

COGNITIVE ROBOTICS

Body Movement Analysis of Human-Robot Interaction

Takayuki Kanda, Hiroshi Ishiguro, Michita Imai, and Tetsuo Ono

ATR Intelligent Robotics & Communication Laboratories
2-2-2 Hikaridai, Seika-cho, Soraku-gun, Kyoto, 619-0288, Japan
kanda@atr.co.jp

Abstract

This paper presents a method for analyzing human-robot interaction by body movements. Future intelligent robots will communicate with humans and perform physical and communicative tasks to participate in daily life. A human-like body will provide an abundance of non-verbal information and enable us to smoothly communicate with the robot. To achieve this, we have developed a humanoid robot that autonomously interacts with humans by speaking and making gestures. It is used as a testbed for studying embodied communication. Our strategy is to analyze human-robot interaction in terms of body movements using a motion capturing system, which allows us to measure the body movements in detail. We have performed experiments to compare the body movements with subjective impressions of the robot. The results reveal the importance of well-coordinated behaviors and suggest a new analytical approach to human-robot interaction.

1 Introduction

Over the past several years, many humanoid robots such as Honda's [Hirai *et al.*, 1998] have been developed. We believe that in the not-too-distant future humanoid robots will interact with humans in our daily life. Their human-like bodies enable humans to intuitively understand their gestures and cause people to unconsciously behave as if they were communicating with humans [Kanda *et al.*, 2002a]. That is, if a humanoid robot effectively uses its body, people will naturally communicate with it. This could allow robots to perform communicative tasks in human society such as route guides.

Several researchers have investigated about social relationships between humans and robots. For example, Kismet was developed for studying early caregiver-infant interaction [Breazeal, 2001]. Also, a robot that stands in a line [Nakauchi *et al.*, 2002] and a robot that talks with multiple persons [Nakadai *et al.*, 2001] have been devel-

oped. Furthermore, various communicative behaviors using a robot's body have been discovered, such as a joint-attention mechanism [Scassellati *et al.*, 2000].

On the other hand, methods of analyzing social robots, especially with respect to human-robot interaction, are still lacking. To effectively develop any systems in general, it is essential to measure the systems' performance. For example, algorithms are compared with respect to time and memory, and mechanical systems are evaluated by speed and accuracy. Without analyzing current performance, we cannot argue advantages and problems. For social robots, no analysis method has yet been established, thus, it is vital to determine what types of measurements we can apply. Although questionnaire-based methods have been often used, they are rather subjective, static and obtrusive (that is, we would interrupt the interaction when we apply a questionnaire). Less commonly, human behaviors are employed for this purpose, such as distance [Hall, 1966], attitude [Reeves and Nass, 1996], eye gaze (often used in psychology), and synchronized behaviors [Ono *et al.*, 2001]. Although those methods are more difficult to apply, the results are more objective and dynamic. Sometimes, interactive systems observe human behaviors for synthesizing behaviors [Jebara and Pentland, 1999]. However, they are still fragments rather than a systematic analysis method applicable for human-robot interaction.

In this paper, we present our exploratory analysis of human-robot interaction. Our approach is to measure the body movement interaction between a humanoid robot and humans, and compare the results with traditional subjective evaluation. We have developed an interactive humanoid robot that has a human-like body as the testbed of this embodied communication. Furthermore, many interactive behaviors have been implemented. It encourages people to treat the robot as a human child. We employ a motion capturing system for measuring time and space accurately.

2 An Interactive Humanoid Robot

2.1 Hardware

Figures 1 and 2 display an interactive humanoid robot "Robovie," which is characterized by its human-like body expression and various sensors. The human-like body

* This research was supported in part by the Telecommunications Advancement Organization of Japan.

consists of eyes, a head and arms, which generate the complex body movements required for communication. The various sensors, such as auditory, tactile, ultrasonic, and visual, enable it to behave autonomously and to interact with humans. Furthermore, the robot satisfies mechanical requirements of autonomy. It includes all computational resources needed for processing the sensory data and for generating behaviors. It can continually operate for four hours with its battery power supply.

2.2 Software

Using its body and sensors, the robot performs diverse interactive behaviors with humans. Each behavior is generated by a *situated module*, each of which consists of *communicative units*. This implementation is based on a constructive approach [Kanda *et al.*, 2002b]: “combining as many simple behavior modules (*situated modules*) as possible.” We believe that the complexity of the relations among appropriate behaviors enriches the interaction and creates perceived intelligence of the robot.

Communicative Unit

Previous works in cognitive science and psychology have highlighted the importance of eye contact and arm movement in communication. *Communicative units* are designed based on such knowledge to effectively use the robot’s body, and each unit is a sensory-motor unit that realizes certain communicative behavior. For example, we have implemented “eye contact,” “nod,” “positional relationship,” “joint attention (gaze and point at object).” When developers create a *situated module*, they combine the *communicative units* at first. Then, they supplement it with other sensory-motor units such as utterances and positional movements for particular interactive behaviors.

Situated Modules

In linguistics, an adjacency pair is a well-known term for a unit of conversation where the first expression of a pair requires the second expression to be of a certain type. For example, “greeting and response” and “question and answer” are considered pairs. We assume that embodied communication is materialized with a similar principle: the action-reaction pair. This involves certain pairs of actions and reactions that also include non-verbal expressions. The continuation of the pairs forms the communication between humans and a robot.

Although the action and reaction happen equally, the recognition ability provided by current computer science is not as powerful as that of humans. Thus, the robot takes the initiative and acts rather than reacting to humans actions. This allows the flow of communication to be maintained. Each *situated module* is designed to realize a certain action-reaction pair in a particular situation (Fig. 3), where a robot mainly takes an action and recognizes the humans’ reaction. Since it produces a particular situation by itself, it can recognize humans’ complex reactions under limited conditions; that is, it expects the human’s reaction. This policy enables developers to easily implement many *situated modules*. On the other hand, when a human takes an action



Figure 1:
“Robovie”

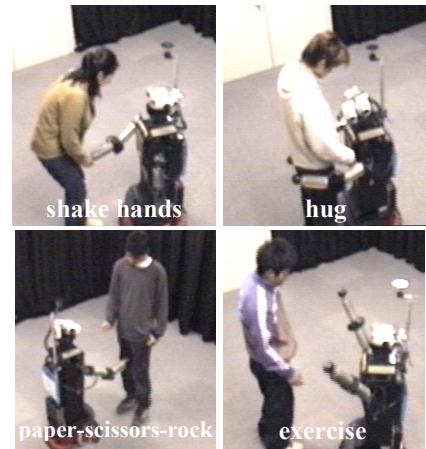


Figure 2: Interactive behaviors

toward the robot, it recognizes the human’s action and reacts by using reactive transition and *reactive modules*; that is, some of the *situated modules* can catch the human’s initiating behaviors and interrupt its operations to react to them (as shown in Fig. 4: the second module TURN).

A *situated module* consists of *precondition*, *indication*, and *recognition parts* (Fig. 3). By executing the *precondition*, the robot checks whether the *situated module* is in an executable situation. For example, the *situated module* that performs a handshake is *executable* when a human is in front of the robot. By executing the *indication part*, the robot interacts with humans. With the handshake module, the robot says “Let’s shake hands,” and offers its hand. This behavior is implemented by combining *communicative units* of eye contact and positional relationships (it orients its body toward the human), and by supplementing a particular utterance (“Let’s shake hands”) and a particular body movement (offering its hand). The *recognition part* is designed to recognize several expected human reactions toward the robot’s action. As for the handshake module, it can detect human handshake behavior if a human touches its offered hand.

The robot system sequentially executes *situated modules* (Fig. 4). At the end of the current *situated module* execution, it records the recognition result obtained by the recognition part, and progresses to the next executable *situated module*. The next module is determined by the results and the execution history of previous *situated modules*, which is similar to a state transition model.

