q = |post(a)|$). Now consider a sequence τ of actions from $S(ACT)$. Remove from τ any action a^i such that $i \neq 2|post(a)| + 1$, and replace actions of the form $a^{2|post(a)|+1}$ by a . We denote the resulting sequence of actions from ACT by $S'(\tau)$. The following properties are easy to check (details are omitted):

Lemma 1 *With the above definitions,*

- (i) *if π is a plan for PE then $S(\pi)$ is a plan for PE' ;*
- (ii) *if τ is an irreducible plan for PE' then $S'(\tau)$ is a plan for PE;*
- (iii) *ACT is acyclic if and only if $S(ACT)$ is acyclic.*

Proposition 4

SE STRIPS PLAN and ACYCLIC SE STRIPS PLAN are PSPACE-complete.

Proof: SE STRIPS PLAN and ACYCLIC SE STRIPS PLAN problems are restrictions of STRIPS PLAN, from which membership in PSPACE follows. PSPACE-hardness of ACYCLIC SE STRIPS PLAN (and so, also of the other problem) is shown by reduction from ACYCLIC STRIPS PLAN. Consider an instance $PE = \langle V, \alpha_0, \gamma, ACT \rangle$ of ACYCLIC STRIPS PLAN. Define $PE' = \langle V', \alpha'_0, \gamma', S(ACT) \rangle$, which by Lemma 1(iii) is an instance of the ACYCLIC SE STRIPS PLAN problem. By Lemma 1(i) and (ii) there exists a plan for PE if and only if there exists a plan for PE' . ■

4 Dominance

The goal of this section is to prove that the GCP-DOMINANCE problem is PSPACE-complete, and that the complexity does not go down even when we restrict the class of inputs to CP-nets. We use the results on propositional STRIPS planning from Section 3 to prove that the general GCP-DOMINANCE problem is PSPACE-complete. We then show that the complexity does not change if we impose the requirements of local consistency and local completeness on input GCP-nets.

The similarities between dominance testing in CP-nets and propositional STRIPS planning were first noted in [Boutilier et al., 1999], where a reduction (presented in more detail in [Boutilier et al., 2004a]) was given from the dominance problem to the plan existence problem for a class of propositional STRIPS planning specifications consisting of *unary* actions (actions with single effects). We prove our results for

the GCP-DOMINANCE and GCP-CONSISTENCY problems by constructing a reduction in the other direction.

This reduction is much more complex than the one used in [Boutilier et al., 1999], due to the fact that CP-nets impose more restrictions than STRIPS planning. Firstly, STRIPS planning allows multiple effects, but GCP-nets only allow flips $x > \neg x$ or $\neg x > x$ that change the value of one variable; this is why we constructed the reduction from STRIPS planning to single-effect STRIPS planning in the last section. Secondly, CP-nets impose two more restrictions, local consistency and local completeness, which do not have natural counterparts in the context of STRIPS planning.

For all dominance and consistency problems considered in the paper, the membership in PSPACE can be demonstrated by considering nondeterministic algorithms consisting of repeatedly guessing appropriate improving flips. Such algorithms work in polynomial space and show the membership of problems they solve in NPSPACE and consequently in PSPACE, since NPSPACE = PSPACE. Therefore, due to space restrictions, from now on we only provide arguments for the PSPACE-hardness of problems we consider.

4.1 Dominance for generalized CP-nets

We will prove that the GCP-DOMINANCE problem is PSPACE-complete by a reduction from the problem SE STRIPS PLAN, which we now know to be PSPACE-complete.

Mapping single-effect STRIPS problems to GCP-nets dominance problems

Let $\langle V, \alpha_0, \gamma, ACT \rangle$ be an instance of the SE STRIPS PLAN problem. For every action $a \in ACT$ we denote by l_a the unique literal in the postcondition of a , that is, $post(a) = l_a$. We denote by $pre'(a)$ the conjunction of all literals in $pre(a)$ different from $\neg l_a$ (we recall that by a convention we adopted earlier, $pre'(a)$ does not contain l_a either). We then define c_a to be the conditional preference rule $pre'(a) : l_a > \neg l_a$ and define $M(ACT)$ to be the GCP-net $C = \{c_a : a \in ACT\}$.

A sequence of states in a plan corresponds to an improving sequence from α_0 to γ , which leads to the following result.

Lemma 2 *With the above notation,*

- (i) *there is a non-empty irreducible plan for $\langle V, \alpha_0, \gamma, ACT \rangle$ if and only if γ dominates α_0 in C ;*
- (ii) *ACT is acyclic if and only if $M(ACT)$ is consistent.*

Theorem 1 *The GCP-DOMINANCE problem is PSPACE-complete. Moreover, this remains so under the restrictions that the GCP-net is consistent and is in conjunctive form.*

Proof: PSPACE-hardness is shown by reduction from ACYCLIC SE STRIPS PLAN (Proposition 4). Let $\langle V, \alpha_0, \gamma, ACT \rangle$ be an instance of the ACYCLIC SE STRIPS PLAN problem. By Lemma 2(ii), $M(ACT)$ is a consistent GCP-net in conjunctive form. Since $\alpha_0 \neq \gamma$, there is a plan for $\langle V, \alpha_0, \gamma, ACT \rangle$ if and only if there is a non-empty irreducible plan for $\langle V, \alpha_0, \gamma, ACT \rangle$, which, by Lemma 2(i), is if and only if γ dominates α_0 in C . ■

4.2 Dominance in CP-nets

In this section we show that GCP-DOMINANCE remains PSPACE-complete under the restriction to locally-consistent and locally-complete GCP-nets, i.e., CP-nets. We refer to this restriction of GCP-DOMINANCE as CP-DOMINANCE.

We will show PSPACE-hardness for CP-DOMINANCE by a reduction from GCP-DOMINANCE for consistent GCP-nets.

Mapping locally-consistent GCP-nets to CP-nets

Let C be a locally-consistent GCP-net. Let $V = \{x_1, \dots, x_n\}$ be the set of variables of C . We define $V' = V \cup \{y_1, \dots, y_n\}$, where $\{y_1, \dots, y_n\} \cap V = \emptyset$. We define a GCP-net C' over V' , which we will show is a CP-net. To this end, for every $z \in V'$ we will define conditional preference rules $q^+(z) : z > \neg z$ and $q^-(z) : \neg z > z$ to be included in C' by specifying formulas $q^+(z)$ and $q^-(z)$.

First, for each variable $x_i \in V$, we set

$$q^+(x_i) = y_i \text{ and } q^-(x_i) = \neg y_i.$$

Thus, x_i depends only on y_i . We also note that the formulas $q^+(x_i)$ and $q^-(x_i)$ satisfy local-consistency and local-completeness requirements.

Next, for each variable y_i , $1 \leq i \leq n$, we define

$$e_i = (x_1 \leftrightarrow y_1) \wedge \dots \wedge (x_{i-1} \leftrightarrow y_{i-1}) \\ \wedge (x_{i+1} \leftrightarrow y_{i+1}) \wedge \dots \wedge (x_n \leftrightarrow y_n),$$

$$f_i^+ = e_i \wedge p^+(x_i) \text{ and } f_i^- = e_i \wedge p^-(x_i).$$

Finally, we define

$$q^+(y_i) = f_i^+ \vee (\neg f_i^- \wedge x_i)$$

and

$$q^-(y_i) = f_i^- \vee (\neg f_i^+ \wedge \neg x_i)$$

Thus, y_i depends on every variable in V' but itself.

We note that by the local consistency of C , formulas $f_i^+ \wedge f_i^-$, $1 \leq i \leq n$, are unsatisfiable. Consequently, formulas $q^+(y_i) \wedge q^-(y_i)$, $1 \leq i \leq n$, are unsatisfiable. Thus, C' is locally consistent. Finally, $q^+(y_i) \vee q^-(y_i)$ is equivalent to $f_i^+ \vee \neg x_i \vee f_i^- \vee x_i$, so is a tautology. Thus, C' is locally complete and hence a CP-net over V' .

Let α and β be outcomes over $\{x_1, \dots, x_n\}$ and $\{y_1, \dots, y_n\}$, respectively. By $\alpha\beta$ we denote the outcome over V' obtained by concatenating n -tuples α and β . Conversely, every outcome for C' can be written in this way.

Let α be an outcome over V . We define $\bar{\alpha}$ to be the outcome over $\{y_1, \dots, y_n\}$ obtained by replacing in α every component of the form x_i with y_i and every component $\neg x_i$ with $\neg y_i$. Then for every i , $1 \leq i \leq n$, $\alpha\bar{\alpha} \models e_i$.

Let s be a sequence $\alpha_0, \dots, \alpha_m$ of outcomes over V . Define $L(s)$ to be the sequence of V' -outcomes: $\alpha_0\bar{\alpha}_0, \alpha_0\bar{\alpha}_1, \alpha_1\bar{\alpha}_1, \alpha_1\bar{\alpha}_2, \dots, \alpha_m\bar{\alpha}_m$. Further, let t be a sequence $\epsilon_0, \epsilon_1, \dots, \epsilon_m$ of V' -outcomes with $\epsilon_0 = \alpha\bar{\alpha}$ and $\epsilon_m = \beta\bar{\beta}$. Define $L'(t)$ to be the sequence obtained from t by projecting each element in t to V and iteratively removing elements in the sequence which are the same as their predecessor (until any two consecutive outcomes are different).

Lemma 3 *With the above definitions,*

- (i) *if s is an improving sequence for C from α to β then $L(s)$ is an improving sequence for C' from $\alpha\bar{\alpha}$ to $\beta\bar{\beta}$;*
- (ii) *if t is an improving sequence from $\alpha\bar{\alpha}$ to $\beta\bar{\beta}$ then $L'(t)$ is an improving sequence from α to β ;*
- (iii) *C is consistent if and only if C' is consistent*

Sketch of proof: Let $e = \bigwedge_{i=1}^n (x_i \leftrightarrow y_i)$. The definitions have been arranged to ensure the following for CP-net C' :

- (a) Suppose e holds in an outcome, so the outcome can be written as $\alpha\bar{\alpha}$ for some α ; then no improving flip changes any variable x_i ; furthermore, there is an improving flip changing variable y_i if and only if there is an improving flip for the GCP-net C from outcome α changing variable x_i . After applying this flip changing variable y_i , there is exactly one improving flip possible, changing x_i , after which e holds again (this follows using (b), as y_i cannot be immediately flipped back again, because C' is locally consistent).
- (b) If e does not hold in an outcome then the only improving flips possible change the value of some variable (x_i or y_i) to make $x_i \leftrightarrow y_i$ hold for some i .

(a) implies (i) and (ii). Also, (i) implies half of (iii), that if C is inconsistent then C' is inconsistent. Conversely, suppose that C' is inconsistent, so there exists an improving sequence t for C' from some outcome to itself. By (b), any improving flip applied to an outcome in which e does not hold increases (by one) the number of i such that $x_i \leftrightarrow y_i$ holds. This implies that e must hold in some outcome in t , because t is cyclic. Write this outcome as $\alpha\bar{\alpha}$. We can cyclically permute t to form an improving sequence from $\alpha\bar{\alpha}$ to itself. Part (ii) then implies that there exists an improving flipping sequence for C from α to itself, showing that C is inconsistent. ■

Theorem 2 CP-DOMINANCE is PSPACE-complete. This holds even if we restrict the CP-nets to being consistent.

Proof: We use a reduction from PSPACE-hardness of the GCP-DOMINANCE problem when the GCP-nets are restricted to being consistent (Theorem 1). Let C be a consistent (and hence locally consistent) GCP-net over V , and let α and β be outcomes over V . Consider the CP-net C' over variables V' constructed above. Lemma 3(i) and (ii) imply that β dominates α in C if and only if $\beta\bar{\beta}$ dominates $\alpha\bar{\alpha}$ in C' . Moreover, C' is consistent by Lemma 3(iii). Thus, the hardness part of the assertion follows. ■

5 Consistency of GCP-nets

In this section we show that the GCP-CONSISTENCY problem is PSPACE-complete, using results from Sections 3 and 4.

Theorem 3

GCP-CONSISTENCY is PSPACE-complete. This holds even under the restriction to GCP-nets in conjunctive form.

Proof: PSPACE-hardness is shown by reduction from ACTION-SET ACYCLICITY. We apply function S from Section 3.2 followed by M from Section 4.1. This maps

instances of ACTION-SET ACYCLICITY to instances of GCP-CONSISTENCY in conjunctive form. By Lemma 1(iii) and Lemma 2 (ii), an instance of ACTION-SET ACYCLICITY is acyclic if and only if the corresponding instance of GCP-CONSISTENCY is consistent, proving the result. ■

We now show that consistency testing remains PSPACE-complete for CP-nets.

Theorem 4 CP-CONSISTENCY is PSPACE-complete.

Proof: Let C be a GCP-net in conjunctive form. We define a CP-net C' as follows. If C is locally inconsistent (the property can be decided in polynomial time), we set C' to be any fixed (independent of C) inconsistent but locally consistent CP-net. (Such CP-nets exist.) Otherwise, C is locally-consistent and for C' we take the CP-net we constructed in Section 4.2. The mapping from locally consistent GCP-nets to CP-nets, described in Section 4.2, preserves consistency (Lemma 3 (iii)). Since local inconsistency implies inconsistency, we have that the GCP-net C is consistent if and only if the CP-net C' is consistent. Thus, PSPACE-hardness of the CP-CONSISTENCY problem follows from Theorem 3. ■

Notice the huge complexity gap with the problem of deciding whether there exists a nondominated outcome, which is “only” NP-complete [Domshlak *et al.*, 2003].

6 Concluding remarks

We have shown that dominance and consistency testing in CP-nets is PSPACE-complete. The repeated use of reductions from planning problems confirms the importance of the structural similarity between STRIPS planning and reasoning with CP-nets. This suggests that the well-developed field of planning algorithms for STRIPS representations, especially for unary operators [Brafman and Domshlak, 2003], could be useful for implementing algorithms for dominance and consistency in CP-nets.

Our theorems extend to CP-nets with non-binary domains, and to extensions and variations of CP-nets, such as TCP-nets [Brafman and Domshlak, 2002] that allows for explicit priority of some variables over others, and the more general preference language described in [Wilson, 2004b; 2004a].

The complexity result for dominance is also relevant for constrained optimisation, where a complete algorithm [Boutilier *et al.*, 2004b] involves many dominance checks. Our results reinforce the need for work on finding special classes of problems where dominance and consistency can be tested efficiently [Domshlak and Brafman, 2002; Boutilier *et al.*, 2004a], and for incomplete methods for checking consistency and constrained optimisation [Wilson, 2004a].

Several open problems remain. We do not know whether dominance and consistency testing remain PSPACE-complete when the number of parents in the dependency graph is bounded by a constant. We also do not know whether these two problems remain PSPACE-complete for CP-nets in conjunctive form (the reduction used to prove Theorems 2 and 4 yields CP-nets that are not in conjunctive form). Fi-

nally, we do not know the complexity of deciding whether the preference relation induced by a CP-net is complete.

Acknowledgements

We are grateful for a reviewer’s simple example of a cyclic CP-net. This material is supported in part by the NSF under Grants No. IIS-0325063 and IIS-0097278, by Science Foundation Ireland under Grant No. 00/PI.1/C075, and by the CNRS program RTP 11: “raisonner et décider”.

References

- [Boutilier *et al.*, 1999] C. Boutilier, R. I. Brafman, H. H. Hoos, and D. Poole. Reasoning with conditional ceteris paribus preference statements. In *Proceedings of UAI99*, pages 71–80, 1999.
- [Boutilier *et al.*, 2004a] C. Boutilier, R. Brafman, C. Domshlak, H. Hoos, and D. Poole. CP-nets: a tool for representing and reasoning with conditional ceteris paribus statements. *JAIR*, 21:135–191, 2004.
- [Boutilier *et al.*, 2004b] C. Boutilier, R. I. Brafman, C. Domshlak, H. Hoos, and D. Poole. Preference-based constrained optimization with CP-nets. *Computational Intelligence*, 20(2):137–157, 2004.
- [Brafman and Dimopoulos, 2004] R. Brafman and Y. Dimopoulos. Extended semantics and optimization algorithms for CP-networks. *Computational Intelligence*, 20(2):218–245, 2004.
- [Brafman and Domshlak, 2002] R. Brafman and C. Domshlak. Introducing variable importance trade-offs into CP-nets. In *Proceedings of UAI-02*, pages 69–76, 2002.
- [Brafman and Domshlak, 2003] R. Brafman and C. Domshlak. Structure and complexity of planning with unary operators. *JAIR*, 18:315–439, 2003.
- [Bylander, 1994] T. Bylander. The computational complexity of propositional STRIPS planning. *Artificial Intelligence Journal*, 69(1-2):165–204, 1994.
- [Domshlak and Brafman, 2002] C. Domshlak and R. I. Brafman. CP-nets—reasoning and consistency testing. In *Proceedings of KR2002*, pages 121–132, 2002.
- [Domshlak *et al.*, 2003] C. Domshlak, F. Rossi, K. B. Venable, and T. Walsh. Reasoning about soft constraints and conditional preferences: complexity results and approximation techniques. In *Proceedings of IJCAI-03*, pages 215–220, 2003.
- [Lang, 2004] J. Lang. Logical preference representation and combinatorial vote. *Annals of Mathematics and Artificial Intelligence*, 42(1):37–71, 2004.
- [Papadimitriou, 1994] Ch. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.
- [Wilson, 2004a] N. Wilson. Consistency and constrained optimisation for conditional preferences. In *Proceedings of ECAI-04*, pages 888–892, 2004.
- [Wilson, 2004b] N. Wilson. Extending CP-nets with stronger conditional preference statements. In *Proceedings of AAAI-04*, pages 735–741, 2004.

The Complexity of Quantified Constraint Satisfaction Problems under Structural Restrictions *

Georg Gottlob

Inst. für Informationssysteme
Technische Universität Wien
A-1040 Vienna, Austria
gottlob@dbai.tuwien.ac.at

Gianluigi Greco

Dip. di Matematica
Università della Calabria
I-87030 Rende, Italy
ggreco@mat.unical.it

Francesco Scarcello

DEIS
Università della Calabria
I-87030 Rende, Italy
scarcello@deis.unical.it

Abstract

We give a clear picture of the tractability/intractability frontier for quantified constraint satisfaction problems (QCSPs) under structural restrictions. On the negative side, we prove that checking QCSP satisfiability remains PSPACE-hard for all known structural properties more general than bounded treewidth and for the incomparable hypergraph acyclicity. Moreover, if the domain is not fixed, the problem is PSPACE-hard even for tree-shaped constraint scopes. On the positive side, we identify relevant tractable classes, including QCSPs with prefix $\exists\forall$ having bounded hypertree width, and QCSPs with a bounded number of guards. The latter are solvable in polynomial time without any bound on domains or quantifier alternations.

1 Introduction

Quantified constraint satisfaction problems (QCSPs) are a generalization of constraint satisfaction problems (CSPs), where variables may be existentially and universally quantified, and nested quantifications are allowed. This framework is clearly much more expressive than plain existential-CSP, and may be fruitfully exploited for modeling a wide spectrum of problems from several domains.

A QCSP *instance* (or quantified constraint formula) ϕ is an expression of the form $Q_1\bar{V}_1 \cdots Q_m\bar{V}_m I$, where I is a constraint network (denoted by $CN(\phi)$), Q_i is a quantifier in $\{\exists, \forall\}$ (with $Q_i \neq Q_{i+1}$), and $\bar{V}_i$ is a set of variables, for $1 \leq i \leq m$. The string of quantifiers $Q_1 \cdots Q_m$ is called the *prefix* of ϕ . Recall that a *constraint network* is a triple $I = (Var, \mathcal{U}, \mathcal{C})$, where Var is a finite set of variables, $\mathcal{U}$ is the set of domains $U(V)$, for each variable $V \in Var$, and $\mathcal{C} = \{C_1, C_2, \dots, C_q\}$ is a finite set of constraints. A constraint $C_i = (S_i, r_i)$ consists of a list of variables S_i called *constraint scope*, and of a relation r_i , called *constraint relation*, providing C_i 's allowed combinations of values for the variables in its scope. Sometimes it is more comfortable to denote C_i by its so called constraint atom $r_i(S_i)$. Then, the

network I may be represented by the conjunction of all its constraint atoms. For simplicity, we limit our attention here to *closed* quantified constraint formulas, where all variables occurring in I are quantified. However, all our results may be easily extended to formulas with free variables.

As an example, consider the following quantified constraint formula ϕ_e : $\forall S, X, Y, T, R, U, P \exists V, Z \forall W a(S, X, T, R) \wedge b(S, Y, U, P) \wedge c(T, U, Z) \wedge d(W, X, Z) \wedge e(Y, Z) \wedge f(R, P, V) \wedge g(X, Y)$. This formula is a QCSP instance, whose constraint network $CN(\phi_e)$ is represented by the constraint atoms occurring in the conjunction. The quantifier prefix (short: prefix) of ϕ_e is the string $\forall\exists\forall$.

Not surprisingly, the increased expressive power of QCSPs comes at a cost. Indeed, while deciding the satisfiability of traditional (i.e., purely existential) CSPs is NP-complete, this problem is PSPACE-complete [Borner *et al.*, 2003], in the general quantified setting. Hence, much effort has been spent to identify *tractable classes* of QCSPs.