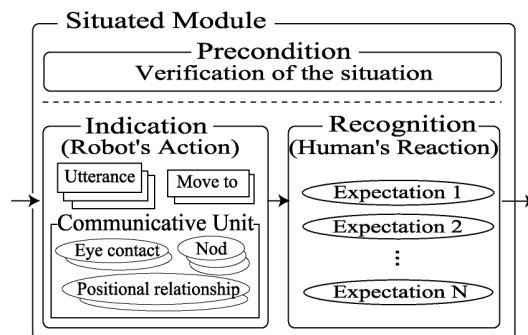


Figure 3: Situated module

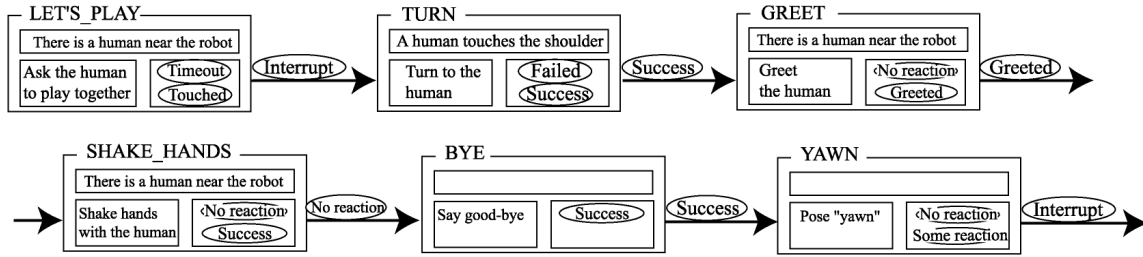


Figure 4: Example of situated module transition

2.3 Realized Interactive Behaviors

We installed this mechanism on “Robovie.” The robot’s task is to perform daily communication as children do. The number of developed *situated modules* has reached a hundred: 70 of which are interactive behaviors such as handshake (Fig. 2, upper-left), hugging (Fig. 2, upper-right), playing paper-scissors-rock (Fig. 2, lower-left), exercising (Fig. 2, lower-right), greeting, kissing, singing a song, short conversation, and pointing to an object in the surroundings; 20 are idling behaviors such as scratching its head, and folding its arms; and 10 are moving-around behaviors, such as pretending to patrol an area and going to watch an object in the surroundings.

Basically, the transition among the *situated modules* is implemented as follows: it sometimes asks humans for interaction by saying “Let’s play, touch me,” and exhibits idling and moving-around behaviors until a human acts in response; once a human reacts to the robot (touches or speaks), it starts and continues the friendly behaviors while the human reacts to these; when the human stops reacting, it stops the friendly behaviors, says “good bye” and re-starts its idling or moving-around behaviors.

3 Body Movement Analysis

3.1 Experiment Settings

We performed an experiment to investigate the interaction of body movements between the developed robot and a human. We used 26 university students (19 men and 7 women) as our subjects. Their average age was 19.9. First, they were shown examples how to use the robot, then they freely observed the robot for ten minutes in a rectangular room 7.5 m by 10 m. As described in section 2.3, the robot autonomously tries to interact with subjects. At the beginning of the free observation, the robot asks subjects to talk and play together, and then subjects usually start touching and talking.

After the experiment, subjects answered a questionnaire about their subjective evaluations of the robot with five adjective pairs shown in Table 1, which was compared with the body movements. We chose these adjective pairs because they had high loadings as evaluation factors for an interactive robot in a previous study [Kanda *et al.*, 2002a].

3.2 Measurement of Body Movements

We employed an optical motion capturing system to measure the body movements. The motion capturing sys-

tem consisted of 12 pairs of infrared cameras and infrared lights and markers that reflect infrared signals. These cameras were set around the room. The system calculates each marker’s 3-D position from all camera images. The system has high resolution in both time (120 Hz) and space (accuracy is 1 mm in the room)

As shown in Fig. 5, we attached ten markers to the heads (subjects wore a cap attached with markers), shoulders, necks, elbows, and wrists of both the robot and the subjects. By attaching markers to corresponding places on the robot and subjects, we could analyze the interaction of body movements. The three markers on the subjects’ head detect the individual height, facing direction, and potential eye contact with the robot. The markers on the shoulders and neck are used to calculate the distance between the robot and subjects, and distance moved by them. The markers on the arms provide hand movement information (the relative positions of hands from the body) and the duration of synchronized movements (the period where the movements of hands of the subject and robot highly correlate). We also analyzed touching behaviors via an internal log of the robot’s touch sensors.

3.3 Results

Comparison between the body movements and the subjective evaluations indicates meaningful correlation. From the experimental results, well-coordinated behaviors such as eye contact and synchronized arm movements proved to be important. This suggests that humans make evaluations based on their body movements.

Subjective Evaluation: “Evaluation Score”

The semantic differential method is applied to obtain subjective evaluations with a 1-to-7 scale, where 7 denotes the most positive point on the scale. Since we chose the adjective pairs that had high loadings as evaluation factors for an interactive robot, the results of all adjective pairs represent subjective evaluation of the robot. Thus, we calculated the *evaluation score* as the average of all adjective-pairs’ scores. Table 1 indicates the adjective pairs used, the averages, and standard deviations.

Correlation between Body Movements and Subjective Impressions

Table 2 displays the measured body movements. Regarding eye contact, the average time was 328 seconds, which is more than half of the experiment time. Since the robot’s eye height was 1.13 m and the average of subject

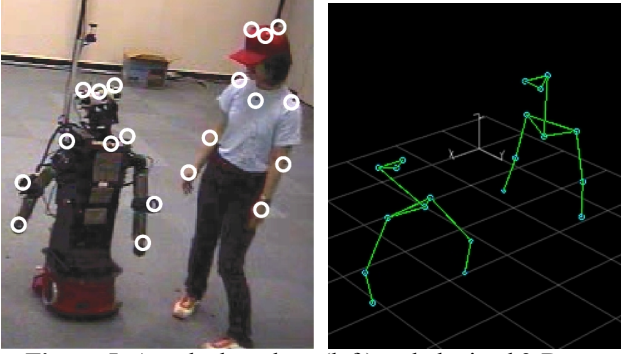


Figure 5: Attached markers (left) and obtained 3-D numerical position data of body movement (right)

In the left figure, white circles indicate the attached markers, and the circles in the right figure indicate the observed position of the markers

eye height was 1.55 m, which was less than their average standing eye height of 1.64 m, several subjects sat down or stooped to bring their eyes to the same height as the robot's. The distance moved was farther than what we expected, and it seemed that subjects were always moving little-by-little. For example, when the robot turned, the subjects would then correspondingly turn around the robot. Some subjects performed arm movements synchronized with the robot's behaviors, such as exercising.

Next, we calculated the correlation between the evaluation score and the body movements (Table 3). Since the number of subjects is 26, each correlation value whose absolute value is larger than 0.3297 is significant. We highlight these significant values with bold face in the table. From the calculated results, we found that eye contact and synchronized movements indicated higher significant correlations with the evaluation score.

According to the correlations among body movements, the following items showed significant correlations: eye contact – distance, eye contact – distance moved, synchronized behaviors – distance moved by hands, and synchronized behaviors – touch. However, these items (distance, distance moved, distance moved by hands, and touch) do not significantly correlate with the evaluation score. That is, only the well-coordinated behaviors correlate with the subjective evaluation. Isolated active body movements of subjects, such as standing near the robot, moving their hands energetically, and touching the robot repetitively, do not correlate to the subjective evaluation.

Estimation of Momentary Evaluation: “Entrainment Score”

The results indicate that there are correlations between subjective evaluation and body movements. We performed multiple linear regression analysis to estimate the evaluation score from the body movements, which confirms the above analysis and reveals how much each body movement affects the evaluation. We then applied the relations among body movements to estimate a momentary evaluation score called the *entrainment score*.

As a result of the multiple linear regression analysis, standardized partial regression coefficients were obtained,

Adjective-pairs		Mean	Std. Dev.
Good	Bad	4.88	0.95
Kind	Cruel	4.85	1.29
Pretty	Ugly	5.08	0.93
Exciting	Dull	4.46	1.61
Likable	Unlikable	4.77	1.03
Evaluation score		4.81	0.92

Table 1: The adjective-pairs used for subjective evaluation, and the mean and resulting mean and standard deviation

	Mean	Std. Dev.
Distance (m)	0.547	0.103
Eye contact (s)	328	61.8
Eye height (m)	1.55	0.124
Distance moved (m)	35.2	17.0
Distance moved by hands (m)	108	29.5
Synchronized movements (s)	7.95	6.58
Touch (num. of times)	54.9	20.8

Table 2: Results for body movement

as shown in Table 4. The obtained multiple linear regression is as follows:

$$E = \alpha_{dist} \bullet DIST + \alpha_{ec} \bullet EC + \alpha_{eh} \bullet EH + \alpha_{dm} \bullet DM + \alpha_{dmh} \bullet DMH + \alpha_{sm} \bullet SM + \alpha_{touch} \bullet TOUCH + \alpha_{const} \quad (1)$$

where *DIST*, *EC*, *EH*, *DM*, *DMH*, *SM*, and *TOUCH* are the standardized values of the experimental results for the body movements. Since the evaluation was scored on a 1-to-7 scale, evaluation score *E* is between 1 and 7. The multiple correlation coefficient is 0.77, thus 59% of the evaluation score is explained by the regression. The validity of the regression is proved by analysis of variance ($F(7,18) = 3.71, P < 0.05$).

The coefficients (Table 4) also indicate the importance of well-coordinated behaviors. Eye contact and synchronized movements positively affected the evaluation score; on the contrary, distance, distance moved and touch seem to have negatively affected the evaluation score. In other words, the subjects who just actively did something (standing near the robot, moved around, and touched repeatedly), especially without cooperative behaviors, did not evaluate the robot highly.