These approaches can be divided into two main groups: techniques that identify tractable classes of QCSPs by exploiting particular properties of constraint relations, and techniques that identify tractable classes by exploiting the structure of constraint scopes, usually known as *structural decomposition methods*. While several deep results have been already achieved by techniques in the former group (see, e.g., [Borner *et al.*, 2003; Bulatov *et al.*, 2000; Bunind *et al.*, 1995; Chen, 2004a; Creignou *et al.*, 2001; Jeavons *et al.*, 1997]), only a few papers focused on structural decomposition methods, though they were proven to be useful in the non-quantified setting (see, e.g., [Dechter, 2003; Gottlob *et al.*, 2000]).

Recall that the structure of constraint network I is best represented by its associated hypergraph $\mathcal{H}(I) = (V, H)$, where $V = Var$ and $H = \{var(S) \mid C = (S, r) \in \mathcal{C}\}$, and $var(S)$ denotes the set of variables in the scope S of constraint C . Some graph-based techniques are based on the *primal graph* $G(\mathcal{H}(I)) = (V, E)$ of $\mathcal{H}(I)$, where two variables are connected in E if they occur together in some hyperedge (i.e., in the scope of some constraint).

Chen recently presented an interesting result about structurally tractable QCSPs [Chen, 2004b]. He describes a polynomial-time algorithm for classes of QCSPs having (primal graphs with) bounded treewidth, fixed domain, and fixed prefix. In fact, the complexity of this algorithm depends

*This work was supported by the Austrian Science Fund (FWF) project: Nr. P17222-N04 *Complementary Approaches to Constraint Satisfaction*.

dramatically on the number of quantifier alternations and on the size of the largest variable domain. As noted in [Chen, 2004b], the same result has been independently derived by [Feder and Kolaitis, 2004], by exploiting Courcelle’s theorem about monadic second order logic on bounded treewidth structures.

Notice that there is no indication that these results are optimal, and in fact several interesting questions arose, and will be the subject of this paper:

(1) Are QCSPs having bounded treewidth tractable if domains are not fixed?

(2) May we extend this result to other structural notions, possibly more general than bounded treewidth?

(3) Are there different kind of restrictions on quantified constraint formulas that make QCSPs tractable?

The answers to these questions comprise both good news and bad news. We prove strong hardness results, but we also identify new tractable classes of QCSPs, having neither fixed bound on domains nor fixed bound on quantifier alternations. Our main contributions, shown in Figure 1, are the following:

- We prove that, without the fixed domain restriction, even QCSP instances whose structure is a tree and whose prefix is $\forall\exists$ are co-NP-hard. Moreover, adding further alternations we get complete problems for all levels of the polynomial hierarchy. It follows that this problem is PSPACE-complete if there are no bounds on the quantifier prefix.
- On the positive side, we prove that, if the prefix is $\exists\forall$ (or some substring of it), then solving acyclic QCSPs is feasible in LOGCFL and hence in polynomial time. Moreover, this tractability extends to all known generalizations of acyclicity, and in particular to bounded hypertree-width QCSPs [Gottlob *et al.*, 2000].
- We prove that, for fixed domains, the tractability result for bounded treewidth is almost optimal. Indeed, solving QCSPs over the binary domain $\{0, 1\}$ remains PSPACE-complete even if the structure is an acyclic hypergraph, whose incidence graph has bounded treewidth, and whose primal graph has small (i.e., logarithmic) treewidth.
- All these results show that traditional structural techniques do not help very much, but for some simple cases and with limited quantification. Indeed, our hardness proofs show that the presence of quantifiers radically alters the structural properties of the constraint scopes. We thus realize that it is worthwhile taking into account how they interact with the scope structure, and in fact considering quantifiers as part of the scope structure itself. Following this idea, we identify a different kind of restriction on quantified constraint formulas that ensure tractability and that is incomparable with the other structural classes. In particular, for any fixed k , we define the class k -GQCSP of k -guarded QCSPs, that are solvable in polynomial time, without any restriction on domains or quantifier alternations.

2 Quantified CSPs

Let $I = (Var, \mathcal{U}, \mathcal{C})$ be a constraint network. An assignment σ for a set of variables $\bar{V} \subseteq Var$ is a function mapping each variable $V \in \bar{V}$ onto its domain $U(V) \in \mathcal{U}$. If $\bar{V} = Var$, σ is said *complete*, otherwise it is a partial assignment. We say that a complete assignment σ satisfies I , denoted by $\sigma \models I$,

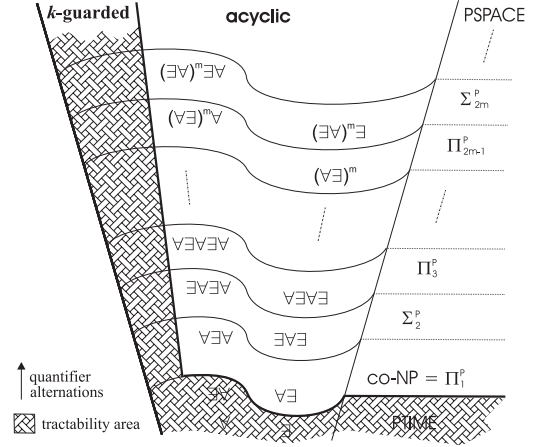


Figure 1: Structural restrictions and (in)tractable QCSPs.

if for each constraint $(S_i, r_i) \in \mathcal{C}$, $\sigma(S_i) \in r_i$. An extension of σ to a set $\bar{V}' \supset \bar{V}$ is an assignment σ' for $\bar{V}'$ such that $\sigma'(V) = \sigma(V)$ for each $V \in \bar{V}$. We denote by $ext(\sigma, \bar{V}')$ the set of all the extensions of σ to $\bar{V}'$. For the trivial assignment $\sigma_\emptyset$ for the empty set of variables, $ext(\sigma_\emptyset, \bar{V}')$ is clearly the set of all assignments for $\bar{V}'$.

Let $\phi : Q_1 \bar{V}_1 Q_2 \bar{V}_2 Q_3 \bar{V}_3 \dots Q_m \bar{V}_m I$ be a QCSP instance, and let σ_0 be the trivial assignment $\sigma_\emptyset$. A *strategy* for ϕ is any function s such that, for each pair Q_i, σ_{i-1} , with $1 \leq i \leq m$, $s(Q_i, \sigma_{i-1})$ is either one assignment in $ext(\sigma_{i-1}, \bar{V}_i)$, if $Q_i = \exists$, or the whole set of possible extensions $ext(\sigma_{i-1}, \bar{V}_i)$, if $Q_i = \forall$. A complete assignment σ_m is *derivable* from a strategy s if there are $m - 1$ assignments $\sigma_1, \dots, \sigma_{m-1}$ such that $\sigma_i \in s(Q_i, \sigma_{i-1})$, for any $1 \leq i \leq m$. Then, s is a *solution* for ϕ if all derivable assignments satisfy I . A QCSP instance is *satisfiable* iff it has a solution.

It is worthwhile noting that, in the definition of QCSPs, different variables have different domains, in general. This is especially useful in the quantified setting. However, in the literature, QCSPs are sometimes defined over a unique domain U or, equivalently, with the same domain $U(V)$ for each variable V . We say that such QCSPs are *untyped*, in contrast to the general ones, called *typed*. The following proposition shows that the two formalisms are in fact logically equivalent.

Proposition 2.1 *For any QCSP instance ϕ , there exists an untyped equivalent instance ϕ' . Moreover, if $CN(\phi)$ is a binary network, ϕ' can be computed in polynomial time.*

Notice that going from typed to untyped instances may be exponential for non-binary networks, as the former setting allows more succinct and efficient representations.

We remark that all complexity results in this paper hold for both settings. Indeed, we prove membership results and provide algorithms for the general typed setting, and prove hardness results by using either binary networks, or a unique binary domain for all variables.

We say that a class S of hypergraphs has the bounded hypertree width property, denoted by BHTW, if there is a $k > 0$ such that every hypergraph in S has hypertree width at most k [Gottlob *et al.*, 2000]. Similarly, we define the property BTW, meaning bounded treewidth [Robertson and Seymour, 1986] of the primal graph (of the constraint hypergraph), and the property BITW, meaning bounded treewidth of the inci-

dence graph. Moreover, we say that a class S of hypergraphs has the small hypertree width property, denoted by SHTW, if the hypertree width of every hypergraph $\mathcal{H} \in S$ is at most $\log |\mathcal{H}|$. The small treewidth property STW of primal graphs is defined similarly. We also consider the property ACYCLIC (resp., TREES) of any class of acyclic hypergraphs (resp., primal graphs).

We study how the complexity of QCSPs change as a function of quantifier alternations and of constraint structures. Moreover, we distinguish the case of arbitrary domains, denoted by ANY, and of binary domains, denoted by $\{0, 1\}$.

Let $\bar{Q}$ be a string of quantifier alternations, $\mathcal{S}$ a hypergraphs property, and $\mathcal{D}$ a domain property in $\{\text{ANY}, \{0, 1\}\}$. Then, $\text{QCSP}(\bar{Q}, \mathcal{S}, \mathcal{D})$ is the problem of deciding whether an instance $\phi \in \text{class}(\bar{Q}, \mathcal{S}, \mathcal{D})$ is satisfiable, where $\text{class}(\bar{Q}, \mathcal{S}, \mathcal{D})$ is any class of QCSP instances over domains of kind $\mathcal{D}$, with alternation prefix $\bar{Q}$, and whose associated hypergraphs have property $\mathcal{S}$.

3 Structural methods do not help very much

3.1 Some tractable instances

From [Gottlob *et al.*, 2000], we already know that $\text{QCSP}(\exists, \text{BHTW}, \text{ANY})$ is in polynomial time, and the same holds for any structural restriction stronger than bounded hypertree-width. Moreover, it is easy to see that $\text{QCSP}(\forall, \text{BHTW}, \text{ANY})$ is even easier. We next show that also $\text{QCSP}(\exists\forall, \text{BHTW}, \text{ANY})$ is tractable.

Let ϕ be a QCSP instance, $I = (\text{Var}, \mathcal{U}, \mathcal{C})$ the constraint network of ϕ , and $\bar{Y}$ be a set of universally quantified variables. Let $r(\bar{X}, \bar{Y}')$ a constraint of I over variable sets $\bar{X}$ and $\bar{Y}'$, where $\bar{Y}'$ is the set of $\bar{Y}$ variables occurring in the scope of r . Denote by $\text{cart}(\bar{Y}')$ the relation containing all combination of values from the domains of variables in $\bar{Y}'$, i.e., $\text{cart}(\bar{Y}') = \times_{Y \in \bar{Y}'} U(Y)$.

Then, define $\bar{Y}\text{-red}(r)$ as the relation containing all and only those tuples t of relation r such that, for any combination t' of values for variables $\bar{Y}'$, $t[\bar{X}] \cdot t' \in r$. Note that, for the special case $\bar{Y}' = \emptyset$, we get $\bar{Y}\text{-red}(r) = r$; for the special case $\bar{X}' = \emptyset$, we simply require all combinations of values for $\bar{Y}'$, that is, $\bar{Y}\text{-red}(r) = r$, if r is precisely $\text{cart}(\bar{Y}')$, otherwise it is the empty relation.

Denote by $\bar{Y}\text{-red}(\phi)$ the QCSP obtained from ϕ by replacing each constraint relation r by $\bar{Y}\text{-red}(r)$.

Lemma 3.1 *For any QCSP ϕ and set of universally quantified variables $\bar{Y}$, $\bar{Y}\text{-red}(\phi)$ can be computed in logspace.*

After the above lemma and exploiting the fact that no useful assignment is lost with this transformation, we can show the following.

Theorem 3.2 *$\text{QCSP}(\exists\forall, \text{BHTW}, \text{ANY})$ is in polynomial time. Moreover, it is LOGCFL-complete, and hence tractable and parallelizable.*

3.2 Encoding Boolean formulas as acyclic QCSPs

To any CNF formula $\Phi = c_1 \wedge \dots \wedge c_m$ over Boolean variables $\bar{V} = \{V_1, \dots, V_n\}$, we associate a binary acyclic constraint network $I(\Phi) = (\text{Var}, \mathcal{U}, \mathcal{C})$. This constraint network will be used hereafter for characterizing the complexity of acyclic and quasi-acyclic quantified CSPs.

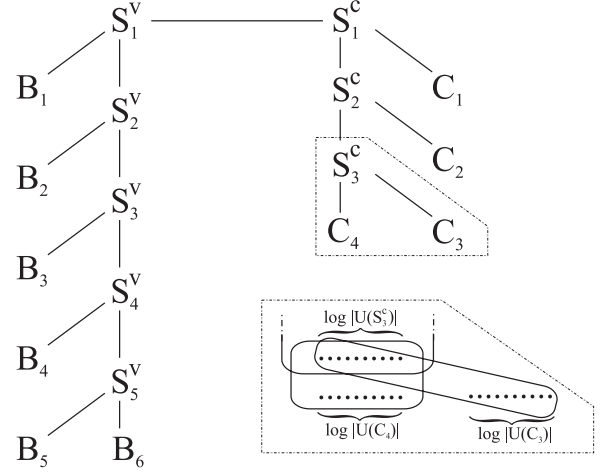


Figure 2: Constraint network $I(\bar{\Phi})$ in Theorem 3.4. In the right-bottom box, encodings of constraints in Theorem 3.7.

Consider the following CNF formula, that we use as a running example: $\bar{\Phi} = (V_1 \vee V_2 \vee V_3) \wedge (V_1 \vee \neg V_4) \wedge (V_1 \vee V_6 \vee V_4) \wedge (V_2 \vee V_6)$. Then, Figure 2 shows the constraint structure (i.e. the constraint hypergraph) of $I(\bar{\Phi})$. Note that in this case hypergraph and primal graph representations coincide, as $CN(\bar{\Phi})$ contains only binary constraints.

The set of variables Var is the union of a set of *clause variables* $\bar{C} = \{C_j \mid 1 \leq j \leq m\}$ corresponding to the m clauses of Φ , of a set $\bar{B} = \{B_i \mid V_i \in \bar{V}\}$, corresponding to the n Boolean variables occurring in Φ , and of two sets $\bar{S}^c = \{S_j^c \mid 1 \leq j \leq m-1\}$ and $\bar{S}^v = \{S_i^v \mid 1 \leq i \leq n-1\}$ of special variables, called *clause selectors* and *variable selectors*, respectively.

For a variable V_i of Φ , the domain of $I(\Phi)$ contains literal constants v_i and $\neg v_i$ associated with its truth-values. Moreover, for any clause c_j in which V_i occurs, the domain of $I(\Phi)$ contains a literal constant in $\{v_i^j, \neg v_i^j\}$ encoding the truth value for V_i that makes c_j true. We denote by l_i (resp., l_i^j) any of these (positive or negative) literals, and by $\neg l_i$ (resp., $\neg l_i^j$) its complement. Moreover, we denote by $\text{satLit}(c_j)$ the set of literals that make c_j true. E.g., for $c_2 = (V_1 \vee \neg V_4)$ in $\bar{\Phi}$, $\text{satLit}(c_2) = \{v_1^2, \neg v_4^2\}$.

In more detail, the domains of $I(\Phi)$ variables are the following: for any variable C_j (corresponding to clause c_j of Φ), $U(C_j) = \text{satLit}(c_j)$; for any Boolean variable B_i , $U(B_i) = \{v_i, \neg v_i\}$; for any clause-selector S_j^c , $U(S_j^c) = \bigcup_{j' \geq j} U(C_{j'})$; finally, for any variable-selector S_i^v , $U(S_i^v) = \bigcup_{i' \geq i} U(B_{i'})$.

Intuitively, Boolean variables encode a truth-value assignment to Φ , whereas any clause variable C_j chooses some literal in $\text{satLit}(c_j)$ that satisfies it. Any selector variable may take a value coming from either variable connected to it. Thus, any choice of all variable selectors corresponds to the propagation of a literal value l_i coming from some Boolean variable B_i in the left branch. Similarly, any choice of clause selectors corresponds to the propagation in the right branch of some literal l_k^j satisfying some clause c_j . If for all possible propagations from both branches, no pair $l_i, \neg l_i^j$ of comple-

mentary literals meets at the topmost constraint of the network, then the values of Boolean variables encode a satisfying truth-value assignment for Φ . We next describe the constraints in $\mathcal{C}$ that implement the above idea, where the indices i and i' actually are values in the interval $[1 \dots n]$ and where the indices j, j' are values in $[1 \dots m]$.

- The topmost constraint (S_1^v, S_1^c) , called *evaluate*, has a constraint relation consisting of $\{(l_i, \neg l_i^j) \mid l_i^j \in \text{satLit}(c_j)\}$. Note that *evaluate* is satisfied only by assignments where its variables take complementary literals.
- For any constraint (S_i^v, B_i) between a variable-selector S_i^v and a Boolean variable B_i , its constraint relation consists of the tuples: $\{(l_i, l_i)\} \cup \{(l_{i'}, l_i) \mid i' > i\}$.
- For any constraint between a pair of adjacent variable-selectors (S_i^v, S_{i+1}^v) and for the constraint (S_i^v, B_n) with $i = n - 1$, the constraint relation is $\{(l_{i'}, l_{i'}) \mid i' > i\} \cup \{(l_i, l_{i'}) \mid i' > i\}$.
- For any constraint (S_j^c, C_j) between a clause-selector S_j^c and a clause variable C_j , its constraint relation consists of the tuples: $\{(l_h^j, l_h^j) \mid l_h^j \in \text{satLit}(c_j)\} \cup \{(l_k^j, l_h^j) \mid l_h^j \in \text{satLit}(c_j), l_k^j \in \text{satLit}(c_{j'}), j' > j\}$.
- For any constraint between a pair of adjacent clause-selectors (S_j^c, S_{j+1}^c) and for the constraint (S_j^c, C_m) with $j = m - 1$, the constraint relation consists of the tuples: $\{(l_h^j, l_h^j) \mid l_h^j \in \text{satLit}(c_{j'})\} \cup \{(l_k^j, l_h^j) \mid l_k^j \in \text{satLit}(c_j), l_h^j \in \text{satLit}(c_{j'}), j' > j\}$.

If μ is a truth-value assignment for all variables $\bar{V}$ of Φ , then, σ_μ , denotes the assignment such that $\sigma_\mu(B_i) = v_i$ if $\mu(V_i) = \text{true}$, and $\sigma_\mu(B_i) = \neg v_i$ if $\mu(V_i) = \text{false}$.

Lemma 3.3 *Let μ be a truth-value assignment for all variables $\bar{V}$ of Φ and σ_μ the corresponding assignment for $\bar{B}$. Then, μ is a satisfying assignment for Φ if and only if there exists an assignment $\sigma' \in \text{ext}(\sigma_\mu, \bar{C})$ such that, for every $\sigma'' \in \text{ext}(\sigma', \bar{S}^v \cup \bar{S}^c)$, $\sigma'' \models I(\Phi)$ holds.*

After this lemma, we immediately get that, even for acyclic binary constraint networks with just two quantifier alternations, solving a quantified CSP is intractable.

Theorem 3.4 $\text{QCSP}(\exists\forall, \text{TREES}, \text{ANY})$ is co-NP-hard.

Proof. From a CNF Boolean formula Φ , we build in polynomial time $QC(\Phi) = \forall \bar{B}, \bar{C} \exists \bar{S} I(\Phi)$, where $I(\Phi)$ is the acyclic constraint network associated with Φ . From Lemma 3.3, satisfying assignments for Φ are in one-to-one correspondence with assignments to the variables in $\bar{B}$ and $\bar{C}$ such that all their complete extensions do not satisfy $I(\Phi)$. Thus, Φ is not satisfiable iff $QC(\Phi)$ is satisfiable. $\square$

3.3 Intractable acyclic instances

After having shown in the previous section the tractability for $\exists\forall$ and the intractability for $\forall\exists$, we now settle the complexity of acyclic QCSPs with arbitrary quantifier prefixes.

Theorem 3.5 *For any natural number $m \geq 1$,*

1. $\text{QCSP}((\forall\exists)^m\forall, \text{BHTW}, \text{ANY})$ is Π_{2m-1}^P -complete, and hardness holds for $\text{QCSP}((\forall\exists)^m, \text{TREES}, \text{ANY})$, too;
2. $\text{QCSP}((\exists\forall)^m\forall, \text{BHTW}, \text{ANY})$ is Σ_{2m}^P -complete, and hardness holds for $\text{QCSP}((\exists\forall)^m\exists, \text{TREES}, \text{ANY})$, too.

Proof. For space limitations we only prove *Hardness* for Point 2, here. For any $m \geq 1$, consider the Σ_{2m}^P -complete problem of deciding whether a quantified Boolean formula $\Psi' = \exists V_1 \forall V_2 \dots \forall V_{2m} \neg \Phi$ is satisfiable, where Φ is in CNF. From this formula, we build in polynomial time the following instance of $\text{QCSP}((\exists\forall)^m\exists, \text{TREES}, \text{ANY})$: $QC(\Psi') = \exists B_1 \forall B_2 \dots \forall B_{2m}, \bar{C} \exists \bar{S} I(\Phi)$. From Lemma 3.3, it can be seen easily that Ψ' is satisfiable iff $QC(\Psi')$ is satisfiable.

Membership. The proof is by induction. First observe that, given any instance $Q = \exists \bar{V}_1 \forall \bar{V}_2 \phi$ of $\text{QCSP}(\exists\forall, \text{BHTW}, \text{ANY})$, its complementary problem Q^c (deciding whether for all assignment σ to $\bar{V}_1$ there exists an extension to $\bar{V}_2$ that does not satisfy ϕ) is in LOGCFL and hence in polynomial time, by Theorem 3.2 and the fact that LOGCFL is closed under complementation. Then, we prove the basis of the induction, $m = 1$. The problem $\text{QCSP}(\forall\exists\forall, \text{BHTW}, \text{ANY})$ is in $\Pi_1^P = \text{co-NP}$. Indeed, let $Q = \forall \bar{V}_1 \exists \bar{V}_2 \forall \bar{V}_3 \phi$ be any instance of this problem. Then, its complement can be decided in NP: guess an assignment σ to $\bar{V}_1$ and check that for all assignment $\sigma' \in \text{ext}(\sigma, \bar{V}_2)$ there is a complete assignment $\sigma'' \in \text{ext}(\sigma', \bar{V}_3)$ that does not satisfy all the constraints in ϕ . From the observation above, this check is feasible in polynomial time. Moreover, $\text{QCSP}(\exists\forall\exists\forall, \text{BHTW}, \text{ANY})$ is in Σ_2^P , as any instance of this problem may be solved by a non-deterministic Turing machine with an oracle for $\text{QCSP}(\forall\exists\forall, \text{BHTW}, \text{ANY})$. The induction step is a simple adaptation of the above reasoning for any $m > 1$. $\square$

Corollary 3.6 *The quantified constraint satisfaction problem (on arbitrary domain) is PSPACE-complete, even if restricted on constraint networks whose structure is a tree.*

3.4 Fixed domain helps only with fixed arity

We now show that hypergraph acyclicity does not help in making easy the QCSP problem, even if we consider Boolean domains only. The same holds even in case we additionally require that the incidence graph has bounded treewidth, and the primal graph has small (logarithmic) treewidth. This entails that the problem remains intractable as long as we have non-fixed arities, even for very simple constraint interactions.

Theorem 3.7 *For any natural number $m \geq 1$,*

- $\text{QCSP}((\exists\forall)^m, \text{ACYCLIC} \cap \text{BITW} \cap \text{STW}, \{0, 1\})$ is Π_{2m-1}^P -complete;
- $\text{QCSP}((\forall\exists)^m\exists, \text{ACYCLIC} \cap \text{BITW} \cap \text{STW}, \{0, 1\})$ is Σ_{2m}^P -complete.

Proof. Let Φ be a Boolean formula, and $I(\Phi) = (\text{Var}, \mathcal{U}, \mathcal{C})$ its associated acyclic constraint network. We consider the network $I'(\Phi) = (\text{Var}', \{0, 1\}, \mathcal{C}')$, defined as follows. For each variable $X \in \text{Var}$, Var' contains $|U(X)|$ distinct variables $X_1, \dots, X_{\log |U(X)|}$, with domain $U'(X_i) = \{0, 1\}$ for each X_i . For each constraint $r(X, Y)$ in $\mathcal{C}$, $\mathcal{C}'$ contains a constraint $r'(X_1, \dots, X_{\log |U(X)|}, Y_1, \dots, Y_{\log |U(Y)|})$, whose constraint relation is such that, for each tuple (x_i, y_j) in r , r' contains the tuple $(\text{enc}(x_i), \text{enc}(y_j))$, where the string of bits $\text{enc}(x_i)$ (resp. $\text{enc}(y_j)$) is the binary encoding of domain value x_i (resp. y_j). The right-bottom box in Figure 2 shows a portion of the constraint network $I'(\Phi)$ associated to the formula Φ of our running example. Observe that the constraint network $I'(\Phi)$ is in $\text{ACYCLIC} \cap \text{BITW} \cap \text{STW}$. Indeed,

the hypergraph associated to $I'(\Phi)$ is acyclic, and the number of variables in each hyperedge is bounded by $2 \log c$, where c is the size of largest domain over all the variables in Var . Therefore, the treewidth of the primal graph is at most $2 \log c$. Moreover, it is easy to check that the treewidth of the incidence graph of $I'(\Phi)$ is 3. Finally, observe that there exists a one-to-one correspondence between assignments to variables in $I(\Phi)$ and in $I'(\Phi)$, and thus the result immediately follows from Theorem 3.5. $\square$

Corollary 3.8 *The quantified constraint satisfaction problem is PSPACE-complete, even if restricted on Boolean constraint networks whose structure is in ACYCLIC $\cap$ BITW $\cap$ STW.*

4 Guarded formulas and tractable CSPs

In this section, we describe a wide class of quantified constraint formulas that are tractable, even if there is no constant bound on domain sizes or quantifier alternations. Recall that any QCSP instance ϕ may be represented by a logical expression, as shown in Section 1 for QCSP ϕ_e . Technically, let us denote the pure logical formula of ϕ (without the encoding of relations, domains, etc.) by $form(\phi)$.

Following [Kolaitis et al., 2000], we denote by $FO_{\wedge,+}$ the fragment of first order sentences where arbitrary quantifications and conjunctions are allowed, but where negations and disjunctions are forbidden. They observed that the existential fragment $\exists FO_{\wedge,+}$ of $FO_{\wedge,+}$ has the same expressive power as the constraint satisfaction problems. By allowing any kind of quantifiers, this observation may be clearly extended to the connection between general $FO_{\wedge,+}$ formulas and quantified constraint formulas. Notice that, in this more general setting, there are different equivalent logical representation of the same instance. For example, one may use parentheses for distinguishing subformulas and delimiting quantifier scopes. In this section, we represent QCSPs by $FO_{\wedge,+}$ formulas that are not necessarily in the traditional prenex form. Notice, however, that each $FO_{\wedge,+}$ formula can be easily transformed into an equivalent prenex formula.

4.1 The fragment k -GQCSP of k -guarded QCSPs

We show that, for each constant k , a simple and appealing fragment of $FO_{\wedge,+}$ is decidable in polynomial time. Since we have no bound on the number of variables in a formula, we assume w.l.o.g. that each variable is quantified over only once, i.e., quantified variables are not reused. In the following, we denote by $free(\psi)$ the free variables of a logical formula ψ .

Definition 4.1 The class k -GQCSP of k -guarded QCSPs consists of those QCSPs instances ϕ whose formula $form(\phi)$ belongs to the fragment G_k^* of $FO_{\wedge,+}$ defined as follows. G_k^* is the smallest subset of $FO_{\wedge,+}$ such that:

- every atom belongs to G_k^* ;
- if ϕ_1 and $\phi_2 \in G_k^*$, then $\phi_1 \wedge \phi_2 \in G_k^*$;
- let $\alpha_1, \dots, \alpha_i$ be atoms, where $i \leq k$, and let ψ be a formula in G_k^* . If the free variables $free(\psi) \subseteq var(\alpha_1) \cup \dots \cup var(\alpha_i)$, then, for each tuple of variables $\bar{y}$ and each quantifier $Q \in \{\exists, \forall\}$, the formula $\psi' : Q\bar{y}(\alpha_1 \wedge \dots \wedge \alpha_i \wedge \psi)$ belongs to G_k^* . The set of atoms $\{\alpha_1, \dots, \alpha_i\}$ is referred to as the guard of ψ' and is denoted by $guard(\psi')$. $\square$

Example 4.2 Consider again QCSP instance ϕ_e presented in the Introduction, and the following equivalent instance, where $form(\phi_e)$ is rewritten as

$$\psi = \forall S, X, Y, T, R, U, P (a(S, X, T, R) \wedge b(S, Y, U, P) \wedge \exists V f(R, P, V) \wedge \exists Z (g(X, Y) \wedge c(T, U, Z) \wedge \forall W d(W, X, Z) \wedge e(Y, Z)))$$

This is a 2-guarded constraint formula, i.e., $\psi \in G_2^*$. The guard of formula ψ is $guard(\psi) = \{a(S, X, T, R), b(S, Y, U, P)\}$. For the formulas $\psi_1 = \exists V f(R, P, V)$, $\psi_2 = \exists Z (g(X, Y) \wedge c(T, U, Z) \wedge \forall W d(W, X, Z) \wedge e(Y, Z))$ and $\psi_3 = \forall W d(W, X, Z)$, we have the following guards: $guard(\psi_1) = \{f(R, P, V)\}$, $guard(\psi_2) = \{g(X, Y), c(T, U, Z)\}$, and $guard(\psi_3) = \{d(W, X, Z)\}$. $\square$

Note that the above definition of k -guardedness is congenial to the specific syntax of quantified CSPs and differs from that of k -guarded first order logic [Andreka et al., 1998; Grädel, 1999]. In the standard formalisms of the guarded fragment GF of first order logic or of the k -guarded fragment GF_k of first order logic [Gottlob et al., 2003], the guards of an existentially quantified subformula ψ are added conjunctively to ψ ($guard(\psi) \wedge \psi$), just as for G_k^* . However, the guards of a universal formula ψ are added in form of an implication: $guard(\psi) \rightarrow \psi$. This is much more natural, since these logics have negation and guardedness needs to be correctly preserved under negations. For example, the negation of a guarded formula $\neg \exists \bar{Y} (g(\bar{Y}) \wedge \psi(\bar{Y}))$ is logically equivalent to $\forall \bar{Y} (g(\bar{Y}) \rightarrow \neg \psi(\bar{Y}))$. Since the logic $FO_{\wedge,+}$ of constraint formulas has conjunction ($\wedge$) as unique binary connective, it is syntactically impossible to express an implication $guard(\psi) \rightarrow \psi$ in $FO_{\wedge,+}$ (in fact, this is also semantically impossible). On the other hand, since negation is missing in $FO_{\wedge,+}$, no problems involving “wrong guards” can arise through negation when using the natural (but non-standard) guards introduced for the above defined fragments G_k^* of $FO_{\wedge,+}$.

For the k -guarded fragment GF_k of first order logic (i.e., for the standard k -guarded fragment) the following tractability result was shown in [Gottlob et al., 2003]:

Proposition 4.3 *The combined complexity of evaluating a GF_k formula ϕ over a set of finite relations D is in $O(|\phi| \times |D|^k)$.*

Let us now show that also the class k -GQCSP of k -guarded QCSPs is tractable, notwithstanding the “nonstandard” guards for universally quantified subformulas.

Lemma 4.4 *There is an algorithm TRANSFORM which for each QCSP $\phi \in k$ -GQCSP computes a pair (D, ϕ^*) where D is a finite database and $\phi^* \in GF_k$, such that ϕ is satisfied iff $D \models \phi^*$. The TRANSFORM algorithm runs in logspace.*

Proof. Let $\phi \in k$ -GQCSP, and let R be the set of constraint relations of $CN(\phi)$. Consider a subformula $\psi' = \forall \bar{Y} (guard(\psi) \wedge \psi)$ of $form(\phi)$, where $\forall \bar{Y}$ is a maximal prefix of universally quantified variables, and where $guard(\psi)$ is a conjunction of m atoms $r_i(\bar{Y}_i, \bar{X}_i)$, with $1 \leq i \leq m \leq k$, where $\bar{Y}_i$ are the variables in its scope included in $\bar{Y}$, and $\bar{X}_i$ its other variables, hence $\bar{X}_i \cap \bar{Y} = \emptyset$. For $1 \leq i \leq m$, we denote by $r_i \downarrow$ the relation obtained from the constraint relation r_i by keeping all $\bar{Y}_i$ -columns and projecting out all $\bar{X}_i$ columns, i.e., $r_i \downarrow := \prod_{\bar{Y}_i} r_i$.

Assume that for some $1 \leq i \leq m$, $r_i \downarrow$ is not equal to the Cartesian product $\text{cart}(\bar{Y}_i)$ of $\bar{Y}_i$ domains. That is, $r_i \downarrow$ does not consist of precisely all possible combinations of constants for all $\bar{Y}$ variables occurring in r_i . Then one tuple $\bar{a}$ of such constants is not in $r_i \downarrow$ and thus the atom $r_i(\bar{Y}_i, \bar{X}_i)$ cannot be satisfied for the substitution $\bar{Y}_i = \bar{a}$. Since r_i occurs positively in ϕ , ϕ cannot be satisfied. The key observation is thus that, whenever ϕ is satisfied, for $1 \leq i \leq m$, $r_i \downarrow$ must be equal to $\text{cart}(\bar{Y}_i)$. Algorithm TRANSFORM starts by checking whether this is true. From Lemma 3.1, this check can be implemented to run in logspace: just compute $\bar{Y}\text{-red}(r_i \downarrow)$, and check that it is not empty. If this test fails for at least one guard relation r_i , then TRANSFORM outputs the formula *false*. Otherwise, let g_ψ be a new constraint atom with constraint scope $\bar{Y}$, whose relation is the join of the $r_i \downarrow$ relations. Now, consider the set of free variables $\bar{X} = \text{free}(\psi')$. If $\bar{X} \neq \emptyset$, since ϕ is a k -guarded formula, there are at most k atoms in ϕ that cover $\bar{X}$. Let $g_{\psi'}$ be a new constraint atom with constraint scope $\bar{X}$, whose relation is the projection over $\bar{X}$ of the join of these guard atoms. Moreover, let g be a new constraint atom with constraint scope $\bar{X} \cup \bar{Y}$, whose relation is the join of g_ψ and $g_{\psi'}$ (or just g_ψ , if $\bar{X}$ is empty). Note that, since k is fixed, TRANSFORM may compute g from its input ϕ in logspace. Then, TRANSFORM replaces our subformula $\psi' = \forall \bar{Y}(\text{guard}(\psi) \wedge \psi)$ by the equivalent subformula $\psi'' = \forall \bar{Y}(g \rightarrow \text{guard}(\psi) \wedge \psi)$. Note that this transformation does not change the set of free variables, as $\text{free}(\psi') = \text{free}(\psi'')$. Thus, the set of atoms that cover these variables and acts as the guard of ψ' in $\text{form}(\phi)$ will be the guard of ψ'' in the new formula.

After having done this for all universal guarded subformulas, TRANSFORM has generated a formula $\phi^* \in GF_k$. Let D the database consisting of all constraint relations of $CN(\phi)$ plus all relations like g computed for modifying the universal quantifications. It can be seen that ϕ is satisfiable iff $D \models \phi^*$. Moreover, the entire algorithm runs in logspace. $\square$

Theorem 4.5 *For each fixed k , the satisfiability of any QCSP in the class k -GQCSP can be checked in polynomial time.*

Proof. Follows from Lemma 4.4 and Proposition 4.3. $\square$

4.2 Extending the k -GQCSP fragment

When looking at the fragment k -GQCSP, and even more generally, at QCSPs in which universal quantifiers appear, we observe that the expressive power of universal quantification is rather poor. In fact, as already observed in the proof of Lemma 4.4, a subformula $\forall \bar{Y} r(\bar{Y}, \bar{X})$ can only be true if the projection over $\bar{Y}$ of r is equal to the cartesian product $\text{cart}(\bar{Y})$ representing the set of all tuples that can be composed from all possible domain elements from the respective domains. This is a rather stringent condition. On the other hand, by standard k -guards of the form $(r_1 \wedge r_2 \wedge \dots \wedge r_m) \rightarrow \psi$, we can express some other interesting properties. For example by standard guards, one may express an *inclusion dependency* stating that part of one constraint relation must be contained in another constraint relation. To add expressive power, we thus suggest to allow the use of standard guards together with conjunctively specified guards (for universally quantified subformulas). We thus

define the fragment k -GQCSP⁺ just as k -GQCSP in Definition 4.1 with the following addition: *If the free variables $\text{free}(\psi) \subseteq \text{var}(\alpha_1) \cup \dots \cup \text{var}(\alpha_i)$, then, for each tuple of variables $\bar{y}$ the formula $\psi' : \forall \bar{y}((\alpha_1 \wedge \dots \wedge \alpha_i) \rightarrow \psi)$ belongs to G_k^* .*

After the results in Section 4.1, it is easy to see that, for each fixed k , the satisfiability of any QCSP in the class k -GQCSP⁺ can be checked in polynomial time. Moreover, we are able to show that k -GQCSP⁺ is strictly more expressive than k -GQCSP (for space reasons, we defer the proof to the full paper).

References

- [Andreka et al., 1998] H. Andreka, J. van Benthem, and I. Nemeti. Modal languages and bounded fragments of predicate logic. *Journal of Philosophical Logic*, 27(3):217–274, 1998.
- [Borner et al., 2003] F. Borner, A. Bulatov, A. Krokhin, and P. Jeavons. Quantified Constraints: Algorithms and Complexity. In *Proc. of CSL'03*, pp. 58–70, 2003.
- [Bulatov et al., 2000] A. Bulatov, A. Krokhin, and P. Jeavons. Constraint satisfaction problems and finite algebras. In *Proc. of IICALP'00*, pp. 272–282.
- [Bunind et al., 1995] H.K. Buning, M. Karpinski, and A. Flogel. Resolution for Quantified Boolean Formulas. *Information and Computation* 117(1):12–18, 1995.
- [Chen, 2004a] H. Chen. Collapsibility and Consistency in Quantified Constraint Satisfaction. In *Proc. of AAAI'04*, pp. 155–160, 2004.
- [Chen, 2004b] H. Chen. Quantified Constraint Satisfaction and Bounded Treewidth. In *Proc. of ECAI'04*, pp. 161–165, 2004.
- [Creignou et al., 2001] N. Creignou, S. Khanna, and M. Sudan. Complexity Classifications of Boolean Constraint Satisfaction Problems. *SIAM Monographs on Discrete Mathematics and Applications*, 7, 2001.
- [Dechter, 2003] R. Dechter. *Constraint Processing*. Morgan Kaufman, 2003.
- [Feder and Kolaitis, 2004] T. Feder and P.G. Kolaitis. Closures and dichotomies for quantified constraints. Manuscript, 2004.
- [Gottlob et al., 2000] G. Gottlob, N. Leone, and F. Scarcello. A Comparison of Structural CSP Decomposition Methods. *Artificial Intelligence*, 124(2): 243–282, 2000.
- [Gottlob et al., 2001] G. Gottlob, N. Leone, and F. Scarcello. The complexity of acyclic conjunctive queries. *JACM*, 48(3):431–498, 2001.
- [Gottlob et al., 2003] G. Gottlob, N. Leone, and F. Scarcello. Robbers, marshals, and guards: game theoretic and logical characterizations of hypertree width. *JCSS*, 66(4):775–808, 2003.
- [Grädel, 1999] E. Grädel. On the restraining power of guards. *Journal of Symbolic Logic*, 64:1719–1742, 1999.
- [Jeavons et al., 1997] P. Jeavons, D. Cohen, and M. Gyssens. Closure Properties of Constraints. *JACM*, 44(4):527–548, 1997.
- [Kolaitis et al., 2000] Ph. G. Kolaitis and M. Y. Vardi. Conjunctive-Query Containment and Constraint Satisfaction. *JCSS*, 61(2):302–332, 2000.
- [Mamoulis and Stergiou, 2004] N. Mamoulis and K. Stergiou. Algorithms for Quantified Constraint Satisfaction Problems. In *Proc. of CP'04*, pp. 752–756.
- [Robertson and Seymour, 1986] N. Robertson and P.D. Seymour. Graph Minors II. Algorithmic aspects of tree width. *Journal of Algorithms*, 7:309–322, 1986.
- [Schaefer, 1978] T.J. Schaefer. The Complexity of Satisfiability Problems In *Proc. of STOC'78*, pp. 216–226, 1978.

DPLL with a Trace: From SAT to Knowledge Compilation

Jinbo Huang and Adnan Darwiche

Computer Science Department
University of California, Los Angeles
Los Angeles, CA 90095
{jinbo, darwiche}@cs.ucla.edu

Abstract

We show that the trace of an exhaustive DPLL search can be viewed as a compilation of the propositional theory. With different constraints imposed or lifted on the DPLL algorithm, this compilation will belong to the language of d-DNNF, FBDD, and OBDD, respectively. These languages are decreasingly succinct, yet increasingly tractable, supporting such polynomial-time queries as model counting and equivalence testing. Our contribution is thus twofold. First, we provide a uniform framework, supported by empirical evaluations, for compiling knowledge into various languages of interest. Second, we show that given a particular variant of DPLL, by identifying the language membership of its traces, one gains a fundamental understanding of the intrinsic complexity and computational power of the search algorithm itself. As interesting examples, we unveil the “hidden power” of several recent model counters, point to one of their potential limitations, and identify a key limitation of DPLL-based procedures in general.

1 Introduction

Knowledge compilation has been a key direction of research in automated reasoning [Selman and Kautz, 1991; Marquis, 1995; Selman and Kautz, 1996; Cadoli and Donini, 1997; Darwiche and Marquis, 2002]. When propositional theories are compiled into a suitable target language, some generally intractable queries may be answered in time polynomial in the size of the compilation. Compiling combinational circuits into OBDDs, for example, allows functional equivalence to be tested in polynomial time (or constant time if the same variable order is used) [Bryant, 1986]. More recent applications of compilation can be found in the fields of diagnosis and planning, involving the use of the DNNF language [Darwiche, 2001a; Barrett, 2004; Palacios *et al.*, 2005].

Propositional Satisfiability (SAT), on the other hand, has been an area of no less importance, or activity. Aside from its theoretical significance as the prototypical NP-complete problem, SAT finds practical applications in many areas of artificial intelligence and computer science at large. While SAT algorithms have substantially improved over the

decades, many of them continue to build on what is known as the DPLL search [Davis *et al.*, 1962] (for examples, see complete SAT solvers in the 2004 SAT Competition: <http://satlive.org/SATCompetition/2004/>).