Because we can momentarily observe all terms involved in the body movements of the regression (1), we can estimate a momentary evaluation score by using the same relations among body movements as follows:

$$E(t) = \alpha_{dist} \bullet DIST(t) + \alpha_{ec} \bullet EC(t) + \alpha_{eh} \bullet EH(t) + \alpha_{dm} \bullet DM(t) + \alpha_{dmh} \bullet DMH(t) + \alpha_{sm} \bullet SM(t) + \alpha_{touch} \bullet TOUCH(t) + \alpha_{const} \quad (2)$$

where designations such as *DIST(t)* are the momentary values of the body movements at time *t*. We named this momentary evaluation score the “*entrainment score*,” with the idea that the robot entrains humans into interaction through its body movements and humans move their body according to their current evaluation of the robot. The

	Evaluation	Dist.	E. C.	E. H.	D.M.	D.M. H.	S. M.	Touch
Dist.	-0.04	1.00						
E.C.	0.57	-0.47	1.00					
E.H.	0.08	-0.39	0.29	1.00				
D.M.	-0.32	0.20	-0.43	0.02	1.00			
D.M.H.	0.01	-0.04	-0.21	-0.09	0.49	1.00		
S. M.	0.54	-0.05	0.28	-0.05	0.15	0.61	1.00	
Touch	0.21	-0.45	0.49	-0.07	-0.15	0.35	0.41	1.00

Table 3: Correlation between subjective evaluation and body movements. (E.C.: eye contact, E.H.: eye height, D. M.: distance moved, D.M.H.: distance moved by hands, S.M.: synchronized movements)

evaluation score and entrainment score satisfy the following equation, which represents our hypothesis that the evaluation forms during the interaction occurring through the exchange of body movements:

$$E = \int_0^t E(t) / t \quad (3)$$

Let us show the validity of the estimation by examining the obtained entrainment score. Figure 6 shows the entrainment scores of two subjects. The horizontal axis indicates the time from start to end (600 seconds) of the experiments. The solid line indicates the entrainment score $E(t)$, while the colored region indicates the average of the entrainment score $E(t)$ from the start to time t (this integration value grows the estimation of E at the end time).

The upper graph shows the score of the subject who interacted with the robot very well. She reported after the experiment that, “It seems that the robot really looked at me because of its eye motion. I nearly regard the robot as a human child that has an innocent personality.” This entrainment-score graph hovers around 5 and sometimes goes higher. This is because she talked to the robot while maintaining eye contact. She performed synchronized movements corresponding to the robot’s exercising behaviors, which caused the high value around 200 sec.

At the other extreme, the lower graph is for the subject who became embarrassed and had difficulty in interacting with the robot. The graph sometimes falls below 0. In particular, at the end of the experiment, it became unstable and even lower. He covered the robot’s eye camera, touched it like he was irritated, and went away from the robot. We consider that those two examples suggest the validity of the entrainment score estimation.

Evaluation of the implemented behaviors

In the sections above, we explained the analysis of body movement interaction. Here, we evaluate the implemented behaviors. Although the application of this result is limited to our approach, our findings also prove the validity and applicability of the entrainment score.

We calculated the evaluation score of each *situated module* based on the average of the entrainment score while each module was being executed. Tables 5 and 6 indicate the worst and best five modules, respectively, and their scores. The worst modules were not so interactive. SLEEP_POSE and FULLY_FED do not respond to human

	Coefficient	Value
Distance	α_{dist}	0.173
Eye contact	α_{ec}	0.476
Eye height	α_{eh}	0.019
Distance moved	α_{dm}	-0.228
Distance moved by hands	α_{dmh}	-0.029
Synchronized movements	α_{sm}	0.535
Touch	α_{touch}	-0.186

Table 4: Standardized partial regression coefficients obtained by multiple linear regression analysis

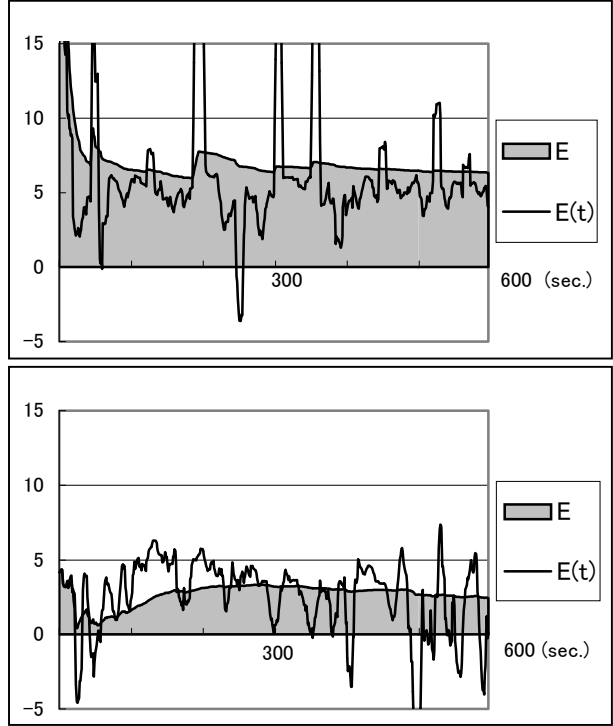


Figure 6: Illustration of entrainment score (upper: subject who treated the robot as if it were a human child, lower: subject who was embarrassed by interacting with it)

action and exhibit behavior similar to the sleeping pose. NOT_TURN is the behavior for brushing off a human’s hand while saying “I’m busy” when someone touches on its shoulder. The best modules were rather interactive modules that entrain humans into the interaction. EXERCISE and CONDUCTOR produce the exercising and imitating of musical conductor behaviors, which induced human synchronized body movements. Other highly rated modules also produce attractive behaviors, such as asking and calling, which induce human reactions. We believe that the entrainment scores provide plenty of information for developing interactive behaviors of robots that communicate with humans.

4 Discussions

The experiment reveals the correlation between humans’ subjective evaluations and body movements. If a human

ID	Contents	Evaluation
TICKLE	Tickle	-2.09
APOLOGIZE	Apologize	-1.96
NOT_TURN	Say, "I'm busy," and refuse to play together	-0.51
SLEEP_POSE	A pose of sleeping	-0.42
FULLY_FED	A pose of being fully fed	0.32

Table 5: Worst 5 *situated modules* based on average entrainment score

evaluates the robot highly, then the human behaves cooperatively with it, which will further improve its evaluation. That is, once they establish cooperative relationships with the robot, they interact well with the robot and evaluate the robot highly. Regarding evaluation of the implemented behaviors, the modules that entrain humans into interaction were highly evaluated, such as asking something that induces human's answer and producing cheerful body movements like exercising to let humans join and mimic the movements. We believe that the entrainment can help us to establish cooperative relationships between humans and robots.

Meanwhile, the multiple linear regression explains 59% of the subjective evaluation. This is remarkable because it is performed without regard to the contents or context of language communication. With speech recognition, the robot can talk with humans, although its ability is similar to that of a little child. Some of the subjects spoke to the robot. Often, there were requests for the robot to present particular behaviors (especially behaviors it had performed just previously), to which it sometimes responded correctly and sometimes incorrectly. To analyze this, we could use several analytical methods such as conversation analysis, however, these methods are rather subjective. On the other hand, our evaluation employed objective measures only: numerically obtained body movements without context, which means there could be a lot of potential usages. For example, an interactive robot could learn and adjust its behavior by using this method. It would be applicable to different subjects (age, culture, etc.), different agents (physical-virtual, body shape, behaviors, etc.), and inter-human communication.

5 Conclusions

This paper reported a new approach to analyzing embodied communication between humans and a robot. Our interactive humanoid robot is able to autonomously interact with humans. This complexity and autonomy is achieved by many simple behaviors. We measured humans' body movements while they observed and interacted with the robot, and the result of the analysis indicates positive correlations between cooperative body movements and subjective evaluations. Furthermore, the multiple linear regression explains 59% of the subjective evaluation without regard to language communication. We consider our approach of body movement analysis to be widely applicable in embodied communication.