This paper sets out to demonstrate a deep connection between SAT solving and knowledge compilation. In the first direction, we show how advances in search-based SAT algorithms will carry over, via an exhaustive version of DPLL, to compiling propositional theories into one of several tractable languages. We start by pointing out that the trace of an exhaustive DPLL search, recorded compactly as a DAG, can be viewed as a compiled representation of the input theory. With different constraints imposed or lifted on the search process, we then show that the compilation will be in the language of d-DNNF, FBDD, and OBDD, respectively. These languages are known to decrease in succinctness: A propositional theory may have a polynomial-size representation in d-DNNF, but not in FBDD; or in FBDD, but not in OBDD [Darwiche and Marquis, 2002].

In the second direction, we formulate two principles by which the intrinsic complexity and computational power of a DPLL-based exhaustive search is related to the language membership of its traces. Applying these principles, we point out that several recent model counters are doing enough work to actually compile theories into d-DNNF. We also discuss a potential limitation on the efficiency of these model counters, as well as knowledge compilers using similar algorithms, based on the fact that these algorithms only generate traces in a specific subset of d-DNNF. Finally, we note that the efficiency of all DPLL-based exhaustive search algorithms are inherently limited by their inability to produce traces beyond d-DNNF. This realization is significant because some important computational tasks, such as existential quantification of variables and computation of minimum-cardinality models, could be efficiently accomplished with a weaker representation known as DNNF, a strict superset of d-DNNF.

Under our uniform DPLL-based framework for knowledge compilation, the power of successful modern SAT techniques is harnessed, including sophisticated conflict analysis, clause learning, faster detection of unit clauses, and new branching heuristics. We also discuss caching methods specific to the needs of compilation and tailored to the desired target compilation language, as well as structure-based complexity guarantees. We finally relate our experimental results on imple-

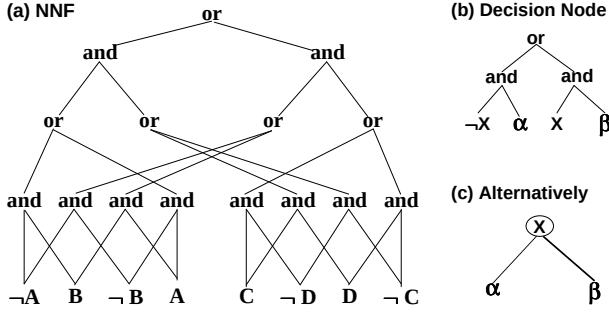


Figure 1: An NNF sentence and a decision node.

mentations of the three respective compilers to the theoretical succinctness relations of the corresponding languages.

The remainder of the paper is organized as follows. Section 2 reviews a number of propositional languages concerned in this work and their theoretical roles and relations in knowledge compilation. In Section 3, we describe a uniform framework for knowledge compilation, with regard to these languages, based on recording the trace of an exhaustive DPLL search, and discuss its implications on our understanding of the complexity and computational power of various search algorithms. We report experimental results in Section 4 and conclude in Section 5.

2 Target Compilation Languages

A logical form qualifies as a target compilation language if it supports some set of nontrivial queries, usually including clausal entailment, in polynomial time. We will review in this section several target compilation languages relevant to the present paper, and refer the reader to [Darwiche and Marquis, 2002] for a more comprehensive discussion.

The languages we shall describe are all subsets of the more general Negation Normal Form (NNF). A sentence in NNF is a propositional formula where conjunction (and, $\wedge$), disjunction (or, $\vee$) and negation (not, $\neg$) are the only connectives and negation only appears next to a variable; sharing of subformulas is permitted by means of a rooted DAG; see Figure 1a.

We start with the DNNF language, which is the set of all NNF sentences that satisfy **decomposability**: conjuncts of any conjunction share no variable. Our next language, d-DNNF, satisfies both decomposability and **determinism**: disjuncts of any disjunction are pairwise logically inconsistent. The formula of Figure 1a, for example, is in d-DNNF.

The FBDD language is the subset of d-DNNF where the root of every sentence is a **decision** node, which is defined recursively as either a constant (0 or 1) or a disjunction in the form of Figure 1b where X is a propositional variable and α and β are decision nodes. Note that an equivalent but more compact drawing of a decision node—shown in Figure 1c—is widely used in the formal verification literature, where FBDDs are equivalently known as Binary Decision Diagrams (BDDs) that satisfy the **test-once** property: each variable appears at most once on any root-to-sink path [Gergov and Meinel, 1994]. See Figure 2c for an FBDD example using this more compact drawing.

Table 1: Polytime queries supported by a language. $\circ$ means “not supported unless $P=NP$ ” and $?$ means “do not know.”

Lang.	CO	VA	CE	IM	EQ	SE	CT	ME
DNNF	✓	$\circ$	✓	$\circ$	$\circ$	$\circ$	$\circ$	✓
d-DNNF	✓	✓	✓	✓	?	$\circ$	✓	✓
FBDD	✓	✓	✓	✓	?	$\circ$	✓	✓
OBDD	✓	✓	✓	✓	✓	$\circ$	✓	✓
OBDD $_{<}$	✓	✓	✓	✓	✓	✓	✓	✓

The OBDD language is the subset of FBDD where all sentences satisfy the **ordering** property: variables appear in the same order on all root-to-sink paths [Bryant, 1986]. See Figure 2d for an OBDD example. For a particular variable order $<$, we also write OBDD $_{<}$ to denote the corresponding OBDD subset where all sentences use order $<$.

Having an option among target compilation languages is desirable, despite their succinctness relations which may be known. The reason is that the succinctness of a language often runs counter to its tractability—that is, the set of queries supported in polynomial time—and the best choice may depend on the task at hand. Given this trade-off between the two criteria, the rule of thumb, according to [Darwiche and Marquis, 2002], is to choose the most succinct language that supports the desired set of queries in polynomial time (in some cases the support of *transformations*, such as Boolean operations, is also a consideration).

Table 1 lists a set of polynomial-time queries of interest supported by each of these languages; the two-letter abbreviations stand for the following eight queries, respectively: **C**onsistency, **V**alidity, **C**lausal **E**ntailment, **I**mplicant, **E**quivalence, **S**entential **E**ntailment, **m**odel **C**ounting, **M**odel **E**numeration [Darwiche and Marquis, 2002].

Interestingly, this table offers one explanation for the popularity of OBDDs in formal verification where efficient equivalence testing, among other things, is often critical. Although more succinct, d-DNNF and FBDD are not known to admit a polynomial-time equivalence test (a polynomial-time probabilistic equivalence test is possible [Darwiche and Huang, 2002; Blum *et al.*, 1980]). Note also that although there is no difference between d-DNNF and FBDD to the extent of this table, the question mark on the equivalence test (EQ) could eventually be resolved differently for the two languages.

In the following section we will use the notion of recording the trace of a DPLL search to establish an important link between SAT and knowledge compilation, providing a uniform framework for compiling knowledge into some of these languages. From this point of view we will then discuss our new understanding of the complexity and computational power of algorithms based on exhaustive DPLL.

3 DPLL with a Trace

We start with the basic DPLL search as in [Davis *et al.*, 1962]. To facilitate our subsequent discussion of variants of this algorithm, we will omit unit resolution from the pseudocode. (All our discussions, however, are valid in the presence of unit resolution; see also the following two footnotes.)

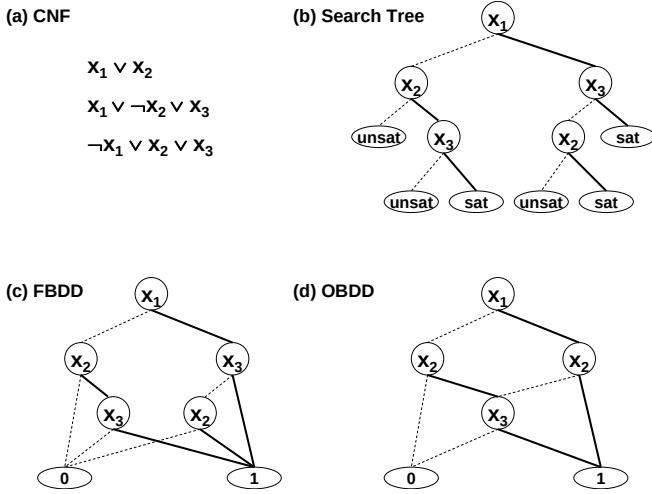


Figure 2: Exhaustive DPLL with a trace.

Algorithm 1 is a summary of DPLL for SAT. It works by recursively doing a case analysis on the assignment to a selected variable (Lines 5&6): The theory is satisfiable if and only if either case results in a satisfiable subtheory. $\Delta|_{x=0}$ ($\Delta|_{x=1}$) denotes the CNF obtained by replacing all occurrences of x with 0 (1) in Δ and simplifying the formula accordingly. In effect, this algorithm performs a search in the space of variable assignments until it finds one that satisfies the given CNF formula or realizes that no satisfying assignment exists.¹

Now consider extending Algorithm 1 so that it will enumerate all satisfying assignments—by always exploring both branches of Line 6—rather than terminate on finding the first one. Figure 2b depicts the search tree of this exhaustive version of DPLL on the CNF of Figure 2a, under some particular variable ordering. We are using a dotted (solid) line to denote setting the variable to 0 (1), and will refer to the corresponding branch of the search as the *low* (*high*) branch. Note that each leaf of this tree gives a partial variable assignment that satisfies the theory regardless of the values of any unassigned variables, and the whole tree characterizes precisely the set of all satisfying assignments.

¹To incorporate unit resolution into this simplified picture where all assignments are by decision, one can assume that when a chosen decision variable (Line 5 of Algorithm 1) has been implied by unit resolution, the algorithm will simply proceed down the right branch according to the implied value of the variable, noting that the other branch leads to unsatisfiability. One may also assume that in choosing a decision variable, those already implied by unit resolution will be favored, although this would represent a restriction on the search traces, which we describe shortly, that can be possibly generated.

Algorithm 1 $\text{DPLL}(\Delta)$: returns satisfiability of CNF Δ

```

1: if there is an empty clause in  $\Delta$  then
2:   return 0
3: if there is no variable in  $\Delta$  then
4:   return 1
5: select variable  $x$  of  $\Delta$ 
6: return  $\text{DPLL}(\Delta|_{x=0})$  or  $\text{DPLL}(\Delta|_{x=1})$ 

```

As the title of this paper suggests, we like to think of this complete search tree, also known as a *termination tree* or *decision tree*, as the *trace* left by the exhaustive DPLL search. Such a trace corresponds to the portion of the search space actually explored by a particular execution of the algorithm. Furthermore, the trace can be viewed as a compiled representation of the original CNF formula, because it uniquely identifies the propositional theory—by specifying its models.

From the viewpoint of knowledge compilation, however, a search trace recorded in its present form may not be immediately useful, because it will typically have a size proportional to the amount of work done to produce it. Answering even a linear-time query on such a compilation, for example, would be as if one were running the whole search over again.

This problem can be remedied by reducing the trace from a tree to a DAG, with repeated applications of the following two rules: (i) Isomorphic nodes (i.e., nodes that have the same label, same low child, and same high child) are merged; (ii) Any node with identical children is deleted and pointers to it are redirected to either one of its children [Bryant, 1986].

If we apply these reduction rules to the tree of Figure 2b, and rename “sat/unsat” to “1/0”, we will get the DAG of Figure 2c (in this particular example the first rule only applies to the terminal nodes and the second rule does not apply). Observe that this DAG is none other than a propositional theory in the language of FBDD. And this is no accident!

3.1 Compiling CNF into FBDD

We are in fact in possession of a CNF-to-FBDD compiler, described more formally in Algorithm 2. The main difference from the original DPLL is on Line 6: we always explore both branches, and the newly introduced function, **get-node**, provides a means of recording the trace of the search in the form of a DAG. Specifically, get-node will return a decision node labeled with the first argument, having the second argument as the low child, and having the third argument as the high child. Note that Lines 2&4 have been modified to return the terminal decision nodes, instead of the Boolean constants.

We point out that the amount of space required by Algorithm 2 to store the FBDD is only proportional to the size of the final result. In other words, it will never create redundant nodes to be reduced later. This is because the two reduction rules are built-in by means of a **unique nodes** table, well-known in the BDD community [Somenzi, Release 240]. Specifically, all nodes created by get-node are stored in a hash table and get-node will not create a new node if (i) the node to be created already exists in the table (that existing node is returned); or (ii) the second and third arguments are the same (either argument is returned).

Algorithm 2 $\text{dpll}_f(\Delta)$: compiles CNF Δ into FBDD (dpll is lowered to distinguish it from Algorithm 3, its full version)

```

1: if there is an empty clause in  $\Delta$  then
2:   return 0-sink
3: if there is no variable in  $\Delta$  then
4:   return 1-sink
5: select variable  $x$  of  $\Delta$ 
6: return get-node( $x$ ,  $\text{dpll}_f(\Delta|_{x=0})$ ,  $\text{dpll}_f(\Delta|_{x=1})$ )

```

Caching. Despite the use of unique nodes which controls the space complexity, Algorithm 2 still has a time complexity proportional to the size of the tree version of the search trace: Portions of the DAG can end up being explored multiple times. To alleviate this problem, one resorts to *formula caching* [Majercik and Littman, 1998].

Algorithm 3 describes the same CNF-to-FBDD compiler, but now with caching: The result of a recursive call $DPLL_f(\Delta)$ will be stored in a cache (Line 10) before being returned, indexed by a key (computed on Line 5) identifying Δ ; any subsequent call on some Δ' will immediately return the existing compilation for Δ from the cache (Line 7) if Δ' is found to be equivalent to Δ (by a key comparison on Line 6).

In practice, one normally focuses on efficiently recognizing formulas that are *syntactically* identical (i.e., have the same set of clauses). Various methods have been proposed for this purpose in recent years, starting with [Majercik and Littman, 1998] who used caching for probabilistic planning problems, followed by [Darwiche, 2002] who proposed a concrete formula caching method in the context of knowledge compilation, then [Bacchus *et al.*, 2003; Sang *et al.*, 2004] in the context of model counting, and then [Huang and Darwiche, 2004; Darwiche, 2004] who proposed further refinements on [Darwiche, 2002].

3.2 From FBDD to OBDD

We now turn to OBDD as our target compilation language. Note that in Algorithm 3, DPLL is free to choose any variable on which to branch (Line 8). This corresponds to the use of a dynamic variable ordering heuristic in a typical SAT solver, in keeping with the spirit of FBDD compilation.

Not surprisingly, a CNF-to-OBDD compiler can be obtained by switching from dynamic to static variable ordering: The new compiler will take a particular variable order π as a second argument, and make sure that this order is enforced when constructing the DAG. See Line 8 of Algorithm 4.

Caching. Naturally, any general formula caching method, such as the ones we described earlier, will be applicable to Algorithm 4. For this more constrained compiler, however, a special method is available where shorter cache keys can be used to reduce the cost of their manipulation. The reader is referred to [Huang and Darwiche, 2004] for details of this method, which allows one to bound the number of distinct cache keys, therefore providing both a space and a time complexity bound. In particular, due to this specific caching scheme, the space and time complexity of compiling OBDDs

was shown to be exponential only in the *cutwidth* of the given CNF. A variant caching scheme allows one to show a parallel complexity in terms of the *pathwidth* (cutwidth and pathwidth are not comparable).

Classical OBDD Construction. We emphasize here that Algorithm 4 represents a distinct way of OBDD construction, in contrast to the standard method widely adopted in formal verification where one recursively builds OBDDs for components of the system (or propositional theory) to be compiled and combines them using the *Apply* operator [Bryant, 1986]. A well-known problem with this latter method is that the intermediate OBDDs that arise in the process can grow so large as to make further manipulation impossible, even when the final result would have a tractable size. Considering that the final OBDD is really all that one is after, Algorithm 4 affords a solution to this problem by building exactly it, no more and no less (although it may do more work than is linear in the OBDD size, both because inconsistent subproblems do not contribute to the OBDD size, and because the caching is not complete). An empirical comparison of this compilation algorithm and the traditional OBDD construction method can be found in [Huang and Darwiche, 2004].

3.3 From FBDD to d-DNNF

Although any FBDD is also a d-DNNF sentence by definition, it remains a reasonable option to compile propositional theories into d-DNNF only, given its greater succinctness. Considering that d-DNNF is a relaxation of FBDD, we can obtain a d-DNNF compiler by relaxing a corresponding constraint on Algorithm 3. Specifically, immediately before Line 8, we need not insist any more that a case analysis be performed on some variable x of the formula; instead, the following technique of **decomposition** can be utilized. See Algorithm 5.²

As soon as variable instantiation finishes without contradiction, we will examine the remaining CNF formula, and partition it into subsets that do not share a variable (Line 5). These subsets can then be recursively compiled into d-DNNF (Lines 7–9) and conjoined as an **and-node** (Line 10). Note that decomposition takes precedence over case analysis: Only when no decomposition is possible do we branch on a selected variable as in regular DPLL (Lines 14&15).

²When unit resolution and clause learning are both integrated into this algorithm, an issue arises regarding implications via learned clauses that span otherwise disjoint components. See [Sang *et al.*, 2004] for more discussion on this issue.

Algorithm 3 $DPLL_f(\Delta)$: compiles CNF Δ into FBDD

```

1: if there is an empty clause in  $\Delta$  then
2:   return 0-sink
3: if there is no variable in  $\Delta$  then
4:   return 1-sink
5:  $key = \text{compute-key}(\Delta)$ 
6: if ( $result = \text{cache-lookup}(key)$ )  $\neq$  null then
7:   return  $result$ 
8: select variable  $x$  of  $\Delta$ 
9:  $result = \text{get-node}(x, DPLL_f(\Delta|_{x=0}), DPLL_f(\Delta|_{x=1}))$ 
10:  $\text{cache-insert}(key, result)$ 
11: return  $result$ 

```

Algorithm 4 $DPLL_o(\Delta, \pi)$: compiles CNF Δ into $OBDD_\pi$

```

1: if there is an empty clause in  $\Delta$  then
2:   return 0-sink
3: if there is no variable in  $\Delta$  then
4:   return 1-sink
5:  $key = \text{compute-key}(\Delta)$ 
6: if ( $result = \text{cache-lookup}(key)$ )  $\neq$  null then
7:   return  $result$ 
8:  $x = \text{first variable of order } \pi \text{ that appears in } \Delta$ 
9:  $result = \text{get-node}(x, DPLL_o(\Delta|_{x=0}, \pi), DPLL_o(\Delta|_{x=1}, \pi))$ 
10:  $\text{cache-insert}(key, result)$ 
11: return  $result$ 

```

Note that our relaxation of Algorithm 3 has resulted in a new type of node, returned by **get-and-node** on Line 10. The old get-node function (Line 15) still returns decision nodes (in a relaxed sense, as their children now are not necessarily decision nodes) in the form of Figure 1c. The unique nodes technique can also be extended in a straightforward way so that isomorphic and-nodes will not be created.

We point out here that Algorithm 5 only **produces sentences in a subset of d-DNNF**, because it only produces a special type of disjunction nodes—decision nodes (again in the relaxed sense). Recall that d-DNNF allows any disjunction as long as the disjuncts are pairwise logically inconsistent. We will come back to this in the next subsection.

Static vs. Dynamic Decomposition. Algorithm 5 suggests a dynamic notion of decomposition, where disjoint components will be recognized after each variable split. This dynamic decomposition was initially proposed and utilized by [Bayardo and Pehoushek, 2000] for model counting and adopted by a more recent model counter [Sang *et al.*, 2004]. [Darwiche, 2002; 2004] proposed another (static) method for performing the decomposition by preprocessing the CNF to generate a *decomposition tree (dtree)*, which is a binary tree whose leaves correspond to the CNF clauses. Each node in the dtree defines a set of variables, whose instantiation is guaranteed to decompose the CNF into disjoint components. The rationale is that the cost of dynamically computing a partition (Line 5) many times during the search is now replaced with the lesser cost of computing a static and recursive partition once and for all. This method of decomposition allows one to provide structure-based computational guarantees as discussed later, and can be orders of magnitude more efficient on some benchmarks, including the ISCAS85 circuits.³

³One may obtain results to this effect by running the model counter [Sang *et al.*, 2004], version 1.1, available at <http://www.cs.washington.edu/homes/kautz/Cachet/>, on benchmarks used in [Darwiche, 2004]. It should be noted that the two programs differ in other aspects, but the decomposition method appears to be the major difference. Note also that using DPLL for compilation incurs higher overhead than for model counting due to the bookkeeping involved in storing the DPLL trace.