ID	Contents	Evaluation
EXERCISE	Exercise	5.75
ASK_SING	Ask humans, "May I sing a song?"	5.59
CONDUCTOR	Pose imitating a musical conductor	4.85
WHERE_FROM	Ask humans, "Where are you from?"	4.55
LET'S_PLAY	Say, "Let's play, touch me"	4.24

Table 6: Best 5 *situated modules* based on average entrainment score

References

- [Breazeal *et al.*, 1999] C. Breazeal and B. Scassellati, A context-dependent attention system for a social robot, *Proc. Int. Joint Conf. on Artificial Intelligence*, pp.1146-1151, 1999.
- [Hall, 1966] E. Hall, *The Hidden Dimension*, Anchor Books/Doubleday
- [Hirai *et al.*, 1998] K. Hirai, M. Hirose, Y. Haikawa, and T. Takenaka, The development of Honda humanoid robot, *Proc. IEEE Int. Conf. on Robotics and Automation*, 1998.
- [Jebara and Pentland, 1999] T. Jebara and A. Pentland, Action Reaction Learning: Automatic Visual Analysis and Synthesis of Interactive Behaviour, *Int. Conf. on Computer Vision Systems*, 1999.
- [Kanda *et al.*, 2002a] T. Kanda, H. Ishiguro, T. Ono, M. Imai, and R. Nakatsu, Development and Evaluation of an Interactive Humanoid Robot "Robovie", *IEEE Int. Conf. on Robotics and Automation*, pp.1848-1855, 2002.
- [Kanda *et al.*, 2002b] T. Kanda, H. Ishiguro, M. Imai, T. Ono, and K. Mase, A constructive approach for developing interactive humanoid robots, *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pp. 1265-1270, 2002.
- [Nakadai *et al.*, 2001] K. Nakadai, K. Hidai, H. Mizoguchi, H. G. Okuno, and H. Kitano, Real-Time Auditory and Visual Multiple-Object Tracking for Robots, *Proc. Int. Joint Conf. on Artificial Intelligence*, pp. 1425-1432, 2001.
- [Nakauchi *et al.*, 2002] Y. Nakauchi and R. Simmons, A Social Robot that Stands in Line, *Autonomous Robots*, Vol. 12, No. 3, pp. 313 – 324, 2002.
- [Ono *et al.* 2001] T. Ono, M. Imai, and H. Ishiguro, A Model of Embodied Communications with Gestures between Humans and Robots, *Proc. of Twenty-third Annual Meeting of the Cognitive Science Society*, pp. 732-737, 2001.
- [Reeves and Nass, 1996] B. Reeves and C. Nass, *The Media equation*, CSLI Publications, 1996.
- [Scassellati *et al.*, 2000] B. Scassellati, Investigating Models of Social Development Using a Humanoid Robot, *Biorobotics*, MIT Press, 2000.

Qualitative Map Learning Based on Co-visibility of Objects

Takehisa Yairi and Koichi Hori

University of Tokyo

4-6-1 Komaba Meguro-ku Tokyo, Japan

yairi@ai.rcast.u-tokyo.ac.jp

Abstract

This paper proposes a unique map learning method for mobile robots based on the *co-visibility* information of objects i.e., the information on whether two objects are visible at the same time or not from the current position. This method first estimates *empirical* distances among the objects using a simple heuristics – “a pair of objects observed at the same time more frequently is likely to be located more closely together”. Then it computes all the coordinates of the objects by multidimensional scaling (MDS) technique. In the latter part of this paper, it is shown that the proposed method is able to learn *qualitatively* very accurate maps though it uses only such primitive information, and that it is robust against some kinds of object recognition errors.

1 Introduction

Map learning problem of autonomous mobile robots has been a central issue in the field of artificial intelligence as well as robotics, because it contains several important aspects of intelligence such as recognition of environment and acquisition of internal representations. In fact, a variety of map building methods have been developed for a wide range of robots, tasks and environments so far. These methods are traditionally classified [Kortenkamp *et al.*, 1998; Murphy, 2000] in terms of the way of map *representation* into *metric* map building methods [Moravec and Elfes, 1985; Uhlmann *et al.*, 1997], *topological* methods [Mataric, 1992; Zimmer, 1996; Shatkay and Kaelbling, 1997], and hybrids of them [Thrun *et al.*, 1998].

On the other hand, when we turn our attention to a new trend of robot navigation called *qualitative navigation* [Levitt and Lawton, 1990; Schlieder, 1993], we can see there is another way of qualitative map representation that is different from both the metric and topological representations. A most important point of the maps used in this qualitative navigation is that they are not required to be accurate in a *metric* sense as long as they correctly preserve the qualitative spatial relationships (such as circular ordering) of the objects or landmarks in the actual environment.

A challenging problem in this paradigm is to construct autonomously such qualitative maps from *qualitative* observation information robots obtain. A representative approach to this qualitative map learning problem is [Sogo *et al.*, 2001], in which qualitative information of “how object positions are classified to two sets with respect to arbitrary straight lines” is used to construct a map by propagating “three point” constraints.

In this paper, we also propose a map learning method for mobile robots based on qualitative observation information. It uses the information of “co-visibility” or whether two objects are visible or not at the same time from robot’s positions, which is much more primitive than the information used in [Sogo *et al.*, 2001] and the ordering information [Schlieder, 1993].

In this method, co-visibility of two objects is translated into an empirical distance based on a simple heuristics “closely located objects are likely to be seen simultaneously more often than distant objects and vice versa” or “temporal and spatial proximities are approximately equivalent”. Then, positions of all objects are calculated by well-known *multidimensional scaling* (MDS) technique.

A noteworthy feature of this method from a practical viewpoint is that it does not require robots localize their own positions while mapping. Moreover, it is shown that it can learn qualitatively accurate maps without higher level information such as ordering, and is robust against observation errors. We also discuss the validity of the heuristics above from several viewpoints.

2 Problem Definition

2.1 Assumptions on Environment and Robot

we consider a map building task by a mobile robot, in which the robot estimates the positions of objects in the environment by repeated explorations and observations (Figure 1). More specifically, we make the following assumptions about the environment and robot.

Environment

The environment is a closed area containing a finite number of objects. Each object is assigned a unique ID. In addition, it is assumed that all objects are about the same size.

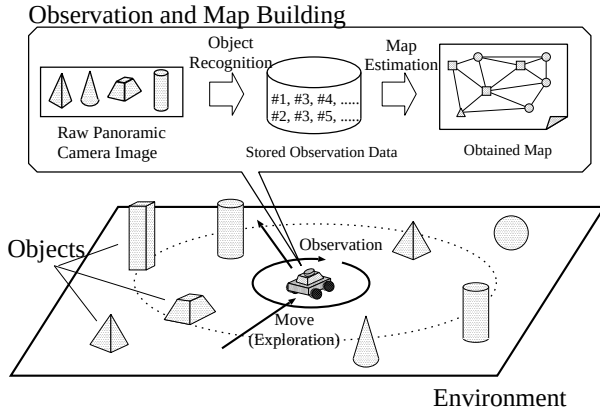


Figure 1: Assumed map building task of a mobile robot (exploration, observation and map estimation)

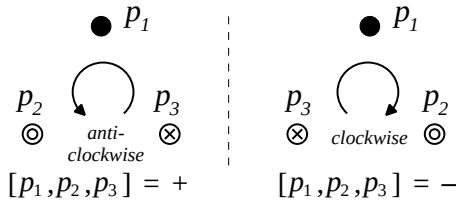


Figure 2: Definition of triangle orientation

Observation

The robot obtains a panoramic camera image at each observation point, and extracts a list of *recognized* or *visible* objects by some image processing and recognition technique. These lists of visible objects are accumulated over time and used to build a map later.

Exploration

The robot explores the environment with some randomness, avoiding collisions with objects.

2.2 Evaluation Criterion for Qualitative Maps

Though a constructed map is represented in the form of numeric coordinates of the objects on 2-D plane as ordinary metric maps, its goodness is measured by the correctness of the *qualitative* spatial relationships rather than by the *metric* accuracy.

To evaluate the qualitative correctness of the obtained maps, we employ the notion of *triangle orientation* [Schlieder, 1993; Sogo et al., 2001] or counter-clockwise order of three points. In short, triangle orientation of three points $[p_1 p_2 p_3]$ in 2-D plane is defined as $+$ when the order p_1, p_2, p_3 yields a counter-clockwise turn, and $-$ otherwise (Figure 2).

When there are N objects in the environment, the number of all possible triangles formed by them becomes ${}_N C_3$. So we define the *orientation error* Err_{ori} of a constructed map as the percentage of triangles with wrong orientations, com-

pared with the *real* configuration of the objects, i.e.,

$$Err_{ori} \equiv \frac{(\text{Number of triangles with wrong orientation})}{{}_N C_3}$$

We use this Err_{ori} as an evaluation criterion of constructed maps in the later simulation study.

2.3 Co-visibility and Distance Between Objects

As previously mentioned, we use a heuristics “a pair of objects observed simultaneously more frequently is likely to be located more closely together” to estimate the distance between them. Though it is hard to prove the validity of this heuristics strictly in general cases, we consider it is approximately appropriate for the following reason.

First, we assume that an object becomes difficult to identify as the distance from the robot increases, because the image size of the object becomes smaller and the chance of occlusion increases.