Algorithm 5 $\text{DPLL}_d(\Delta)$: compiles CNF Δ into d-DNNF

```

1: if there is an empty clause in  $\Delta$  then
2:   return 0-sink
3: if there is no variable in  $\Delta$  then
4:   return 1-sink
5:  $\text{components} = \text{disjoint partitions of } \Delta$ 
6: if  $|\text{components}| > 1$  then
7:    $\text{conjuncts} = \{\}$ 
8:   for all  $\Delta_c \in \text{components}$  do
9:      $\text{conjuncts} = \text{conjuncts} \cup \{\text{DPLL}_d(\Delta_c)\}$ 
10:  return  $\text{get-and-node}(\text{conjuncts})$ 
11:  $\text{key} = \text{compute-key}(\Delta)$ 
12: if  $(\text{result} = \text{cache-lookup}(\text{key})) \neq \text{null}$  then
13:   return  $\text{result}$ 
14: select variable  $x$  of  $\Delta$ 
15:  $\text{result} = \text{get-node}(x, \text{DPLL}_d(\Delta|_{x=0}), \text{DPLL}_d(\Delta|_{x=1}))$ 
16:  $\text{cache-insert}(\text{key}, \text{result})$ 
17: return  $\text{result}$ 

```

Caching. Several caching methods have been proposed for d-DNNF compilation, the latest and most effective of which appeared in [Darwiche, 2004]. However, we refer the reader to [Darwiche, 2001b] for a caching scheme that is specific to the dtree-based decomposition method. This scheme is not competitive with the one in [Darwiche, 2004] in that it may miss some equivalences that would be caught by the latter, yet it allows one to show that the space and time complexity of d-DNNF compilation is exponential only in the *treewidth* of the CNF formula (as compared to the pathwidth and cutwidth in OBDD compilation). Interestingly, no similar structure-based measure of complexity appears to be known for FBDDs.

Relation to AND/OR Search. Recent work has explored the long established notion of AND/OR search to process queries on belief and constraint networks [Dechter and Mateescu, 2004b; 2004a]. An AND/OR search is characterized by a search graph with alternating layers of and-nodes and decision-nodes, the former representing decomposition and the latter branching. The DAGs produced by Algorithm 5 are indeed AND/OR graphs and, conversely, the AND/OR search algorithms described in [Dechter and Mateescu, 2004b; 2004a] can be used to **compile** networks into the multi-valued equivalent of d-DNNF. This implies that these AND/OR search algorithms are capable of many more tasks than what they were proposed for—model counting (or other equivalent tasks such as computing the probability of a random variable assignment satisfying the constraint query). We discuss this further in the following subsection.

3.4 Understanding the Power and Limitations of DPLL

The main proposal in this paper has been the view of exhaustive-DPLL traces as sentences in some propositional language. This view provides a unified framework for knowledge compilation as we have shown earlier, but we now show another major benefit of this framework: By using known results about the succinctness and tractability of languages, one can understand better the intrinsic complexity and computational power of various exhaustive DPLL procedures.

Consider a particular variation of DPLL, say DPLL_x , and suppose that its traces belong to language L_x . We then have:

1. DPLL_x will not run in polynomial time on formulas for which no polynomial-size representation exists in L_x .
2. If DPLL_x runs in polynomial time on a class of formulas, then DPLL_x (with some trivial modification) can answer in polynomial time any query on these formulas that is known to be tractable for language L_x .

Take for example the model counters recently proposed in [Bayardo and Pehoushek, 2000; Sang *et al.*, 2004], which employ the techniques of decomposition and (the latter also) caching. A simple analysis of these model counters shows that their traces are in the d-DNNF language (for specific illustrations, see the DDP algorithm of [Bayardo and Pehoushek, 2000] and Table 1 of [Sang *et al.*, 2004]). Therefore, neither of these model counters will have a polynomial time complexity on formulas for which no polynomial-size representations exist in d-DNNF.

In fact, one can take this analysis one step further as follows. These model counters, and the compiler of [Darwiche, 2004], actually produce traces in a strict subset of d-DNNF, call it d-DNNF', which employs a syntactic notion of determinism; that is, every disjunction in their trace has the form $(x \wedge \alpha) \vee (\neg x \wedge \beta)$, where x is a splitting variable. The d-DNNF language, however, does not insist on syntactic determinism: It allows disjunctions $\eta \vee \psi$ where $\eta \wedge \psi$ is logically inconsistent, yet η and ψ do not contradict each other on any particular variable x . If d-DNNF' turns out to be not as succinct as d-DNNF, then one may find another generation of model counters and d-DNNF compilers that can be exponentially more efficient than the current ones.

As an example of the second principle above, consider the query of testing whether the minimization of a theory Δ implies a particular clause α , $\min(\Delta) \models \alpha$, where $\min(\Delta)$ is defined as a theory whose models are exactly the minimum-cardinality models of Δ . This query is at the heart of diagnostic and nonmonotonic reasoning and is known to be tractable if Δ is in d-DNNF. Therefore, this query can be answered in polynomial time for any class of formulas on which the model counters in [Bayardo and Pehoushek, 2000; Sang *et al.*, 2004] have a polynomial time complexity. Similarly, a probabilistic equivalence test can be performed in polynomial time for formulas on which these models counters run in polynomial time.

Beyond DPLL. DPLL traces are inherently bound to be NNF sentences that are both deterministic and decomposable. Decomposability alone, however, is sufficient for the tractability of such important tasks as clausal entailment testing, existential quantification of variables, and cardinality-based minimization [Darwiche and Marquis, 2002]. DPLL cannot generate traces in DNNF that are not in d-DNNF, since variable splitting (the heart of DPLL) amounts to enforcing determinism. It is this property of determinism that provides the power needed to do model counting (#SAT), which is essential for applications such as probabilistic reasoning. But if one does not need this power, then one should go beyond DPLL-based procedures; otherwise one would be solving a harder computational problem than is necessary.

4 Experimental Results

By way of experimentation, we compiled a set of CNF formulas into OBDD, FBDD and d-DNNF, both to show the practicality of the DPLL-based compilation approach and to relate the results to the theoretical succinct relations of the three languages. The benchmarks we used include random 3-CNF and graph coloring problems from [Hoos and Stützle, 2000], and a set of ISCAS89 circuits.⁴ The compilation was done with the OBDD compiler of [Huang and Darwiche, 2004], using the MINCE variable order [Aloul *et al.*, 2001], an FBDD compiler we implemented according to Subsection 3.1 using the VSIDS variable ordering heuristic [Moskewicz *et al.*, 2001], and the d-DNNF compiler of [Darwiche, 2004].

⁴The CNF formulas used for these sequential circuits model the functionality of their combinational parts; they also define the transition relations of the circuits. Compilations of these theories will thus be useful, for example, for reachability analysis of the circuits.

Table 2: Compiling CNF into OBDD, FBDD, and d-DNNF.

CNF Name	Number of Models	OBDD		FBDD		d-DNNF	
		Size	Time	Size	Time	Size	Time
uf75-01	2258	10320	0.14	3684	0.02	822	0.02
uf75-02	4622	22476	0.15	14778	0.04	1523	0.03
uf75-03	3	450	0.02	450	0.02	79	0.01
uf100-01	314	2886	2.22	2268	0.01	413	0.02
uf100-02	196	1554	0.91	1164	0.07	210	0.04
uf100-03	7064	12462	0.78	9924	0.12	1363	0.02
uf200-01	112896	8364	651.04	7182	35.93	262	3.66
uf200-02	1555776	—	—	12900	33.72	744	2.64
uf200-03	804085558	—	—	662238	56.61	86696	10.64
flat75-1	24960	23784	0.16	10758	0.04	2273	0.01
flat75-2	774144	13374	0.28	8844	0.04	1838	0.01
flat75-3	25920	84330	0.29	26472	0.07	4184	0.04
flat100-1	684288	62298	0.78	37704	0.10	3475	0.03
flat100-2	245376	88824	1.57	39882	0.30	6554	0.09
flat100-3	11197440	15486	0.15	21072	0.09	2385	0.02
flat200-1	5379314835456	—	—	—	—	184781	56.86
flat200-2	13670940672	—	—	134184	7.07	9859	23.81
flat200-3	15219560448	—	—	358092	4.13	9269	3.28
s820	8388608	1372536	72.99	364698	0.69	23347	0.07
s832	8388608	1147272	76.55	362520	0.70	21395	0.05
s838.1	73786976294838206464	87552	0.24	—	—	12148	0.02
s953	35184372088832	2629464	38.81	1954752	4.01	85218	0.26
s1196	4294967296	4330656	78.26	4407768	12.49	206830	0.44
s1238	4294967296	3181302	158.84	4375122	12.14	293457	0.94
s1423	2475880078570760549798248448	—	—	—	—	738691	4.75
s1488	16384	6188427	50.35	388026	1.14	51883	0.19
s1494	16384	3974256	31.67	374760	1.07	55655	0.18

The results of these experiments are shown in Table 2, where the running times are given in seconds based on a 2.4GHz CPU. The size of the compilation reflects the number of edges in the NNF DAG. A dash indicates that the compilation did not succeed given the available memory (4GB) and a 900-second time limit. It can be seen that for most of these propositional theories, the compilation was the smallest in d-DNNF, then FBDD, then OBDD; a similar relation can be observed among the running times. Also, the number of instances successfully compiled was the largest for d-DNNF, then FBDD, then OBDD. This tracks well with the theoretical succinctness relations of the three languages. (However, note that FBDD and d-DNNF are not canonical representations and therefore compilations smaller than reported here are perfectly possible; smaller OBDD compilations are, of course, also possible under different variable orderings.)

We close this section by noting that the implementations of these knowledge compilers bear witness to the advantage of the DPLL-based framework we have described. The first compiler is based on an existing SAT solver [Moskewicz *et al.*, 2001], and the other two on our own implementation of DPLL, all three benefiting from techniques that have found success in SAT, including conflict analysis, clause learning, and data structures for efficient detection of unit clauses.

5 Conclusion

We established an important relationship between SAT and knowledge compilation, based on studying the trace of an ex-

haustive DPLL search. This relationship provides a uniform framework for compiling propositional theories into various languages of interest, and throws light on the intrinsic complexity and computational power of various DPLL-based algorithms. As interesting examples, we unveiled the “hidden power” of several recent model counters and discussed one of their potential limitations. We also pointed out the inability of exhaustive DPLL to produce traces in strict DNNF, which limits its power from a knowledge compilation point of view.

Acknowledgments

We thank the reviewers for commenting on an earlier version of this paper. This work has been partially supported by NSF grant IIS-9988543 and MURI grant N00014-00-1-0617.

References

- [Aloul *et al.*, 2001] Fadi Aloul, Igor Markov, and Karem Sakallah. Faster SAT and smaller BDDs via common function structure. In *International Conference on Computer Aided Design (ICCAD)*, pages 443–448, 2001.
- [Bacchus *et al.*, 2003] Fahiem Bacchus, Shannon Dalmao, and Toniann Pitassi. Algorithms and complexity results for #SAT and Bayesian inference. In *44th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 340–351, 2003.
- [Barrett, 2004] Anthony Barrett. From hybrid systems to universal plans via domain compilation. In *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 44–51, 2004.
- [Bayardo and Pehoushek, 2000] Roberto Bayardo and Joseph Pehoushek. Counting models using connected components. In *Proceedings of the 17th National Conference on Artificial Intelligence (AAAI)*, pages 157–162, 2000.
- [Blum *et al.*, 1980] Manuel Blum, Ashok K. Chandra, and Mark N. Wegman. Equivalence of free Boolean graphs can be decided probabilistically in polynomial time. *Information Processing Letters*, 10(2):80–82, 1980.
- [Bryant, 1986] R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35:677–691, 1986.
- [Cadoli and Donini, 1997] Marco Cadoli and Francesco M. Donini. A survey on knowledge compilation. *AI Communications*, 10:137–150, 1997.
- [Darwiche and Huang, 2002] Adnan Darwiche and Jinbo Huang. Testing equivalence probabilistically. Technical Report D-123, Computer Science Department, UCLA, 2002.
- [Darwiche and Marquis, 2002] Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *Journal of Artificial Intelligence Research*, 17:229–264, 2002.
- [Darwiche, 2001a] Adnan Darwiche. Decomposable negation normal form. *Journal of the ACM*, 48(4):608–647, 2001.
- [Darwiche, 2001b] Adnan Darwiche. On the tractability of counting theory models and its application to belief revision and truth maintenance. *Journal of Applied Non-Classical Logics*, 11(1-2):11–34, 2001.
- [Darwiche, 2002] Adnan Darwiche. A compiler for deterministic decomposable negation normal form. In *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI)*, pages 627–634, 2002.
- [Darwiche, 2004] Adnan Darwiche. New advances in compiling CNF into decomposable negation normal form. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI)*, pages 328–332, 2004.
- [Davis *et al.*, 1962] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem proving. *Journal of the ACM*, (5)7:394–397, 1962.
- [Dechter and Mateescu, 2004a] Rina Dechter and Robert Mateescu. The impact of AND/OR search spaces on constraint satisfaction and counting. In *Proceedings of the 10th International Conference on Principles and Practice of Constraint Programming (CP)*, pages 731–736, 2004.
- [Dechter and Mateescu, 2004b] Rina Dechter and Robert Mateescu. Mixtures of deterministic-probabilistic networks and their AND/OR search spaces. In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 120–129, 2004.
- [Gergov and Meinel, 1994] J. Gergov and C. Meinel. Efficient analysis and manipulation of OBDDs can be extended to FBDDs. *IEEE Transactions on Computers*, 43(10):1197–1209, 1994.
- [Hoos and Stützle, 2000] Holger H. Hoos and Thomas Stützle. SATLIB: An Online Resource for Research on SAT. In *I.P.Gent, H.v.Maaren, T.Walsh, editors, SAT 2000*, pages 283–292. IOS Press, 2000. SATLIB is available online at www.satlib.org.
- [Huang and Darwiche, 2004] Jinbo Huang and Adnan Darwiche. Using DPLL for efficient OBDD construction. In *Proceedings of the Seventh International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pages 127–136, May 2004.
- [Majercik and Littman, 1998] Stephen M. Majercik and Michael L. Littman. Using caching to solve larger probabilistic planning problems. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI)*, pages 954–959, 1998.
- [Marquis, 1995] Pierre Marquis. Knowledge compilation using theory prime implicates. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 837–843, 1995.
- [Moskewicz *et al.*, 2001] Matthew Moskewicz, Conor Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 39th Design Automation Conference*, pages 530–535, 2001.
- [Palacios *et al.*, 2005] Hector Palacios, Blai Bonet, Adnan Darwiche, and Hector Geffner. Pruning conformant plans by counting models on compiled d-DNNF representations. In *Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS)*, 2005.
- [Sang *et al.*, 2004] Tian Sang, Fahiem Bacchus, Paul Beame, Henry Kautz, and Toniann Pitassi. Combining component caching and clause learning for effective model counting. In *Proceedings of the Seventh International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pages 20–28, 2004.
- [Selman and Kautz, 1991] Bart Selman and Henry Kautz. Knowledge compilation using horn approximation. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI)*, pages 904–909, 1991.
- [Selman and Kautz, 1996] Bart Selman and Henry Kautz. Knowledge compilation and theory approximation. *Journal of the ACM*, 43(2):193–224, 1996.
- [Somenzi, Release 240] Fabio Somenzi. CUDD: CU Decision Diagram Package. Release 2.4.0.

Optimal Refutations for Constraint Satisfaction Problems

Tudor Hulubei and Barry O’Sullivan

Cork Constraint Computation Centre

Department of Computer Science, University College Cork, Ireland

tudor@hulubei.net, b.osullivan@cs.ucc.ie

Abstract

Variable ordering heuristics have long been an important component of constraint satisfaction search algorithms. In this paper we study the behaviour of standard variable ordering heuristics when searching an insoluble (sub)problem. We employ the notion of an optimal refutation of an insoluble (sub)problem and describe an algorithm for obtaining it. We propose a novel approach to empirically looking at problem hardness and typical-case complexity by comparing optimal refutations with those generated by standard search heuristics. It is clear from our analysis that the standard variable orderings used to solve CSPs behave very differently on real-world problems than on random problems of comparable size. Our work introduces a potentially useful tool for analysing the causes of the heavy-tailed phenomenon observed in the runtime distributions of backtrack search procedures.

1 Introduction

The conventional wisdom in systematic backtrack search is that a variable ordering heuristic should be chosen so that if a bad assignment is made, and search enters an insoluble (sub)tree, the heuristic will prefer variables that have a high likelihood of proving quickly that the current (sub)tree is indeed unproductive. This property of a variable ordering heuristic is often referred to as *fail-firstness* [Haralick and Elliott, 1980]. Fail-firstness can be seen as an explanation for why particular variable ordering heuristics are better than others for solving CSPs, and it has received considerable attention from the research community [Smith and Grant, 1998; Beck *et al.*, 2004]. Various ways of explaining the quality of variable ordering heuristics in terms of fail-firstness have been proposed. Some authors argue that heuristics that minimise branching depth are best [Haralick and Elliott, 1980], while others argue that such a policy can cause an increase in branching factor which can have the effect of increasing the size of the resultant (sub)tree [Smith and Grant, 1998].

A search algorithm enters an insoluble search tree either because the problem itself is insoluble, or because the current partial solution has just been extended with an assignment

that cannot lead to a complete solution. In either case, the ideal is that insolubility can be proven with the least amount of effort. In this paper we will measure such effort by the number of nodes visited to prove that an insoluble (sub)tree has no solution. We can evaluate variable ordering heuristics by comparing the number of nodes that they visit to prove the insolubility of a (sub)problem with the minimum number of nodes required to draw the same conclusion.

By comparing optimal refutations with those generated by MAC [Sabin and Freuder, 1994], when combined with various standard variable ordering heuristics, we show empirically that there is a significant difference between the performance of heuristics on typical random problems and those found in the real world. We also suggest that small-to-medium sized random binary CSPs are of limited value when studying and developing variable ordering heuristics. While this has been the intuition amongst constraints researchers, our analysis shows empirically that this is indeed the case and gains insights into the nature of the difference in behaviour between random and real-world problems. Finally, we discuss how our results may provide further insights into the causes of the heavy-tailed phenomenon observed in the runtime distributions of backtrack search procedures. For example, by focusing on the distributions of actual and optimal refutations we can study whether heavy-tailedness is present because of the variable ordering heuristic, the value ordering heuristic, or is more fundamentally related to the structure of the problem.

2 Preliminaries

Definition 1 (Constraint Satisfaction Problem). We define a CSP as a 3-tuple $P \triangleq \langle V, D, C \rangle$ where V is a finite set of n variables $V \triangleq \{V_1, \dots, V_n\}$, D is a set of finite domains $D \triangleq \{D(V_1), \dots, D(V_n)\}$ such that $D(V_i)$ is the finite set of possible values for V_i , and C is a finite set of constraints such that each $C_{ij} \in C$ is a subset of $D(V_i) \times D(V_j)$ specifying the combinations of values allowed between V_i and V_j , where $i < j$. We say that P is arc-consistent (AC) if $\forall v_k \in D(V_i)$ and $\forall j$ such that $C_{ij} \in C$, $\exists v_l \in D(V_j)$ with $(v_k, v_l) \in C_{ij}$. An assignment $A_{ik} \triangleq \langle V_i = v_k \rangle$ represents a reduction of $D(V_i)$ to $\{v_k\} \subseteq D(V_i)$. A solution to P is a set of distinct assignments $\mathcal{S} \triangleq \{A_{l_1 k_1}, \dots, A_{l_n k_n} \mid (v_{k_i}, v_{k_j}) \in C_{ij}\}$.

Definition 2 (Search Algorithm). A search algorithm $\Theta \triangleq \langle \Delta, \Delta, \prec_V, \prec_v \rangle$ is a combination of a branching method Δ , a

consistency enforcement method Δ , a variable ordering $\prec_V$ and a value ordering $\prec_v$, both of which can be either static or dynamic.

Definition 3 (Search Tree). A search tree $\mathcal{T}$ for a problem P is a set of nodes and arcs. Each node corresponds to a set of assignments, $\mathcal{N} \triangleq \{A_{l_1 k_1}, \dots, A_{l_{p-1} k_{p-1}}, A_{l_p k_p}\}$, totally ordered by a dynamic variable ordering heuristic $\prec_V$. The root of the search tree is a special node $\mathcal{R} \triangleq \emptyset$. Two nodes $\mathcal{N}_1$ and $\mathcal{N}_2$ are connected by an arc if $\exists A_{ij}$ such that $\mathcal{N}_2 = \mathcal{N}_1 \cup A_{ij}$, in which case we say that $\mathcal{N}_1$ is the parent of $\mathcal{N}_2$, and $\mathcal{N}_2$ is the child of $\mathcal{N}_1$. For every node $\mathcal{N}$, its children are totally ordered by a dynamic value ordering heuristic $\prec_v$.

Search trees are defined in the context of a specific search algorithm. For a particular CSP instance P , and search algorithm Θ , a one-to-one mapping exists between the nodes in the search tree $\mathcal{T}$ and the assignments made by Θ .

Definition 4 (Mistake Point). For a soluble problem P , a mistake point $\mathcal{M}$ is a node identified by a set of assignments $\mathcal{M} \triangleq \{A_{l_1 k_1}, \dots, A_{l_{p-1} k_{p-1}}, A_{l_p k_p}\}$, totally ordered by $\prec_V$, for which $\exists S \in \mathcal{S}$ such that $\mathcal{M} \setminus \{A_{l_p k_p}\} \subset S$, but $\neg \exists S \in \mathcal{S}$ such that $\mathcal{M} \subset S$. Since an insoluble problem does not admit any solutions, we define the mistake point associated with an insoluble problem as the root of its search tree.

Informally, a mistake point corresponds to an assignment that, given past assignments, cannot lead to a solution even though, in the case of a soluble problem, a solution exists. Whenever the value ordering heuristic makes such a mistake, the role of the variable ordering heuristic is to guide the search out of that insoluble search tree as quickly as possible. However, it is important to realise that the actual set of mistake points encountered during search is also dependent on the variable ordering heuristic used.

For a soluble problem P , let $P_{\mathcal{M}} \triangleq \{V_{\mathcal{M}}, D_{\mathcal{M}}, C_{\mathcal{M}}\}$ be the insoluble (sub)problem corresponding to $\mathcal{M}$, where $V_{\mathcal{M}} \triangleq V \setminus \{V_1, \dots, V_p\}$, $C_{\mathcal{M}} \triangleq \{C_{ij} | V_i, V_j \in V_{\mathcal{M}}, C_{ij} \in C\}$, and $D_{\mathcal{M}}$ is the set of current domains after arc-consistency has been restored to reflect the domain reductions due to $\mathcal{M}$. If P is insoluble, as a notational convenience, we define $\mathcal{M} \triangleq \emptyset$ and $P_{\mathcal{M}}$ as the arc-consistent version of P . For brevity, we will refer to the *insoluble (sub)tree* rooted at a mistake point and its corresponding *insoluble (sub)problem* interchangeably.

Definition 5 (Refutations). Given a search algorithm Θ , a *refutation* for a given insoluble (sub)problem $P_{\mathcal{M}}$, rooted at mistake point, $\mathcal{M}$, is simply the corresponding search tree $\mathcal{T}_{\mathcal{M}}$. We will refer to $|\mathcal{T}_{\mathcal{M}}|$, the number of nodes in $\mathcal{T}_{\mathcal{M}}$, as the size of the refutation.

We study the refutations found using a version of MAC that uses AC-3 [Mackworth, 1977] for consistency enforcement and selects values randomly. Our goal is to determine how close to optimality are the refutations obtained when well known variable ordering heuristics, with randomly broken ties, are substituted for $\prec_V$. For each heuristic $\prec_V$ we first collect the mistake points it generates when using MAC

(note that each variable ordering heuristic will generate a different set of mistake points). When we analyse MAC in conjunction with a certain $\prec_V$ on a mistake point $\mathcal{M}$, we will refer to the refutation for the (sub)problem $P_{\mathcal{M}}$ as the *actual refutation*. We will contrast the actual refutation with the *optimal refutation* for $P_{\mathcal{M}}$, obtained by replacing $\prec_V$ with a new variable ordering heuristic $\prec_V^*$ such that $|\mathcal{T}_{\mathcal{M}}|$ is minimised.

Sometimes, rather than using the optimal refutation, we compare against the *quasi-optimal*, defined as the smallest refutation whose depth does not exceed that of the actual. By selecting variables based on a depth-first traversal of the tree of minimum size, $\prec_V^*$ causes MAC to generate the smallest possible search tree proving insolubility for $P_{\mathcal{M}}$. In our experience it is extremely rare that the quasi-optimal refutation is larger than the optimal. By accepting quasi-optimality we can use the depth of the actual refutation as an upper bound, dramatically speeding-up search for better refutations.

Definition 6 (Improvement Ratios). For a given mistake point, the *refutation improvement ratio* is the ratio between the size of the actual refutation and that of the optimal refutation. The *overall refutation improvement ratio* is simply the ratio between the total number of nodes in all the actual refutations and the total number of nodes in all their corresponding optimal refutations encountered when finding the first solution or proving insolubility.

3 Analysing Refutations

Our focus here is only on how a variable ordering heuristic behaves when trying to *prove insolubility* for a (sub)problem. We are not interested in focusing on *how often* a variable ordering heuristic enters an insoluble (sub)tree. Therefore, our analysis is centered around finding optimal refutations for all the mistake points in a search tree. Unfortunately, this is a herculean task, as we have to search through all the possible permutations of assignments, i.e. consider all possible search trees rooted at each mistake point. Without clever optimisations, this would not be feasible for anything but the tiniest problems.

3.1 Basic Algorithm

Given a problem instance, a search algorithm (MAC in our case) and the variable ordering heuristic to be evaluated, we begin by searching for a solution, with the goal of identifying all the mistake points. The mistake points represent the roots of the insoluble (sub)trees the search algorithm happens to encounter before finding the first solution. For each such mistake point we compute both the actual refutation, by re-running MAC on the current (sub)problem using the heuristic we are evaluating, and the optimal refutation, by considering all the possible permutations of assignments to the variables involved in that (sub)problem. Note that insoluble instances are just a special case where the only mistake point is before the first assignment. Furthermore, when analysing insoluble problems, it is important to realise that the optimal refutation is independent of the heuristic used; in other words *no other ordering* could possibly prove insolubility more quickly.

Both the search algorithm with the variable ordering heuristics being analysed and the exhaustive search for the

optimal refutation use the same method of enforcing consistency (AC in our case). Failure to do so would make the refutation improvement ratio a function of both the variable ordering and the level of consistency enforced, thus being detrimental to our goal of analysing variable ordering heuristics.

The version of MAC that we use employs k-way branching [Smith and Sturdy, 2004], rather than binary branching, so that selecting a variable V_i creates $|D(V_i)|$ branches in the search tree. Thus, we make sure that refutation sizes are not influenced by the value ordering.

3.2 Optimisations

There are a number of optimisations that we have applied.

Firstly, we can easily maintain an upper bound on the size of the optimal refutation by initially setting it to the size of the actual refutation, then updating it as smaller refutations are found. Therefore, the search for the optimal refutation can be implemented as a branch-and-bound procedure.

Secondly, we can improve the search by looking ahead to avoid choices that cannot improve on the current smallest refutation. Whenever a variable V_i is selected at a certain depth all the values in its domain have to be tried, and they all have to fail for the current (sub)problem to be proved insoluble. Consequently, we know that by selecting V_i , the size of the current partial refutation will increase by at least a number of nodes equal to $|D(V_i)|$. We call this a 1-level look-ahead.

By temporarily assigning to V_i , in turn, every value v in its domain, and by attempting to restore arc-consistency after every such assignment, we can associate with each v a minimum contribution to the size of the refutation. If the assignment makes the subproblem arc-inconsistent, v 's contribution will be 1, given by the node corresponding to the assignment itself. However, if arc-consistency can be restored after the assignment, at least one more variable will have to be considered before the current subproblem can be proved insoluble. Therefore, v will carry a minimum contribution equal to the smallest domain size amongst all the remaining unassigned variables. We call this a 2-level look-ahead.

In general, V_i 's selection would increase the size of the current partial refutation by at least the sum of the minimum contributions of all the values in its domain. If that would cause the current partial refutation to exceed the size of the smallest refutation found so far, V_i will be skipped at the current depth. This dramatically reduces the search space and brings reasonable size problems within reach.

4 Experiments

The basic experiment we run is this: for each variable ordering heuristic we first find all the mistake points that MAC visits when finding the first solution or proving insolubility; for each mistake point, we find MAC's actual refutation (still with the same heuristic); using the size of the actual refutation as an upper bound, we find the (quasi-)optimal refutation. This approach was adopted because different variable orderings generate potentially different sets of mistake points.

We have performed experiments¹ on random binary problems, various instances of the n -queens problem

¹ Code freely available with source at <http://hulubei.net/tudor/csp>.

for varying values of n , and a subset of the RL-FAP Celar Scen11 [Cabon *et al.*, 1999]. We have studied the following dynamic variable ordering heuristics: min-domain [Haralick and Elliott, 1980], max-degree, min(domain/degree) [Bessière and Regin, 1996], and bre-laz [Brélaz, 1979]; their corresponding anti-heuristics; and random variable selection. These are regarded as the standard general-purpose variable ordering heuristics used in the CSP literature. Ties were broken randomly. Throughout our experiments we have used a dynamic random value ordering.

Note that for heuristics we report the optimal refutations for random problems and n -queens, and the *quasi*-optimal refutations for RL-FAP. For anti-heuristics we always compute *quasi*-optimal refutations. While the true optimal refutations for RL-FAP may be smaller than those reported, based on our observations the difference is not significant. Limiting the search to quasi-optimal proofs allowed us to gather significantly more data for this problem.

4.1 Random and n -Queens Problems

Due to the extreme computational requirements of the search for optimal refutations, we had to limit our experiments to random problems with 15 variables and uniform domain sizes of 10 values. We generated 800 instances for each tightness point using a random problem generator that conforms to the Model-B specification [Gent *et al.*, 2001].

We selected instances at the highest peak of difficulty for these problems, found to pass through a plane corresponding to density 0.95. Tightness was varied along this plane. Specifically, we identify a distinct tightness at which we move from generating a set of problem instances that are all soluble to generating instances that are all insoluble. In Figure 1 this transition occurs at a tightness of 0.3. We indicate the range of tightness for which all problems were soluble (resp. insoluble) with a "high" (resp. "low") line.

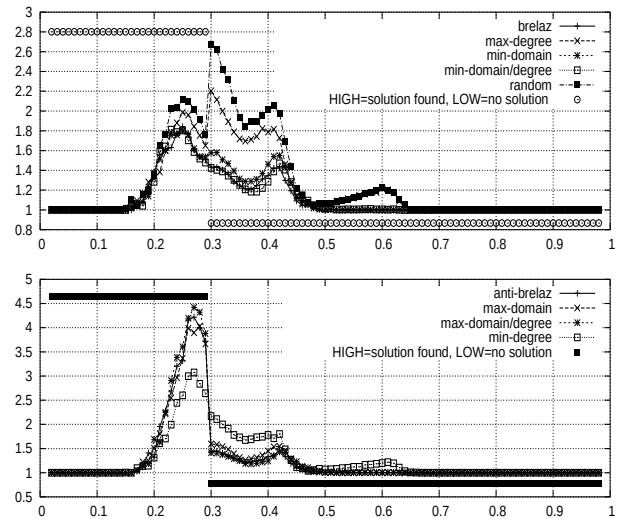


Fig. 1: Mean overall improvement ratios (y-axis) for random problems as a function of tightness (x-axis). Heuristics (anti-heuristics) are in the top (bottom) plot. Median plots are virtually identical.

In Figure 1 we present the results associated with each of the variable ordering heuristics studied. Note that we are not ranking heuristics, but are simply studying how close their actual refutation sizes are to optimality. The overall refutation improvement ratio peaks at 1.8 for min(domain/degree) at a tightness of about 0.24, meaning that for the set of mistake points encountered when solving these problems using min(domain/degree) there does not exist a variable ordering that could decrease the number of nodes in the corresponding refutations by more than that factor, on the average. Since at the peak of difficulty proving insolubility accounts for most of the time spent in search, this result suggests that no improvement in the ordering of variables, for the problems we have studied, would ever lead to an order of magnitude speedup. Indeed, in the soluble region we can see that at best we can hope to halve the time taken by max-degree. In theory, however, for soluble problems heuristics could do better by failing less.

Once we move through the phase-transition to insoluble problems, all heuristics offer improvement ratios less than 2.2 for max-degree and less than 1.6 for all other heuristics, except random. In the case of min(domain/degree) we see that when using MAC at the peak of difficulty the ratio is approximately 1.4. This is an interesting result since it demonstrates that for proving insolubility *no variable ordering heuristic exists*, for these problems, that can improve upon min(domain/degree) by more than a factor of 1.4 at the peak of difficulty.

Figure 2 presents a plot of the distribution of actual refutation sizes against optimal, per mistake point, for the random and min(domain/degree) variable ordering heuristics. Of interest here is the apparent *linear relationship* between actual versus optimal refutation size. A similar linear relationship was observed for all other heuristics and anti-heuristics studied. When we discuss the performance of variable ordering heuristics on a real-world problem in the next section we will see that this linear relationship does not hold and that for very large refutations found by a heuristic, the optimal refutation can be many orders of magnitude smaller.

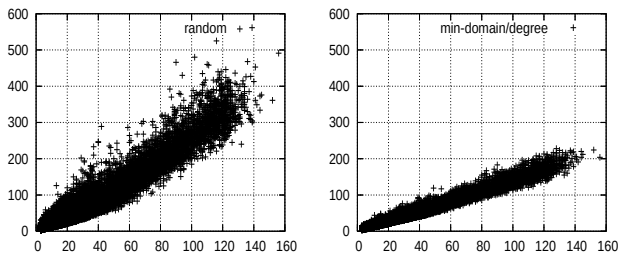


Fig. 2: Refutation sizes for mistake points in random problems when using random and min(domain/degree) variable orderings: actual (y-axis) as a function of optimal (x-axis).

The performance of a random variable ordering is of comparable quality to that of a good variable ordering heuristic in this problem class. From Figure 1 we see that the improvement ratio of the random heuristic is approximately 2.7 at the phase transition, close to that of standard heuristics.

Finally, in Figure 3 we present, for each variable ordering heuristic, the distribution of insoluble (sub)trees in the data-set as a function of the improvement ratio. We see that almost all ratios were close to 1, meaning that almost all of the time the standard variable ordering heuristics performed very close to perfection. We also see that for the random ordering heuristic ratios of 10 are found less than 0.01% of the time and that there does not exist a single insoluble (sub)tree that provides an opportunity to make improvements of more than an order of magnitude over current heuristics.

In experiments performed over a large data-set of insoluble subtrees generated from problems ranging from 8 to 22 queens we have observed similarly small *refutation improvement ratios* for n -queens, only slightly bigger than those observed for random problems. Also, the linear relationship between actual and optimal refutation size was observed, as well as a similar distribution of ratios.

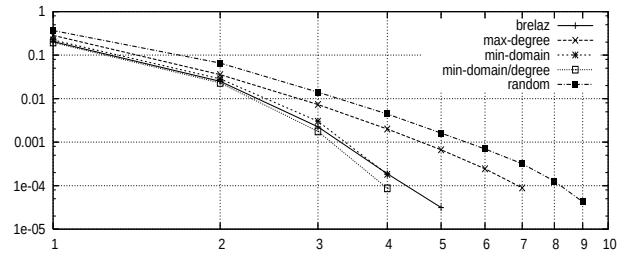


Fig. 3: Probability of an insoluble (sub)tree in the data-set for random problems (y-axis) having an improvement ratio greater than a certain value (x-axis).

4.2 RLFAP Celar Scen11

The small improvement ratios obtained for random problems and n -queens, and estimated in [Li and Gérard, 2000] for hard random unsatisfiable 3-SAT, do not seem to hold for real-world problems. A popular benchmark for binary CSPs, RLFAP Celar Scen11 was an obvious candidate for our analysis. Unfortunately, at 680 variables, we could not analyse the entire problem. Instead, we extracted a subset of the original problem made up of 22 variables that were part of the backdoor set [Williams *et al.*, 2003] over which MAC backtracked while solving the original problem. We analysed all the mistake points found when solving this subset of RLFAP over 500 times for all variable ordering heuristics.

Refutation improvement ratios of 20 and over were observed, even when using the best heuristics, as seen in Figures 4 and 5, while they were non-existent for random and n -queens problems. Comparing Figures 2 and 4 we can clearly see that heuristics behaves very differently on this problem than on the random problems presented earlier. Specifically, we can see that actual refutations correlate very poorly with quasi-optimal and that no linear relationship exists; note that a log-scale had to be used on the y-axis of the left plot in Figure 4 due to the spread of actual refutation sizes for each quasi-optimal point on the x-axis.

However, of particular interest in this problem is the pattern we observe in Figure 5. While the vast majority of refutations

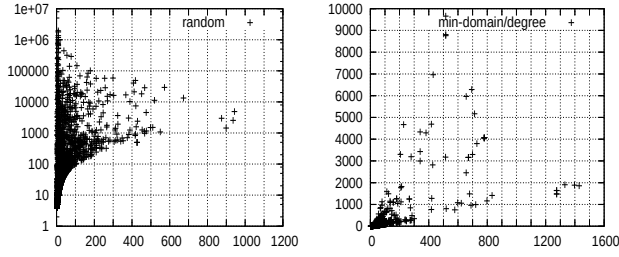


Fig. 4: Refutation sizes for mistake points in the RLFAP data-set when using random and min(domain/degree) variable orderings: actual (y-axis) as a function of quasi-optimal (x-axis).

offer small improvement ratios, it is possible to find subtrees on which the standard heuristics perform very poorly. For example, we can see that in a small number of subtrees max-degree did 100 times worse than quasi-optimal, while the other heuristics, except min(domain/degree), sometimes did 50 times worse than quasi-optimal. Clearly, the behaviour of standard variable orderings is significantly different on real-world problems than on the random problems that many researchers use as a basis for studying these heuristics. In the case of a random variable ordering, we observed a very small number of subtrees where the refutation found was more than a factor of 10^5 worse than quasi-optimal, and that in the vast majority of cases it performed more than an order of magnitude worse than quasi-optimal.

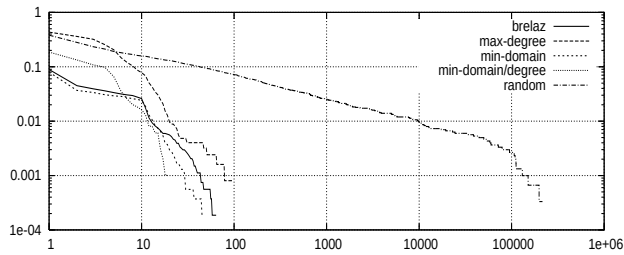


Fig. 5: Probability of an insoluble subtree in the RLFAP data-set (y-axis) having an improvement ratio greater than a certain value (x-axis).

Therefore, our standard heuristics behave very differently, indeed much worse, on real-world problems than on random and n -queens problems. In particular, they can be less robust in the sense that they can sometime waste significant amounts of time finding refutations for insoluble (sub)trees that are far worse than optimal.

5 Discussion

It is clear from our analysis that the standard variable orderings used to solve CSPs behave very differently on real-world problems than on random problems of comparable size, which are frequently used to study the performance of heuristics in the research literature. Our results demonstrate that there are considerable opportunities to improve search performance on real-world problems. Some researchers

have already begun developing the necessary infrastructure needed to study real-world problems in a systematic way [Gomes *et al.*, 2000; Walsh, 1999; Walsh, 2001]. There have also been some recent successful results demonstrating that one can benefit considerably from developing variable ordering heuristics that can exploit characteristics one encounters in real-world problems [Boussemart *et al.*, 2004; Dubois and Dequen, 2001; Refalo, 2004].

It is not known whether the linear relationship between actual and optimal refutations is preserved as the size of random problems increases. Recent work on studying the performance of variable ordering heuristics on very large CSPs has shown that there can be orders of magnitude differences between heuristics [Bessière *et al.*, 2001]. However, those results do not contradict the results presented in this paper with respect to the relative performance of standard variable orderings on random and non-random problems.

Our analysis is related to the work of Li and Gerard on studying the limits of branching heuristics for random 3-SAT [Li and Gérard, 2000]. However, our interests here are in studying the limits of variable ordering heuristics in a constraint satisfaction context. Furthermore, our analysis is capable of finding exact optimal refutations, rather than approximating them, and focuses on the differences observed between random and real-world problems. Our work is also closely related to the theoretical analysis of proof complexity for satisfiability and constraint satisfaction [Beame *et al.*, 1998; Mitchell, 2003].

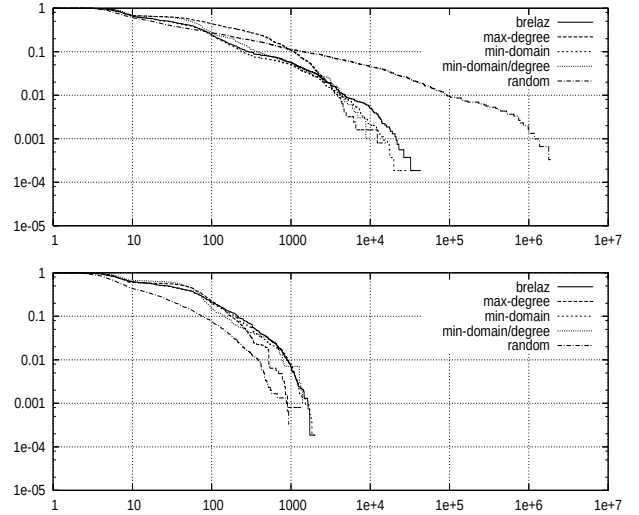


Fig. 6: Probability of an insoluble subtree in the RLFAP data-set (y-axis) being greater than an optimal/actual (top/bottom) refutation of a given size (x-axis).

In this paper we introduce a potentially useful tool for analysing the causes of the heavy-tailed phenomenon observed in the runtime distributions of backtrack search procedures [Gomes *et al.*, 2000]. Figures 5 and 6 show, for RLFAP, a very small percentage of insoluble (sub)problems for which standard variable ordering heuristics will find refutations that are significantly larger than optimal, leading to a

long runtime. Furthermore, it is also interesting to note from Figure 5 that the vast majority of actual refutations are very close to being optimal, with the exception of those found with a random variable ordering heuristic.

Gomes et al. [2004] report a high correlation between the distribution of the depth of mistake points and the runtime distribution, e.g. the presence of heavy-tails. We can further extend this analysis by also considering the size of optimal refutations rooted at these mistake points. This may provide a way of isolating the causes of heavy-tailed behaviour by making a distinction between the effects of the algorithm and its components (propagation method, variable and value orderings), versus the effects of the inherent structure of the problem [Walsh, 1999; Walsh, 2001]. As part of our future work in this area we will study optimal refutations for problems that are well known to exhibit heavy-tail behaviour.