Given this assumption, the total area size of the region where the robot can observe two objects simultaneously decreases monotonically according to the distance between the objects. For example, consider the case the probability that an object can be observed from the robot at a distance of r is $P_1(r) = e^{-a \cdot r^2}$ (Figure 3). If two object A and B are located at $(-\frac{l}{2}, 0)$ and $(\frac{l}{2}, 0)$ respectively, the probability that both objects are visible from an arbitrary robot position X (x, y) becomes

$$P_2(l, x, y) = P_1(\overline{AX}) \cdot P_1(\overline{BX}) = e^{-\frac{al^2}{2}} \cdot e^{-2a(x^2+y^2)}$$

Then, the expected size of area where both objects are visible simultaneously is

$$S_{covis}(l) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} P_2(l, x, y) dx dy = \frac{\pi}{2a} e^{-\frac{al^2}{2}}$$

In the simplest case that the robot is located at any place in the environment with equal probability, the probability that the two objects are co-visible is expected be proportional to $S_{covis}(l)$, which is a monotonic decreasing function of l .

In section 4, we will discuss this “co-visibility and distance” heuristics in a more general form: “temporal and spatial proximities are approximately equivalent”.

3 Proposed Method

3.1 Outline of CoviMap

Based on the assumptions above, we propose *CoviMap* – a map learning method based on the co-visibility of objects. In this method, the mobile robot obtains approximate positions of objects based on the co-visibility frequencies of them, which are updated repeatedly by explorations and observations. The outline of *CoviMap* is described as below:

1. The robot repeats the following behavior steps and updates the number of observations of each object (n_i), and the number of simultaneous observations of every pair of objects ($n_{i,j}$).
 - (a) The robot moves to the next observation position, avoiding collisions with the objects.

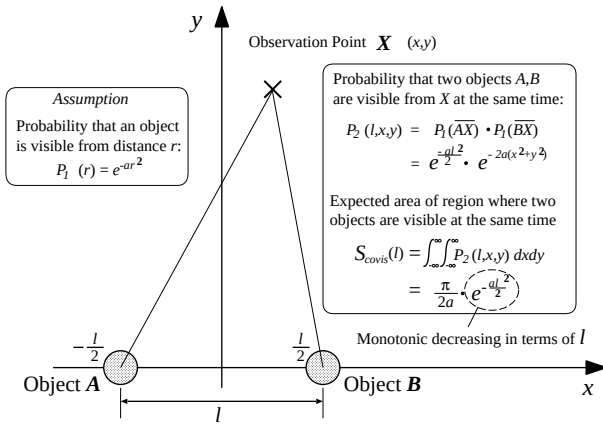


Figure 3: Relationship between co-visibility and distance of two objects in a simplified environment

- (b) It obtains a list of visible objects $\mathcal{L}_o$ from the panoramic image captured at the current position, and updates n_i and $n_{i,j}$ as below:

$$n_i \leftarrow n_i + 1 \quad \text{for each object } i \text{ in } \mathcal{L}_o$$

$$n_{i,j} \leftarrow n_{i,j} + 1 \quad \text{for each pair of } i, j \text{ in } \mathcal{L}_o$$
2. After a specified number of steps, co-visibility frequencies $f_{i,j}$ is computed for each pair of objects based on n_i , n_j , and $n_{i,j}$. Then empirical distance $d'_{i,j}$ of each pair is computed from $f_{i,j}$.
3. The robot obtains the estimated positions of all objects ($\hat{x}_1, \dots, \hat{x}_N$) by applying *Multi-Dimensional Scaling* (MDS) to the distance matrix $\mathbf{D}$ whose (i, j) element is $d'_{i,j}$.

In the remaining of this section, we explain the second and third parts in the above procedure.

3.2 Computation of Co-visibility Frequency and Empirical Distance

Co-visibility frequency $f_{i,j}$ between two objects is defined as follows:

$$f_{i,j} = \frac{n_{i,j}}{n_i + n_j - n_{i,j}}$$

$f_{i,j}$ stands for the conditional probability that two objects are visible at the same time, given that at least one of them is visible. It takes a value between 0 and 1. This definition of $f_{i,j}$ is also known as Jaccard's coefficient.

With the definition of co-visibility frequency $f_{i,j}$, the heuristics introduced in 2.3 can be rewritten as “distance between two objects $d_{i,j}$ monotonically decreases as $f_{i,j}$ increases”. Figure 4 (scattered points) illustrates the actual relationship between the *real* (squared) distance and co-visibility frequency $f_{i,j}$ in the simulation environment in section 4. The result indicates that the heuristics is approximately valid.

Therefore, we introduce a notion of *empirical (squared) distance* $d'_{i,j}$ between an arbitrary pair of objects, which is defined by some monotonic decreasing non-negative function ϕ , i.e.,

$$d'_{i,j} = \phi(f_{i,j})$$

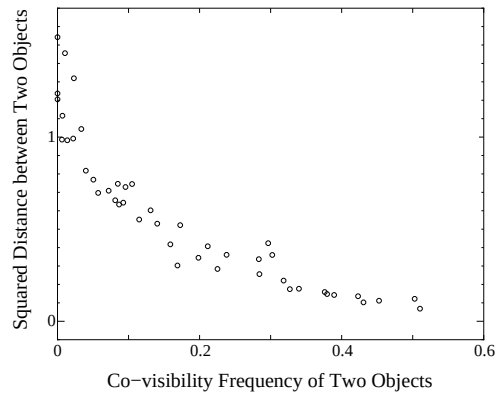


Figure 4: Relationship between squared distance $d_{i,j}^2$ and co-visibility frequency $f_{i,j}$ of objects in the simulation environment

We also define an *empirical distance matrix* $\mathbf{D}$ whose (i, j) element equals to $d'_{i,j}$. $\mathbf{D}$ is a symmetric matrix (i.e., $d'_{i,j} = d'_{j,i}$), and its diagonal components ($d'_{i,i}$) equal to zero.

A possible solution to the problem of deciding a suitable combination of function and parameter values for ϕ is to select the one which minimizes the *stress* value of the MDS results described later. Furthermore, if no appropriate empirical distance functions are provided, non-metric MDS (or ordinal MDS) can be used instead.

Another problem is that $d'_{i,j}$ cannot be computed properly for the pairs whose co-visibility frequencies are zero, because ϕ becomes some constant value. In this case, *CoviMap* utilizes the *triangular inequality constraint* (TIC) to correct the value of $d'_{i,j}$ as follows:

$$d'_{i,j} \leftarrow \min_k (d'_{i,k} + d'_{k,j})$$

3.3 Map Construction Based on Multi-Dimensional Scaling (MDS)

Multi-dimensional scaling (MDS)[Young and Householder, 1938] is a multivariate data analysis technique used to visualize a potentially high-dimensional data structure by mapping it into a relatively low dimensional space[Cox and Cox, 2001]. The purpose of MDS is to find an optimal configuration of objects in a low-dimensional space, when dissimilarities or distances among them are given.

While there are several kinds of MDS methods[Cox and Cox, 2001], *CoviMap* employs *classical scaling*[Young and Householder, 1938] method that is a kind of *metric MDS* to reproduce a map of objects in 2-D plane from the set of empirical distances. In brief, it obtains a set of 2-D coordinates of N objects $\hat{x}_i$ ($i = 1, \dots, N$) by applying *Young-Householder transformation* and *spectral decomposition* to the empirical distance matrix $\mathbf{D}$.

A non-trivial problem of using MDS in *CoviMap* is that there is a fifty percent chance of obtaining the *mirror image* of the expected map. However, if any correct *triangle orientation* of three objects is given, *CoviMap* can detect the mirror map and get the right map by turning it over.

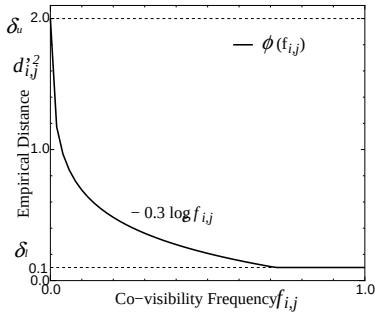


Figure 5: Empirical distance functions $\phi(f_{i,j})$ used in the simulation

4 A Simulation Study

We conducted a simulation study to examine the effectiveness of our method with Cyberbotics' WEBOTS simulator (ver.2.0).

4.1 Settings

Environment and Objects

The environment is a square field of $1.5[\text{m}] \times 1.5[\text{m}]$ containing 5 – 30 objects. Each object is cylinder-shaped (height: 160[mm], diameter: 48[mm]), and is given a unique ID number like *ob0*, *ob1*, $\dots$.

Observation and Object Recognition

The robot has a camera with a resolution of 160×120 pixels, and obtains a panoramic image by rotating and capturing several images at each observation position. We make a simplified assumption that the robot can recognize an object if its image is wider than 10 pixels.

Exploration Strategy

At each observation position, the robot chooses its next moving direction randomly within the range of $\pm\theta_r = 45[\text{deg}]$, and proceeds $l_r = 100[\text{mm}]$ in the direction. In addition, the robot has 8 proximity sensors to detect nearby obstacles (objects and walls) around it, and avoids collisions with them.