6 Conclusions

In this paper we have proposed a novel approach to empirically looking at problem hardness and typical-case complexity by comparing optimal refutations with those generated by standard search heuristics. We have compared refutations of insoluble (sub)trees found using well known variable ordering heuristics combined with a standard CSP search algorithm against optimal refutations. Our results show that there is a significant difference between the performance of heuristics on typical random problems and those found in the real-world. We suggest that small-to-medium sized random binary CSPs are of limited value to constraints researchers and introduced a potentially useful tool for analysing the causes of the heavy-tailed phenomenon observed in the runtime distributions of backtrack search procedures.

Acknowledgments

This material is based on work supported by Science Foundation Ireland under Grant 00/PI.1/C075. We would like to thank Barbara M. Smith, Eugene C. Freuder, Nic Wilson, Chris Beck, Bart Selman, Carla Gomes and Susan Epstein for their comments and suggestions, and to John Morrison and the Boole Centre For Research in Informatics for providing access to their Beowulf cluster.

References

- [Beame et al., 1998] P. Beame, R. Karp, T. Pitassi, and M. Saks. On the complexity of unsatisfiability of random k -cnf formulas. In *Proceedings of ACM STOC-1998*, pages 561–571, 1998.
- [Beck et al., 2004] J.C. Beck, P. Prosser, and R.J. Wallace. Trying again to fail-first. In *Proceedings of CSCLP-2004*, LNCS 3419, pages 41–55, 2004.
- [Bessière and Regin, 1996] C. Bessière and J-C. Regin. MAC and combined heuristics: two reasons to forsake FC (and CBJ?) on hard problems. In *Proceedings of CP-1996*, LNCS 1118, pages 61–75, 1996.
- [Bessière et al., 2001] C. Bessière, A. Chmeiss, and L. Sais. Neighbourhood-based variable ordering heuristics for the constraint satisfaction problem (long version). Available from: <http://www.lirmm.fr/~bessiere/pubs>, 2001.
- [Boussemart et al., 2004] F. Boussemart, F. Hemery, C. Lecoutre, and L. Sais. Boosting systematic search by weighting constraints. In *Proceedings of ECAI-2004*, pages 146–150, 2004.
- [Brélaz, 1979] D. Brélaz. New methods to color the vertices of a graph. *Communications of the ACM*, 22(4):251–256, 1979.
- [Cabon et al., 1999] B. Cabon, S. Givry, L. Lobjois, T. Schiex, and J.P. Warners. Radio link frequency assignment. *Constraints*, 4(1):79–89, 1999.
- [Dubois and Dequen, 2001] O. Dubois and G. Dequen. A backbone-search heuristic for efficient solving of hard 3-SAT formulae. In *Proceedings of IJCAI-2001*, pages 248–253, 2001.
- [Gent et al., 2001] I.P. Gent, E. MacIntyre, P. Prosser, B.M. Smith, and T. Walsh. Random constraint satisfaction: Flaws and structure. *Constraints*, 6(4):345–372, 2001.
- [Gomes et al., 2000] C.P. Gomes, B. Selman, N. Crato, and H. Kautz. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *Automated Reasoning*, 24(1/2):67–100, 2000.
- [Gomes et al., 2004] C.P. Gomes, C. Fernández, B. Selman, and C. Bessière. Statistical regimes across constrainedness regions. In *Proceedings of CP-2004*, LNCS 3258, pages 32–46, 2004.
- [Haralick and Elliott, 1980] R.M. Haralick and G.L. Elliott. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14(3):263–313, 1980.
- [Li and Gérard, 2000] C.M. Li and S. Gérard. On the limit of branching rules for hard random unsatisfiable 3-sat. In *Proceedings of ECAI-2000*, pages 98–102, 2000.
- [Mackworth, 1977] A.K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8(1):99–118, 1977.
- [Mitchell, 2003] D.G. Mitchell. Resolution and constraint satisfaction. In *Proceedings of CP-2003*, LNCS 2833, pages 555–569, 2003.
- [Refalo, 2004] P. Refalo. Impact-based search strategies for constraint programming. In *Proceedings of CP-2004*, LNCS 3258, pages 557–571, 2004.
- [Sabin and Freuder, 1994] D. Sabin and E.C. Freuder. Contradicting conventional wisdom in constraint satisfaction. In *Proceedings of ECAI-1994*, pages 125–129, 1994.
- [Smith and Grant, 1998] B.M. Smith and S.A. Grant. Trying harder to fail first. In *Proceedings of ECAI-1998*, pages 249–253, 1998.
- [Smith and Sturdy, 2004] B.M. Smith and P. Sturdy. An empirical investigation of value ordering for finding all solutions. In *Workshop on Modelling and Solving Problems with Constraints*, 2004.
- [Walsh, 1999] T. Walsh. Search in a small world. In *Proceedings of IJCAI-1999*, pages 1172–1177, 1999.
- [Walsh, 2001] T. Walsh. Search on high degree graphs. In *Proceedings of IJCAI-2001*, pages 266–271, 2001.
- [Williams et al., 2003] R. Williams, C.P. Gomes, and B. Selman. Backdoors to typical case complexity. In *Proceedings of IJCAI-2003*, pages 1173–1178, 2003.

Efficient Stochastic Local Search for MPE Solving

Frank Hutter

Computer Science Dept.
Univ. of British Columbia
Vancouver, BC, Canada
hutter@cs.ubc.ca

Holger H. Hoos

Computer Science Dept.
Univ. of British Columbia
Vancouver, BC, Canada
hoos@cs.ubc.ca

Thomas Stützle

Computer Science Dept.
Darmstadt Univ. of Technology
Darmstadt, Germany
stuetzle@informatik.tu-darmstadt.de

Abstract

Finding most probable explanations (MPEs) in graphical models, such as Bayesian belief networks, is a fundamental problem in reasoning under uncertainty, and much effort has been spent on developing effective algorithms for this $\mathcal{NP}$ -hard problem. Stochastic local search (SLS) approaches to MPE solving have previously been explored, but were found to be not competitive with state-of-the-art branch & bound methods. In this work, we identify the shortcomings of earlier SLS algorithms for the MPE problem and demonstrate how these can be overcome, leading to an SLS algorithm that substantially improves the state-of-the-art in solving hard networks with many variables, large domain sizes, high degree, and, most importantly, networks with high induced width.

1 Introduction

Since Pearl’s classic text [Pearl, 1988], graphical models such as Bayesian networks have become the prime representation for uncertainty in AI. This paper deals with the problem of finding the *Most Probable Explanation (MPE)* for some evidence when reasoning under uncertainty. More specifically, in the light of uncertain knowledge represented as a probabilistic graphical model, this problem is cast as finding the most probable instantiation of all the model’s variables $\mathbf{V}$ given the observed values $\mathbf{e}$ for a subset $\mathbf{E} \subseteq \mathbf{V}$.

The MPE problem in graphical models has applications in different fields, such as medical diagnosis [Jaakkola and Jordan, 1999], fault diagnosis [Rish *et al.*, 2002], computer vision [Tappen and Freeman, 2003], and prediction of side-chains in protein folding [Yanover and Weiss, 2003], to name just a few. Consequently, many algorithms have been suggested to solve this problem, but since it is $\mathcal{NP}$ -hard, many hard problem instances still cannot be solved efficiently. The available algorithms for MPE solving include exact methods like variable elimination (VE) [Dechter, 1999], junction tree (JT) [Cowell *et al.*, 1999] and conditioning techniques [Pearl, 1988], but also systematic search algorithms, such as branch & bound (B&B), guided by a Mini-Buckets (MB) heuristic [Dechter and Rish, 2003]. Since in practice many applications require efficient online algorithms for networks of high induced width, there is also much research

in approximate MPE algorithms, reaching from loopy belief propagation (BP) [Pearl, 1988] and generalized BP [Yedidia *et al.*, 2002] to stochastic local search (SLS) algorithms [Kask and Dechter, 1999; Park, 2002] and specialized algorithms, such as graph cuts for certain pairwise Markov Random Fields (MRFs) that, for example, occur in computer vision [Boykov *et al.*, 2001].

B&B approaches have recently been shown to be state-of-the-art methods in MPE solving, and it has been claimed that for MPE, B&B algorithms clearly outperform GLS, the best performing SLS algorithm known so far, in terms of the ability to find high-quality solutions quickly [Marinescu *et al.*, 2003]. This is in stark contrast to results for numerous other combinatorial optimisation problems, such as weighted MAX-SAT, where SLS algorithms clearly define the state-of-the-art (see, e.g., [Hoos and Stützle, 2004]).

In this work, we analyse the shortcomings of previous SLS algorithms for MPE and demonstrate that these weaknesses can be overcome based on the careful consideration of important issues, such as the time complexity of individual search steps, search stagnation and thorough parameter tuning. In particular, we introduce improvements to Park’s GLS algorithm [Park, 2002] that overcome its inferior scaling behaviour with network and domain size. Our new algorithm, GLS⁺, clearly outperforms current state-of-the-art MPE algorithms on various types of networks, especially for hard networks with high induced width, and hence establishes stochastic local search as a highly attractive and competitive approach to MPE solving.

The remainder of this paper is structured as follows. We first introduce some basic concepts and notation in Section 2; next, in Section 3, we describe GLS and GLS⁺ and present some computational results illustrating the improvement of GLS⁺ over GLS. In Section 4 we present empirical results that establish GLS⁺ as a new state-of-the-art algorithm for finding MPEs. We close with some conclusions and a brief outlook on future work in Section 5.

2 Preliminaries

A discrete *Bayesian belief network* (or *Bayes net*) $\mathcal{B}$ is a quadruple $(\mathbf{V}, \mathbf{D}, \mathcal{G}, \Phi)$, where $\mathbf{V}$ is an ordered set of random variables, $\mathbf{D}$ is an ordered set of finite domains D_{V_i} for each $V_i \in \mathbf{V}$, $\mathcal{G} = (\mathbf{V}, \mathcal{E})$ is a directed acyclic graph (DAG), and Φ is an ordered set of CPTs $\phi_V = P(V|pa(V))$, specifying the conditional probability distribution of each $V \in \mathbf{V}$.

given its parents $pa(V)$ in $\mathcal{G}$. Semantically, a Bayes net specifies a joint probability distribution ϕ over its variables $\mathbf{V}$ in factored form: $\phi = P(\mathbf{V}) = \prod_{V \in \mathbf{V}} \phi_V$.

Given a Bayes net $\mathcal{B} = \langle \mathbf{V}, \mathbf{D}, \mathcal{G}, \Phi \rangle$ and a set of evidence variables $\mathbf{E} = \mathbf{e}$, the *Most Probable Explanation (MPE)* problem is to find an instantiation $\mathbf{V} = \mathbf{v}$ with maximal probability $p(\mathbf{v}) := \prod_{\phi \in \Phi} \phi[\mathbf{V} = \mathbf{v}]$ over all variable instantiations $\mathbf{v}$ consistent with evidence $\mathbf{e}$. While all networks in our experimental analysis are Bayes nets, our algorithms are equally applicable to other graphical models, such as MRFs or general factor graphs.

CPTs are a special case of *potentials*, which are functions that have non-negative entries for any assignment of their variables. One well-known method for solving MPE in general networks is variable elimination (VE) (see, e.g., [Dechter, 1999]); it iteratively eliminates variables V by multiplying all potentials that are defined over V and then maximizing V out of the product thus obtained. Once all variables are eliminated, the best assignment can be recovered in linear time. Mini-Buckets with i -bound ib (MB(ib)) [Dechter and Rish, 2003] approximates VE by splitting each product into smaller products with at most ib variables; MB-w(s) is a new variant of MB that instead limits the number of entries in each product by s . For constant domain size d , MB(ib) and MB-w(d^{ib}) are equivalent, but for networks with different domain sizes MB-w performed better in our experiments. All of VE, MB, and MB-w employ a min-weight heuristic.

3 SLS for MPE: From GLS to GLS⁺

Probably the most prominent Stochastic Local Search algorithm for inference in Bayesian networks is a method called *Stochastic Simulation* [Pearl, 1988; Kask and Dechter, 1999], also known as Gibbs sampling. However, for MPE, this method, as well as Simulated Annealing, was shown to be clearly outperformed by a simple algorithm called *Greedy + Stochastic Simulation (G+StS)* [Kask and Dechter, 1999], which probabilistically chooses between greedy and sampling steps.

The MPE problem is closely related to weighted MAX-SAT [Park, 2002] (and MAX-CSP [Marinescu and Dechter, 2004]); based on this close relationship, Park adapted two high-performance MAX-SAT algorithms, DLM and GLS [Mills and Tsang, 2000], to the MPE problem [Park, 2002].¹ His computational experiments identified GLS as the state-of-the-art SLS algorithm for MPE solving: he showed that GLS and DLM clearly outperform G+StS and that GLS performed better than DLM on most instances. However, as shown in [Marinescu *et al.*, 2003], GLS does not reach the performance of current B&B algorithms.

GLS has been applied successfully to many combinatorial problems, including TSP [Voudouris and Tsang, 1999], SAT and weighted MAX-SAT [Mills and Tsang, 2000]. It can be classified as a dynamic local search algorithm [Hoos and Stützle, 2004] and uses penalties associated with solution components to guide the search process; these penalties

¹Although Park motivated his versions of these algorithms with the MAX-SAT domain, his implementation works on MPE directly. To our best knowledge, his suggested encoding has never been implemented, and we are not aware of any implementation of GLS for weighted MAX-SAT with real-valued weights.

Algorithm 1: GLS/GLS⁺ for MPE

GLS and GLS⁺ differ in the procedure used to generate an initial solution, the subsidiary local search procedure, and, most importantly, in the evaluation function $g(\mathbf{v}|V_i = v_i)$. These differences are explained in the text.

The utilities $util(\phi, \mathbf{v})$ are defined in the text.

Input: Bayes net $\mathcal{B} = \langle \mathbf{V}, \mathbf{D}, \mathcal{G}, \Phi \rangle$, evidence $\mathbf{E} = \mathbf{e}$, time bound t , smoothing factor ρ , smoothing interval N_ρ .
Output: Variable assignment $\mathbf{V} = \mathbf{v}$ with highest probability $\prod_{\phi \in \Phi} \phi[\mathbf{V} = \mathbf{v}]$ found in time t

```

//== Initialize variable assignment  $\mathbf{v}$ , penalties  $\lambda_\phi$ , and local optima counter  $LO$ .
1  $\mathbf{v} \leftarrow \text{GenerateInitialSolution}(\mathcal{B}, \mathbf{e})$ 
2 foreach  $\phi \in \Phi$  and all instantiations  $V_\phi = v_\phi$  do
3    $\lambda_\phi[V_\phi = v_\phi] \leftarrow 0$ 
4  $\#(LO) \leftarrow 0$ 
//== Alternate local search and updates of the evaluation function until termination.
5 while  $\text{runtime} < t$  do
6    $\mathbf{v} \leftarrow \text{SubsidiaryLocalSearch}(\mathcal{B}, \mathbf{v}, \mathbf{e})$ 
//== Local optimum, update evaluation function.
7   foreach  $\phi \in \Phi$  do
8     if  $util(\phi, \mathbf{v}) = \max_{\phi \in \Phi} util(\phi, \mathbf{v})$  then
9        $\lambda_\phi[\mathbf{V} = \mathbf{v}] \leftarrow \lambda_\phi[\mathbf{V} = \mathbf{v}] + 1$ 
//== Regularly smooth penalties.
10   $\#(LO) \leftarrow \#(LO) + 1$ 
11  if  $\#(LO) \bmod N_\rho = 0$  then
12    for  $\phi \in \Phi$  and all instantiations  $V_\phi = v_\phi$  do
13       $\lambda_\phi[V_\phi = v_\phi] \leftarrow \lambda_\phi[V_\phi = v_\phi] * \rho$ 

```

are dynamically updated whenever the search reaches a local optimum.

An outline of the GLS algorithm is shown in Figure 1. At a high level, after initialising the search and setting all penalties to zero, GLS alternates between two search phases: First, a simple iterative improvement search is performed with respect to the evaluation function g that takes into account the current penalty values. After this process has reached a local optimum o of g , certain penalty values are incremented by one. Only penalties of solution components present in o can be selected to be incremented; this selection is based on the contribution of the respective solution component to $g(o)$ and its current penalty value. (For details, see Algorithm 1 or [Park, 2002].) Additionally, all penalties are regularly multiplied by a factor $\rho \leq 1$; this smoothing mechanism prevents the penalty values from growing too large and is performed every N_ρ local optima, where N_ρ is a parameter.

In Park's GLS for MPE, the solution components are partial instantiations of the variables. More specifically, for each potential $\phi \in \Phi$, every instantiation $V_\phi = v_\phi$ of its variables is a solution component.² The evaluation function to be

²Note that the number of components present in any candidate solution for an MPE instance is constant; this is because every variable instantiation $\mathbf{V} = \mathbf{v}$ is consistent with exactly one (partial) instantiation $V_\phi = v_\phi$ of each potential ϕ , and thus the number of solution features present in each instantiation is the total number of potentials $|\Phi|$.

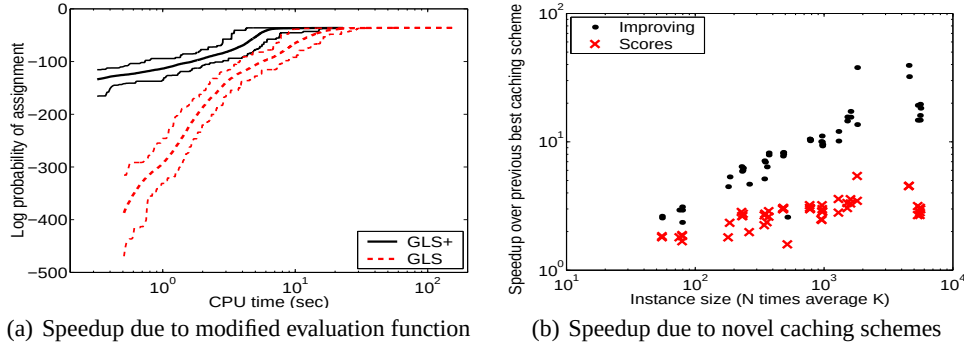


Figure 1: (a) Effect of the modification of the evaluation function in GLS⁺ compared to GLS on instance *Munin2*. Given are the best, average and worst solution quality over computation time for 25 runs of GLS and GLS⁺. The GLS⁺ plot ends after 22.94 CPU seconds when all its runs have found the optimal solution quality. (b) The speedup achieved by our new caching schemes over the previously best-performing caching scheme on a collection of randomly generated and real-world instances, where instance size is given as the number of variables (N) times the average domain size (K).

minimized is defined as $g(\mathbf{v}) = \sum_{\phi \in \Phi} \lambda_{\phi}[\mathbf{V} = \mathbf{v}]$, where $\lambda_{\phi}[\mathbf{V} = \mathbf{v}]$ is the penalty associated with solution component $V_{\phi} = v_{\phi}$. It may be noted that this deviates from the standard form of the evaluation function for the general GLS algorithm, which also captures the contribution of each solution component to the optimisation objective $p(\mathbf{v})$.

Due to this non-standard evaluation function, the sole interaction of objective function and penalties in Park’s GLS for MPE is via the utilities of potentials ϕ under the current assignment $\mathbf{v}$, which are defined as $util(\phi, \mathbf{v}) = -\phi[\mathbf{V} = \mathbf{v}]/(1 + \lambda_{\phi}[\mathbf{V} = \mathbf{v}])$. An entry $\phi[V_{\phi} = v_{\phi}]$ with high probability is assigned low utility, and its associated penalty $\lambda_{\phi}[V_{\phi} = v_{\phi}]$ will be increased less often, driving the search towards the partial variable instantiation $V_{\phi} = v_{\phi}$ eventually, but possibly only after a considerable delay.

Due to this initial and persistent lack of greediness, we expected the performance of GLS for MPE to be boosted significantly by integrating the objective function into the search heuristic. We achieved this by a change in GLS’s evaluation function. Our improved version of GLS, which we call GLS⁺, adds the logarithmic objective function to the appropriate penalties, making the new evaluation function to be maximized $g(\mathbf{v}) = \sum_{\phi \in \Phi} \log(\phi[\mathbf{V} = \mathbf{v}]) - w \times \lambda_{\phi}[\mathbf{V} = \mathbf{v}]$, where w is a weighting factor.³ Indeed, this modification of the evaluation function has major consequences on search behaviour and especially boosts the search in early phases of the search. Figure 1(a) illustrates this for the real-world network *Munin2* (from the Bayesian Network Repository), where GLS⁺ finds optimal-quality solutions 10 times faster on average.

GLS⁺ differs from GLS in a number of other components, namely the parameter setting, the caching scheme, and the initialization. All of these modifications contribute significantly to GLS⁺’s improved performance (a detailed analysis of their individual importance can be found in [Hutter, 2004]). Firstly, a thorough experimental analysis showed that, although Park states that “GLS had no parameters to

tune.” [Park, 2002], its performance can be boosted by up to several orders of magnitude by simply changing Park’s default smoothing value $\rho = 0.8$ to the constant value $\rho = 0.999$. For example, for the *Hailfinder* network GLS finds the optimal solution 10 times faster with $\rho = 0.999$ than with $\rho = 0.8$, and for several random instances this effect is much more pronounced. It is interesting to note that the very small amount of smoothing performed at $\rho = 0.999$ can be rather important: Without smoothing ($\rho = 1$), we found rare but conclusive evidence for search stagnation on random networks.

Secondly, the subsidiary local search procedure in GLS uses a computationally efficient first-improvement strategy, whereas GLS⁺ employs a more powerful best-improvement procedure in conjunction with newly developed, powerful strategies for caching and updating the effects of variable flips on the evaluation function. While previously used caching schemes only locally update the partial evaluation function value involving variables in the Markov blanket of a flipped variable, we developed two substantial improvements. At every step, we maintain the score of flipping each variable to any of its values (caching scheme *Scores*), and on top of that the set of all variables that lead to an improvement in evaluation function value when flipped to some value (caching scheme *Improving*); since after an initial search phase this quantity is a low constant in practice, this latter caching scheme enables an evaluation of the whole neighbourhood of a search position in constant time. Figure 1(b) demonstrates the large performance gains of our new caching schemes over the previous state-of-the-art caching scheme.

Thirdly, GLS initializes the search randomly. However, it has been shown that strong initial solutions, such as the ones obtained with MB, lead to much better overall solutions [Kask and Dechter, 1999]. Consequently, GLS⁺ initializes its search using the MB variant MB-w(10^5) (see Section 2), which improves the quality of found solutions considerably and for some instances speeds up the search for optimal solutions by up to two orders of magnitude.

We also study a version of GLS⁺ that has an additional preprocessing stage based on VE. In this preprocessing, VE is applied until a potential with more than b entries is obtained, where b is a parameter ($b = 0$ results in the standard

³In order to achieve a meaningful guidance by this evaluation function in areas of the search space with probability zero, we treat $\log(\phi[\mathbf{V} = \mathbf{v}])$ as -10^4 for $\phi[\mathbf{V} = \mathbf{v}] = 0$; we also fixed the weighting factor w to 10^4 to make the penalties comparably large.

Distribution	Stats		GLS ⁺ (default)	GLS (orig)	BBMB		AOMB	
	W	Opt			static(10)	dynamic(4)	static(10)	dynamic(6)
Random networks								
Base-line: $N = 100, K = 2, P = 2$	16.5	100	0.11	1.88(1/0)	0.05	1.69	0.33	1.10
$N = 200, K = 2, P = 2$	34.8	0	0.70	33.97(68/0.8464)	61.95(88/0.2694)	47.66(86/0.4168)	-(100/0)	-(100/0)
$N = 400, K = 2, P = 2$	72.6	0	10.13	-(100/0.0355)	-(100/0.0020)	-(100/0.0254)	-(100/0)	-(100/0)
$N = 100, K = 3, P = 2$	16.5	100	0.34	37.15(81/0.7392)	5.51	28.90(7/0.5608)	15.38	33.72(20/0)
$N = 100, K = 4, P = 2$	16.3	60	0.88	-(100/0.2225)	21.86(25/0.4856)	29.88(65/0.5071)	42.41(53/0)	59.02(91/0)
$N = 100, K = 2, P = 3$	28.3	47	0.42	13.26(3/0.9659)	36.12(48/0.3650)	31.49(74/0.3884)	43.01(62/0)	53.17(62/0)
$N = 100, K = 2, P = 4$	37.0	0	0.81	41.89(4/0.8538)	46.83(98/0.1079)	2.07(99/0.1502)	-(100/0)	-(100/0)
Grid networks								
Base-line: $N = 10, K = 2$	11.8	100	0.01	4.55(5/0.1989)	0.05	20.23(7/0.6847)	0.13	2.56
$N = 15, K = 2$	22.4	0	0.90	50.00(99/0.4201)	9.75(95/0.2686)	-(100/0.0880)	-(100/0)	-(100/0)
$N = 20, K = 2$	32.8	0	30.01	-(100/0.0031)	-(100/0.0058)	-(100/0.0015)	-(100/0)	-(100/0)
$N = 25, K = 2$	44.3	0	52.06	-(100/0)	-(100/0.0002)	-(100/0)	-(100/0)	-(100/0)
$N = 10, K = 3$	12.0	100	0.34	-(100/0.4955)	0.98	47.16(97/0.3187)	1.90	43.29(53/0)
$N = 10, K = 4$	11.7	95	1.17	-(100/0.0496)	4.77(3/0.5950)	-(100/0.1830)	15.59(4/0)	68.32(95/0)
$N = 10, K = 5$	11.8	84	5.05	-(100/0.0098)	6.41(9/0.5619)	-(100/0.1334)	76.52(33/0)	-(100/0)
$N = 10, K = 6$	11.9	0	17.89	-(100/0.0049)	-(100/0)	-(100/0.0940)	-(100/0)	-(100/0)

Table 1: Scaling of performance for random networks and grid networks with network size N , domain size K , and number of parents P . For each problem distribution, there are 100 networks. W gives their average induced width, Opt the number of networks for which our quasi-optimal solutions are provably optimal (we never found a better solution than the quasi-optimal one). For each algorithm and problem distribution, we list the average time to find a quasi-optimal solution. If an algorithm did not find the quasi-optimal solution for all instances within 100 CPU seconds, we give its average time for its solved instances, and in parentheses its number of unsolved instances, followed by its average approximation quality for these instances.

GLS⁺ algorithm, and $b = \infty$ yields pure VE). The resulting reduced network can then quickly be solved with GLS⁺, and the eliminated variables can be instantiated optimally in linear time like in regular VE.

4 Experimental Results

We conducted a number of computational experiments to compare the performance and scaling behaviour for the original GLS algorithm, GLS⁺, and the current state-of-the-art in MPE solving.⁴ In [Marinescu *et al.*, 2003], B&B with static MB heuristic (s-BBMB) and a B&B algorithm with MB tree elimination heuristic (BBBT) were claimed to be the state-of-the-art for MPE solving. [Marinescu and Dechter, 2004] then introduced B&B with dynamic MB heuristic (d-BBMB), as well as versions of s-BBMB and d-BBMB that employ an AND/OR search tree; these are called s-AOMB and d-AOMB.

We used Radu Marinescu’s C++ implementations of s-BBMB, d-BBMB, s-AOMB, and d-AOMB.⁵ Furthermore, we used Marinescu’s C++ implementation of GLS instead of the original Java implementation by Park [Park, 2002], since the former was orders of magnitude faster than the latter on all instances we tried; our GLS⁺ implementation is also written in C++. We employed a fixed parameter setting of $\langle \rho, N_\rho \rangle = \langle 0.999, 200 \rangle$ for GLS⁺ and the overall best-performing fixed i -bound $ib \in \{2, 4, 6, 8, 10, 12\}$ for each B&B algorithm. The preprocessing stage in GLS⁺ only improves its performance for structured networks. Thus, we set

⁴All experiments were carried out on compute servers each equipped with dual 2GHz Intel Xeon CPUs with 512KB cache and 4GB RAM running Linux version 2.4.20, build 28.9. Our implementation and the benchmark instances we used are available online at <http://www.cs.ubc.ca/labs/beta/Projects/SLS4MPE/>

⁵We do not report results for BBBT, since the implementation available to us could only correctly handle networks generated within the REES system (<http://www.ics.uci.edu/~radum/rees.html>).

$b = 0$ in the experiments on random instances.

In the first two experiments, for each network, we executed each algorithm once for a maximum of 100 CPU seconds and call the best solution found in any such run of an algorithm *quasi-optimal*. For the second and third experiment, we ran the B&B algorithms for a long time to obtain provably optimal solution qualities, which we found to always agree with our quasi-optimal solutions. For a fair comparison, we always report the times each algorithm requires for finding a quasi-optimal solution; only for the third experiment we additionally report the time required for proving optimality. Finally, we define the *approximation quality* of an algorithm run as the ratio of the probability of the solution it found and the quasi-optimal probability.

Our first experiment evaluates how the various algorithms scale with important instance characteristics, such as network size (N), domain size of the variables (K), and network density, here controlled by the number P of parents of each node. We created instances with a random network generator provided by Radu Marinescu (this is, e.g., described in [Marinescu *et al.*, 2003]), topologically sampled all variables to guarantee non-zero joint probability, and randomly picked 10 evidence variables. Table 1 shows that for small and easy networks, identified as “Base-line” in the table, the original GLS and the B&B algorithms are competitive with GLS⁺, but that when scaling any of N , K , or P , the performance of all algorithms except GLS⁺ degrades rapidly. [Marinescu *et al.*, 2003] showed that s-BBMB scales much better with domain size K than the original GLS algorithm. While this is confirmed by our experiments (see Table 1), our results in Figure 2(a) show that GLS⁺ substantially outperforms s-BBMB for larger K , and that the relative variability in run-time increases with K for s-BBMB, but remains constant for GLS⁺.

GLS⁺ does not only outperform the other algorithms in terms of runtime for finding quasi-optimal solutions, but also in terms of solution quality found in a given fixed time. This

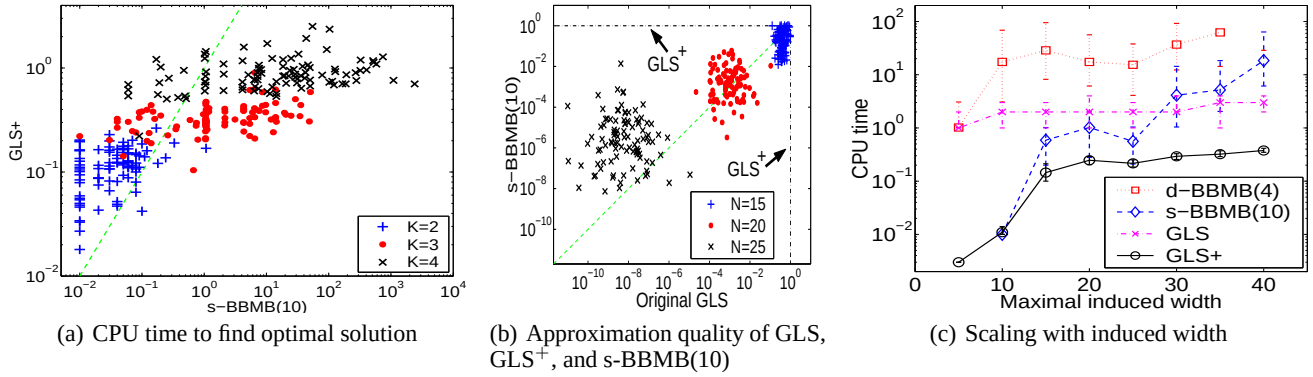


Figure 2: (a) CPU time required by GLS^+ and $\text{s-BBMB}(10)$ to solve random network instances to optimality. Each point is the result of a single run on one instance. The instances are the same as summarized in Table 1. The same CPU-time for both algorithms is indicated by the line. (b) Comparison of the quality reached by the three algorithms GLS , GLS^+ , and $\text{s-BBMB}(10)$ within 100 CPU seconds. For all instances, GLS^+ found the best known solution quality, thus its approximation quality is always 1. Each point in the figure is the result of a single run for one instance by GLS and $\text{s-BBMB}(10)$, while the two lines show the approximation quality of GLS^+ . (c) Scaling with induced width for random networks with 100 variables, domain size 2, and maximal node degree 5. For each induced width, based on one run on each of 100 instances, we plot the median runtime for finding the optimal solution. The errorbars are based on the quantiles $q_{0.25}$ and $q_{0.75}$. We do not employ means and standard deviations since the B&B algorithms did not succeed in finding an optimal solution for every network. For example, for the networks of induced width 40, GLS^+ took 0.5 seconds in the worst case, whereas $\text{s-BBMB}(10)$ failed to find the optimal solution for 18 of the 100 networks within 100 seconds.

can be seen from Table 1 for all of N , K and P , and is visualized in Figure 2(b) for scaling N in random grid networks. This scatter plot compares the original GLS , GLS^+ , and s-BBMB all given the same maximum computation time; this is possible since GLS^+ always finds the quasi-optimal solutions, such that its approximation quality is constantly 1. Even though $\text{s-BBMB}(10)$ scales better than the original GLS , GLS^+ shows even better scaling behaviour. Once again, not only the average quality difference to GLS^+ grows, but for both GLS and s-BBMB , the relative variability also increases with network size.

These scaling results are further extended in our second experiment, where we studied the impact of the induced width of networks on the performance of GLS^+ vs. GLS and the B&B algorithms. For this, we used networks generated with BNGenerator ,⁶ which allows to generate networks with a rather accurate upper bound on the induced width. The results in Figure 2(c) show that the induced width has a major effect on algorithm performance, and that GLS and GLS^+ scale much better with induced width than the B&B algorithms (we omit s-AOMB and d-AOMB which were worse than d-BBMB). We attribute this to the B&B algorithm’s MB heuristic whose guidance is impaired for high induced widths.

In our third experiment we studied real-world networks from the Bayesian Network Repository.⁷ Here, GLS^+ also performs very well, except for large networks with low induced width, which are easily solved with VE. As we show in Table 2, a short preprocessing stage ($b > 0$) significantly improves the performance of GLS^+ for these structured networks, making it faster than all B&B algorithms (with optimal i -bound) for all but one network.⁸ In particular, note

that the Link network could not be solved by any of the B&B algorithms, while GLS and GLS^+ find the optimal solution in one CPU second. Interestingly, $\text{MB}(2)$ already finds a tight upper bound on solution quality (in 10 CPU milliseconds), but with feasible i -bounds never even comes close to a tight lower bound. Consequently, only a combination of SLS and MB can find the optimal solution and prove its optimality (and this within one CPU second).

5 Conclusion and Future Work

In this work, we identified various weaknesses of GLS , the previously best-performing SLS algorithm for the MPE problem, and introduced GLS^+ , a novel variant of GLS that pays more attention to such important concerns as algorithmic complexity per search step, thorough parameter tuning, and strong guidance by the evaluation function. For a wide range of MPE instances, GLS^+ widely outperforms GLS and the best-performing exact algorithms (all of which are B&B algorithms) that defined the state-of-the-art in MPE solving. Most importantly, we demonstrated that the performance of GLS^+ scales much better than GLS and B&B algorithms with the network and domain size, as well as with network density and induced width. GLS^+ also shows the best performance for real-world instances.

In contrast to recent claims that stochastic local search algorithms are not competitive for MPE solving [Marinescu *et al.*, 2003], our results establish SLS as a state-of-the-art approach for MPE solving that merits further investigation. In particular, the anytime characteristics and excellent scaling

without the preprocessing ($b = 0$), we performed 25 additional runs of two hours on the instances *Diabetes* and *Munin4*. Out of these, eleven runs solved the *Diabetes* network, but the *Munin4* network was never solved. This highlights the importance of preprocessing for structured networks.

⁶<http://www.pmr.poli.usp.br/ltd/Software/BNGenerator/>

⁷<http://www.cs.huji.ac.il/labs/compbio/Repository/>

⁸To study whether GLS^+ eventually finds the optimum even

Instance	Stats			GLS ⁺	GLS ⁺			VE	GLS (orig)	BBMB				AOMB			
	<i>N</i>	<i>K</i>	<i>w</i>	with <i>b</i> = 0	optimal	<i>b</i>	<i>N</i> _{rest}			static	ib	dynamic	ib	static	ib	dynamic	ib
Barley	48	8.77	7	19.22	17.95	10 ⁴	31	-	(-)	2.58/2.58	6	6.28/68.91	4	3.08/3.08	6	43.37/43.37	4
Diabetes	413	11.34	4	(0.0099)	3.61	≥ 10 ⁶	0	3.61/3.61	(-)	4.55/4.55	≥6	-	-	8.64/8.64	≥6	162.28/162.28	8
Link	724	2.53	15	1.25	1.16	10 ²	352	-	1.0	(0.0050)	10	(0.0050)	4	-	-	-	-
Munin1	189	5.26	11	0.34	0.34	10 ⁴	45	-	1.0	1.25/10.57	6	1.03/45.84	4	6.42/ 6.42	6	38.15/38.15	4
Munin2	1003	5.36	7	0.96	0.91	10 ⁴	79	1.9/1.9	1.0	3.98/6.55	8	76.36/-	6	2.77/2.77	6	10.24/40.24	8
Munin3	1044	5.37	7	0.87	0.83	10 ⁴	62	1.5/1.5	(-)	4.34/4.34	12	27.59/44.10	10	2.39/2.39	6	12.92/12.92	6
Munin4	1041	5.43	8	(0.0347)	1.87	10 ⁴	157	17.69/17.69	(-)	21.30/24.75	12	(0.0090)	4	20.77/20.77	8	128.24/128.24	8

Table 2: Performance on networks from the Bayesian Network Repository. *N* is the number of variables of the network, *K* its average domain size, and *w* its induced width. For GLS⁺, *b* is the limit on the number of entries for applying VE in the extended version of GLS⁺ and *N_{rest}* is the remaining number of variables after the preprocessing with that bound; we report the combined time for preprocessing and search. For each network and SLS algorithm, if all of 25 runs found the optimal solution within 600 CPU seconds, we give their average runtime. Otherwise, we give the average approximation quality in parentheses. For each B&B algorithm, we report the best result for any *i*-bound in {2, 4, 6, 8, 10, 12}. If a B&B algorithm found the optimal solution within 600 CPU seconds, we give the time it required to find it, followed by the time it required to prove optimality (if applicable). If it did not find the optimal solution, we give its approximation quality in parentheses. For each network, we highlight the fastest solution time and the fastest proof time. Remarkably, the B&B algorithms often prove optimality shortly after finding the optimal solution, which suggests that this task is only slightly harder. The networks *Alarm*, *Insurance*, *Hailfinder*, *Mildew*, *Pigs*, and *Water* are not listed in this table since they are always solved in well below a second by all algorithms except the original GLS, which does not solve *Mildew*.

behaviour of GLS⁺ suggest its use for large problems with real-time constraints, such as finding the MPE in pairwise MRFs for early computer vision [Boykov *et al.*, 2001]. We therefore plan a comparative study of GLS⁺, generalized BP and graph cuts, which currently define the state-of-the-art in that domain [Tappen and Freeman, 2003]. Furthermore, in the near future we plan to implement an extension of GLS⁺ that computes a diverse set of high-quality MPEs—the SLS paradigm accommodates such an extension easily, for example, by using the best solutions encountered along the search trajectory.

Acknowledgments. We would like to thank Radu Marinescu for providing implementations of GLS and of several B&B algorithms, as well as code for the generation of Bayes nets. We also thank James Park for providing his original GLS code.

References

- [Boykov *et al.*, 2001] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *PAMI*, 23(11), 2001.
- [Cowell *et al.*, 1999] R. G. Cowell, A. P. Dawid, S. L. Lauritzen, and D. J. Spiegelhalter. *Probabilistic Networks and Expert Systems*. Statistics for Engineering and Information Science. Springer, 1999.
- [Dechter and Rish, 2003] R. Dechter and I. Rish. Mini-buckets: A general scheme for bounded inference. *J. of the ACM*, 50(2):107–153, 2003.
- [Dechter, 1999] R. Dechter. Bucket elimination: A unifying framework for reasoning. *AIJ*, 113(1-2):41–85, 1999.
- [Hoos and Stützle, 2004] H. H. Hoos and T. Stützle. *Stochastic Local Search - Foundations & Applications*. Morgan Kaufmann, SF, CA, USA, 2004.
- [Hutter, 2004] F. Hutter. Stochastic local search for solving the most probable explanation problem in Bayesian networks. Master’s thesis, Darmstadt University of Technology, Darmstadt, Germany, September 2004.
- [Jaakkola and Jordan, 1999] T. S. Jaakkola and M. I. Jordan. Variational probabilistic inference and the QMR-DT network. *JAIR*, 10:291–322, 1999.
- [Kask and Dechter, 1999] K. Kask and R. Dechter. Stochastic local search for Bayesian networks. In *Proc. of AISTATS-99*, January 1999.
- [Marinescu and Dechter, 2004] R. Marinescu and R. Dechter. AND/OR tree search for constraint optimization. In *Soft’04 workshop at CP’2004*, 2004.
- [Marinescu *et al.*, 2003] R. Marinescu, K. Kask, and R. Dechter. Systematic vs. non-systematic algorithms for solving the MPE task. In *Proc. of UAI-03*, 2003.
- [Mills and Tsang, 2000] P. Mills and E. P. K. Tsang. Guided local search for solving SAT and weighted MAX-SAT problems. In *SAT2000*, pages 89–106. IOS Press, Amsterdam, The Netherlands, 2000.
- [Park, 2002] J. D. Park. Using weighted MAX-SAT engines to solve MPE. In *Proc. of AAAI-02*, pages 682–687, 2002.
- [Pearl, 1988] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, SF, CA, USA, 1988.
- [Rish *et al.*, 2002] I. Rish, M. Brodie, and S. Ma. Accuracy vs. efficiency trade-offs in probabilistic diagnosis. In *Proc. of AAAI-02*, pages 560–566, 2002.
- [Tappen and Freeman, 2003] M. F. Tappen and W. T. Freeman. Comparison of graph cuts with belief propagation for stereo, using identical MRF parameters. In *Proc. of ICCV-03*, volume 2, pages 900 – 906, 2003.
- [Voudouris and Tsang, 1999] C. Voudouris and E. P. K. Tsang. Guided Local Search and its application to the travelling salesman problem. *EJOR*, 113(2):469–499, 1999.
- [Yanover and Weiss, 2003] C. Yanover and Y. Weiss. Approximate inference and protein-folding. In *Proc. of NIPS-02*, pages 1457–1464, 2003.
- [Yedidia *et al.*, 2002] J.S. Yedidia, W.T. Freeman, and Y. Weiss. Understanding belief propagation and its generalizations. Technical report, Mitsubishi Electrical Research Laboratories Inc., TR-2001-22, 2002.