Empirical Distance Function

We chose a logarithmic function as the empirical distance function ϕ as below:

$$\phi = \begin{cases} -\alpha \cdot \log_e f_{i,j} & \text{if } \exp\{-\frac{\delta_u}{\alpha}\} \leq f_{i,j} \leq \exp\{-\frac{\delta_l}{\alpha}\} \\ \delta_u & \text{if } 0 \leq f_{i,j} < \exp\{-\frac{\delta_u}{\alpha}\} \\ \delta_l & \text{if } \exp\{-\frac{\delta_l}{\alpha}\} < f_{i,j} \leq 1 \end{cases}$$

where, $\alpha, \delta_l, \delta_u$ are parameters which take positive values. In this simulation, we set them as: $\alpha = 0.3, \delta_l = 0.1, \delta_u = 2.0$, because the stress values of MDS were relatively small with this combination compared with others. Figure 5 shows the shape of this empirical distance function. In addition, *triangular inequality constraint* (TIC) mentioned in 3.2 was used to correct the empirical distances for the object pairs whose co-visibility frequencies were zero.

4.2 Experiment 1 : Varying Number of Objects

First, we examined the basic map building capabilities of *CoviMap* with varying number of objects in the environment

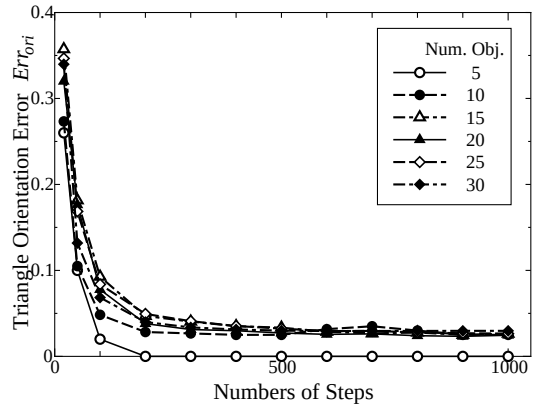


Figure 6: (Experiment 1) Change of average map errors : comparison among various numbers of objects

Steps	5 obj.	10	15	20	25	30
20	26.0	27.3	35.7	32.0	34.7	34.0
100	2.00	4.83	9.28	7.77	8.37	6.84
500	0.00	2.50	3.34	2.72	3.24	3.00
1000	0.00	2.50	2.64	2.46	2.64	2.97

Table 1: Change of average map errors [%] with 6 different numbers of objects

from 5 to 30. In each case, the robot repeated the observation and exploration steps for 1000 times, and built a map eachtime the number of steps reached one of specified numbers (20,50,100,...,1000). For each case, we prepared 5 lay-outs of objects which are randomly generated and averaged the orientation errors.

Figure 6 and Table 1 show how the average orientation error Err_{ori} changes according to the number of steps for each number of objects. As can be seen from this, Err_{ori} converges to a narrow range (approximately between 2.5[%] and 3.0[%]) after several hundreds of steps, when the number of objects varies from 10 to 30. This means that *CoviMap* is not affected drastically by the increase in the number of objects.

Figure 7 illustrates an example of real configuration which contains 30 objects, while Figure 8 illustrates a map constructed by *CoviMap* after 500 steps. In this case, out of total 4060 triangles, the number of triangles whose orientations are inconsistent between the real map and the constructed map is 98 ($Err_{ori} = 2.4[\%]$). Interestingly, if we consider only the triangles whose largest angles are smaller than $170[\text{deg}]$ in the real map (the number of such triangles is 3835), the number of inconsistent triangles falls to 27 ($Err_{ori} = 0.70[\%]$). This means our method is able to acquire very accurate qualitative layouts of the objects, unless the objects are almost on a line. Dotted lines in the figures represent *Delaunay graphs* of the configurations. A comparison of them also tells that the constructed map is *qualitatively* very accurate.

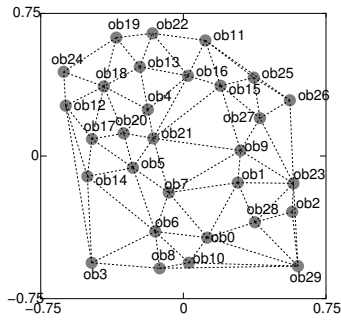


Figure 7: (Experiment 1) An example of object configuration with 30 objects. dotted lines are edges of *Delaunay* graph

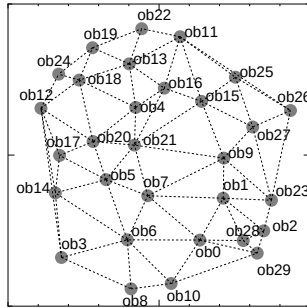


Figure 8: (Experiment 1) Constructed map after 500 steps ($Err_{ori} = 2.4\%$ for all triangles)

4.3 Experiment 2 : Robustness Against Object Recognition Errors

In the previous experiment, it was assumed that there are no errors in the object recognition process. In the real environment, however, it is almost inevitable to fail in recognizing objects occasionally, due to various uncertainties. Therefore, we examined how the recognition errors affect the quality of maps constructed by *CoviMap*.

More specifically, we considered two kinds of recognition errors – *non-recognition* and *mis-recognition*. These errors were artificially generated as follows:

Non-recognition : Randomly chosen elements in $\mathcal{L}_o$ are removed

Mis-recognition : Randomly chosen elements in $\mathcal{L}_o$ are replaced with other object IDs.

In this experiment, we fixed the number of objects to 15 and conducted 5 trials for each of 5 different object configurations.

Result with Non-recognition Error

Figure 9 shows the average *orientation* errors of the constructed maps when the percentage of non-recognition error is set to 0.0 (no errors), 3.0, 10, 20[%] respectively. In an early stage of learning, i.e., with fewer steps, the map error Err_{ori} becomes larger according to the magnitude of the non-recognition error. However, the difference becomes

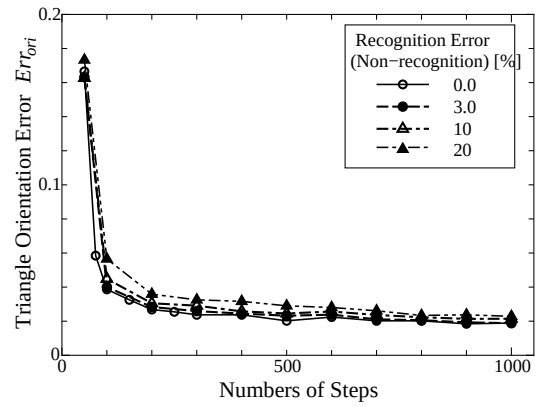


Figure 9: (Experiment 2) Change of average map errors : comparison among various *non-recognition* rates

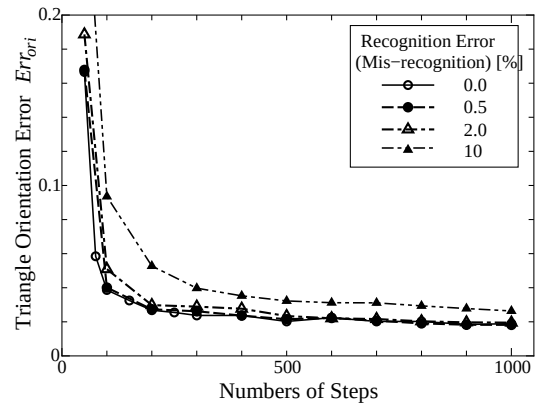


Figure 10: (Experiment 2) Change of average map errors : comparison of various *mis-recognition* rates

smaller and almost negligible as the number of steps increases.

Result with Mis-recognition Error

Next, Figure 10 shows the results when mis-recognition error is set to 0.0, 0.5, 2, 10[%] respectively. Not surprisingly, the influence of this kind of error on the map error Err_{ori} is relatively larger than the non-recognition error.

In both cases, however, we can conclude *CoviMap* is robust against those recognition errors, in that the map error steadily decreases according to the increase of the number of steps.

5 Related Works

As mentioned before, the central idea of our map building method is a simple heuristics – “a pair of objects observed simultaneously more frequently is likely to be located more closely”, which is used to estimate the distances and qualitative configuration of objects from the co-visibility information of them. In this section, we discuss the meaning of this heuristics from other viewpoints, especially, considering several related works in other fields.

First, as *co-visibility* of objects can be regarded as *co-occurrence* of events that objects are visible, a lot of studies on learning of behavior and models based on the *co-occurrence* information have been made. The most basic principle related to this issue is *Hebb's rule* which postulates that if two events co-occur often, the connection of them should be strengthened. Many recently developed *reinforcement learning* methods such as Q-learning[Watkins and Dayan, 1992] are also considered to be based on the *co-occurrence* or *temporal proximity* between behavior and reward events. Interestingly, there is another similarity between reinforcement learning and our *CoviMap*. That is to say, the former learns a complicated behavior system from discrete events of *rewards*, while the latter learns a qualitatively accurate map from discrete events of *co-visibility* among objects.

The heuristics above can also be regarded as a special case of more general one – “temporal and spatial proximities are approximately equivalent in many environments”. In the research of human spatial memory, it has been reported that humans often rely on the *temporal* proximity rather than the *spatial* proximity to memorize spatial structures of environments[Curiel and Radvansky, 1998]. This might suggest that the principles and results of *CoviMap* presented in this paper are helpful to understand some aspects of human spatial memory and cognitive mapping process, although we doubt that an MDS-like algorithm exists in human brain.

6 Conclusion

In this paper, we proposed a unique map building method for mobile robots named *CoviMap* which is based on the *co-visibility* information of objects in the environment. It can learn *qualitatively* accurate maps from such primitive information using a simple heuristics “temporal and spatial proximities are approximately equivalent” and multidimensional scaling (MDS) technique. It was also shown by the simulation results that *CoviMap* is applicable to large environments containing dozens of objects, and robust against observation errors such as non-recognition and mis-recognition.

A most important theoretical issue to be studied is to validate and generalize the heuristics on the *equivalence of temporal and spatial proximities*. Especially, we need to investigate how our method will be affected if the camera image is not panoramic but more restricted, or the object recognition rate is not isotropic but dependent on the direction. On the other hand, future works from a practical point of view will include experiment with real robots, integration of *CoviMap* with qualitative navigation[Levitt and Lawton, 1990], and extension or replacement[Faloutsos and Lin, 1995] of MDS.

References

- [Cox and Cox, 2001] T. Cox and M. Cox. *Multidimensional Scaling*. Chapman & Hall/Crc, 2001.
- [Curiel and Radvansky, 1998] J. M. Curiel and G. A. Radvansky. Mental organization of maps. *Journal of Experimental Psychology*, 24(1):202–214, 1998.
- [Faloutsos and Lin, 1995] C. Faloutsos and K. Lin. FastMap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, pages 163–174, 1995.
- [Kortenkamp et al., 1998] D. Kortenkamp, R. Bonasso, and R. Murphy, editors. *Artificial Intelligence and Mobile Robots : Case Studies of Successful Robot Systems*. MIT Press, 1998.
- [Levitt and Lawton, 1990] T. Levitt and D. Lawton. Qualitative navigation for mobile robots. *Artificial Intelligence*, 44(3):305, 361 1990.
- [Mataric, 1992] M. Mataric. Integration of representation into goal-driven behavior-based robots. *IEEE Transactions on Robotics and Automation*, 8(3):304–312, 1992.
- [Moravec and Elfes, 1985] P. Moravec and A. Elfes. High resolution maps from wide angle sonar. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 116–121, 1985.
- [Murphy, 2000] R. Murphy. *Introduction To AI Robotics*. MIT Press, 2000.
- [Schlieder, 1993] C. Schlieder. Representing visible locations for qualitative navigation. In N. P. Carret’e and M. G. Singh, editors, *Qualitative Reasoning and Decision Technologies*, pages 523–532. 1993.
- [Shatkay and Kaelbling, 1997] H. Shatkay and L. Kaelbling. Learning topological maps with weak local odometric information. In *Proc. of IJCAI-97*, pages 920–927, 1997.
- [Sogo et al., 2001] T. Sogo, H. Ishiguro, and T. Ishida. Acquisition and propagation of spatial constraints based on qualitative information. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(3):268–278, 2001.
- [Thrun et al., 1998] S. Thrun, J. Gutmann, D. Fox, W. Burgard, and B. Kuipers. Integrating topological and metric maps for mobile robot navigation: A statistical approach. In *Proc. of AAAI-98*, pages 989–995, 1998.
- [Uhlmann et al., 1997] J. K. Uhlmann, S. J. Julier, and M. Csorba. Nondivergent simultaneous map-building and localization using covariance intersection. In *Navigation and Control Technologies for Unmanned Systems II*, pages 2–11. 1997.
- [Watkins and Dayan, 1992] C. Watkins and P. Dayan. Technical note: Q-learning. *Machine Learning*, 8(3-4):279–292, 1992.
- [Young and Householder, 1938] G. Young and A. Householder. Discussion of a set of points in terms of their mutual distances. *Psychometrika*, 3:19–22, 1938.
- [Zimmer, 1996] U. R. Zimmer. Robust world-modelling and navigation in a real world. *Neurocomputing*, 13:247–260, 1996.

CONSTRAINTS

Propagate the Right Thing: How Preferences Can Speed-Up Constraint Solving

Christian Bessière

Anaïs Fabre*

LIRMM-CNRS (UMR 5506)

161, rue Ada

F-34392 Montpellier Cedex 5

{bessiere,fabre}@lirmm.fr

Ulrich Junker

ILOG S.A.

1681, route des Dolines

F-06560 Valbonne

ujunker@ilog.fr

Abstract

We present an algorithm Pref-AC that limits arc consistency (AC) to the preferred choices of a tree search procedure and that makes constraint solving more efficient without changing the pruning and shape of the search tree. Arc consistency thus becomes more scalable and usable for many real-world constraint satisfaction problems such as configuration and scheduling. Moreover, Pref-AC directly computes a preferred solution for tree-like constraint satisfaction problems.

1 Introduction

In the last two decades, considerable research effort in AI has been spent on studying the complexity of reasoning and problem solving, and the identification of tractable cases has become an important goal of this research. However, even if a problem can be solved by a polynomial algorithm, or if search effort can be reduced by polynomial algorithms, this might not be sufficient for real-world AI systems that face challenges such as interactivity and real-time. We give some examples from constraint satisfaction, where achieving arc consistency might be too costly, although it is tractable:

1. Web-based configuration systems must provide a response in a few seconds while serving multiple users. Although arc consistency (AC) is an appropriate technique to handle compatibility constraints, it will be too expensive in presence of large product catalogs.
2. Constraint-based approaches to problems such as scheduling, vehicle routing, and personnel planning often only maintain bound consistency in constraints involving time or other variables with large domains. As a consequence, complex rules on breaks, rest-times, and days-off provide only a poor propagation, which could be improved by arc consistency.

These examples illustrate that domain reduction does not develop its full power in many real-world applications of constraint programming. Pruning power and, indirectly, solution quality are traded against short response times. In certain circumstances, such a trade-off can be avoided if the system focuses on the right deductions and computations. In this paper,

*A. Fabre's work has been carried out during a stay at ILOG.

we argue that preferences are one possible way for achieving this. As stated by Jon Doyle [Doyle, 2002], preferences can play different roles and need not only represent desiderata or user preferences. Preferences can also control reasoning meaning that only preferred inferences and domain reductions are made. Non-interesting deductions are left apart, but may become preferred if additional information is added.

We can easily apply this idea to typical backtrack tree search algorithms that maintain arc consistency when solving a constraint satisfaction problem. The crucial question is whether such an algorithm needs to achieve full arc consistency and to remove all values that are not in the maximal arc-consistent domains. In [Schiex *et al.*, 1996], it has been shown that the fails of the algorithm can be preserved if it simply constructs some arc-consistent domains, thus reducing the computation effort in average. In this paper, we show that the search algorithm can preserve its search strategy and the overall pruning if 1. the search algorithm uses a variable and value ordering heuristics that can be described by (static) preference ordering between variables and values and 2. we construct some arc-consistent domains that contain the preferred values. The first condition is, for example, met by configuration problems [Junker and Mailharro, 2003]. The second condition needs some careful adaption of the definition of arc consistency, which will be elaborated in this paper.

We can thus cut down the propagation effort in each search node without changing the search tree, which leads to an overall gain if the search tree will not be explored completely. This happens if we are interested in one solution, the best solution in a given time frame, or preferred solutions as defined in [Junker, 2002]. Arc consistency thus becomes more scalable and less dependent on the size of domains, which opens exciting perspectives for large-scale constraint programming.

We first introduce preferences (Section 2), use them to define a directed constraint graph (Section 3), and discuss arc consistency for preferred values (Sections 4 and 5). After introducing preferred supports (Section 6), we present the algorithm Pref-AC (Section 7).

2 Preferred Solutions

In this paper, we consider constraint satisfaction problems of the following form. Let $\mathcal{X}$ be a set of variables and $\mathcal{D}$ be a set of values. A constraint c of arity k_c has a sequence of variables $X(c) = \{x_1(c), \dots, x_{k_c}(c)\}$ from $\mathcal{X}$ and a relation R_c

that is a set of tuples from $\mathcal{D}^{k_c}$. Let $\mathcal{C}$ be a set of constraints. The triple $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ is then called a *constraint network*.

A *solution* of a constraint network is a mapping ν of the variables $\mathcal{X}$ to the values in $\mathcal{D}$ such that each constraint is satisfied, i.e., $(\nu(x_1(c)), \dots, \nu(x_{k_c}(c))) \in R_c$ for all constraints c . The *constraint satisfaction problem (CSP)* involves finding a solution of a constraint network.

Alternatively, we can represent the tuples by sets of assignments of the form $x = v$ between a variable x and a value v : if t is a tuple of the form $(v_1, \dots, v_{k_c})$ in R_c then let $\hat{t}$ be the set of assignments $\{x_1 = v_1, \dots, x_{k_c} = v_{k_c}\}$. Furthermore, let $\hat{R}_c$ be the set of all $\hat{t}$ such that t is in R_c . This representation will facilitate definitions and permits a straightforward logical characterization of a constraint c :

$$\bigvee_{t \in R_c} \bigwedge_{(x=v) \in \hat{t}} (x = v) \quad (1)$$

We further suppose that the following kinds of preferences are given. For each variable x , we introduce a strict partial order $\prec_x \subseteq \mathcal{D} \times \mathcal{D}$ among the possible values for x . This order can represent user preferences as occurring in configuration problems (cf. [Junker and Mailharro, 2003]). For example, the user might prefer a red car to a white car. Projecting preferences on criteria to decision variables [Junker, 2002] also produces such an order (e.g. if price is minimized then try cheaper choices first). Furthermore, we consider a strict partial order $\prec_{\mathcal{X}} \subseteq \mathcal{X} \times \mathcal{X}$ between variables. For example, we may state that choosing a color is more important than choosing a seat material and should therefore be done first. In the sequel, we suppose that the search procedure follows these preferences as described in [Junker and Mailharro, 2003] and always chooses a $\prec_{\mathcal{X}}$ -best variable and a $\prec_x$ -best value for x . These preferences are given statically and thus correspond to a static variable and value ordering heuristics, which are usually sufficient for configuration problems.

We now use the preferences to define a preferred solution. First of all we choose a linearization of the orders $\prec_x$ and $\prec_{\mathcal{X}}$ in form of total orders $<_x$ and $<_{\mathcal{X}}$ that are supersets of the strict partial orders. Then we consider a ranking $\chi(1), \dots, \chi(n)$ of the variables in $\mathcal{X}$ that corresponds to the order $<_{\mathcal{X}}$, i.e., $\chi(i) <_{\mathcal{X}} \chi(j)$ iff $i < j$. We then consider two solutions ν_1 and ν_2 and compare them lexicographically

$$\begin{aligned} (\nu_1(\chi(1)), \dots, \nu_1(\chi(n))) &<_{lex} (\nu_2(\chi(1)), \dots, \nu_2(\chi(n))) \\ &\text{iff} \\ \exists k : \nu_1(\chi(k)) &<_{\chi(k)} \nu_2(\chi(k)) \text{ and} \\ \nu_1(\chi(i)) = \nu_2(\chi(i)) &\text{ for all } i = 1, \dots, k-1 \end{aligned} \quad (2)$$

Definition 1 A solution ν of a constraint network $\mathcal{P} := (\mathcal{X}, \mathcal{D}, \mathcal{C})$ is a preferred solution of $\mathcal{P}$ iff there exist linearizations $<_{\mathcal{X}}$ of $\prec_{\mathcal{X}}$ and $<_x$ of $\prec_x$ for each $x \in \mathcal{X}$ s.t. ν is the best solution of $\mathcal{P}$ w.r.t. the lexicographical order defined by $<_{\mathcal{X}}$ and $<_x$.

These preferred solutions correspond to the extreme solutions in [Junker, 2002] that are obtained if all variables are criteria. If the search procedure follows the preferences then its first solution is a preferred solution. Hence, the notion of a preferred solution helps us to forecast the first solution in certain cases.

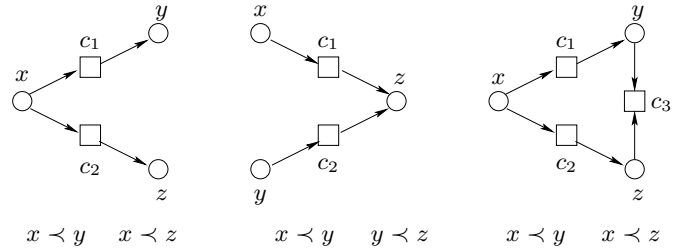


Figure 1: Directed constraint graphs.

3 Preference-based Constraint Graph

In order to find a preferred solution, the search procedure chooses values for more important variables first. We will see later that, in some cases, we can anticipate a preferred solution and construct it by a constraint propagation procedure if this procedure does not only follow the constraints, but also the order among the variables used by the search procedure.

In order to illustrate this, consider the bipartite constraint graph that connects the constraints with their variables. Suppose that a variable x is connected via a constraint c to a less important variable y . Search will then assign a value to x and cause a domain reduction of y . We can thus say that the preferences $\prec_{\mathcal{X}}$ among variables impose an order on the constraint graph. For example, we will say that the arc between x and c leads from x to c and the arc between y and c leads from c to y .

We generalize this idea for arbitrary constraints and partial orders. For each constraint c , we consider the $\prec_{\mathcal{X}}$ -best variables in $X(c)$ and call them input variables. The other variables are called output variables. Arcs lead from input variables to a constraint and from there to the output variables.

Definition 2 Let $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ be a constraint network and $\prec_{\mathcal{X}}$ be a strict partial order between the variables $\mathcal{X}$. A variable x of a constraint c is called input variable of c iff there is no other variable y of c s.t. $y \prec_{\mathcal{X}} x$. A variable x of c is called output variable of c iff it is not an input variable of c . The directed constraint graph of the CSP is a bipartite graph having the nodes $\mathcal{X} \cup \mathcal{C}$ and the set of arcs (x, c) s.t. x is an input variable of c and (c, y) s.t. y is an output variable of c .

Each constraint can have several input and several output variables (cf. figure 1). If its variables are not comparable at all, then the constraint has no output variable, and it is a sink of the directed graph. However, each constraint has at least one input variable. As a consequence, no constraint can be a source of the directed graph. In general, the graph is not guaranteed to be acyclic, but it satisfies the following properties which are needed for our proofs: 1. All $\prec_{\mathcal{X}}$ -best variables of the set $\mathcal{X}$ are sources of the constraint graph since they cannot be an output variable of any constraint. However there can be sources that are not $\prec_{\mathcal{X}}$ -best variables. 2. If χ is a ranking of the variables $\mathcal{X}$ that is a linearization of $\prec_{\mathcal{X}}$ then each output variable of each constraint is preceded by at least one input variable of the same constraint in the ranking.

4 Arc Consistency

A typical systematic search procedure for solving CSPs proceeds as follows. In each search node, it picks an assignment $x = v$ for such a variable and creates two successor nodes, one for $x = v$ and one for $x \neq v$. Throughout the paper, we assume that a variable x is eliminated from the CSP of a successor node by a preprocessing step iff it has only a single possible value. Usually, the search procedure will not select an arbitrary assignment from the set $\mathcal{A}$ of all assignments, but tries to anticipate certain cases where the sub-tree of an assignment does not contain a solution. If such an assignment is detected it is eliminated from $\mathcal{A}$. Hence, the search procedure maintains a set $A \subseteq \mathcal{A}$ of possible assignments and considers only elements of A . Each solution can be characterized by a subset of A that assigns exactly one value to each variable and that satisfies all constraints. As a consequence, if a variable x has no assignment in A then no solution exists and the search will backtrack. In the sequel, we write A_x for the set $\{v \in \mathcal{D} \mid (x = v) \in A\}$ of possible values for x in A .

Assignments can be eliminated from A if they lack a support on a constraint:

Definition 3 A set of assignments S is a support for $x = v$ in c iff 1. $S \cup \{x = v\}$ is an element of $\hat{R}_c$ and 2. $(x = v) \notin S$.

Let $S_{x=v}^c$ be the set of supports for $x = v$ in c . In this paper, we are interested in search procedures that maintain arc consistency, i.e. that work with a set A where all assignments have supports S that are subsets of A and this on all relevant constraints:

Definition 4 Let $\mathcal{A}$ be a set of assignments. A subset A of $\mathcal{A}$ is an arc-consistent set of $\mathcal{A}$ iff 1. A_x is non empty for any $x \in \mathcal{X}$ and 2. for all $(x = v) \in A$ and all constraints c of x , there exists a support S for $(x = v)$ in c such that S is a subset of A .

If an arc-consistent set A contains a single assignment for each variable then A corresponds to a single solution. If no arc-consistent set exists then the CSP has no solution. Otherwise, there is a unique maximal arc-consistent set, which is a superset of all arc-consistent sets, including the solutions.

Alternatively, we can consider a set Δ of some assignments for which we know that they do not belong to any solution:

$$\text{if all supports } S \text{ in } c \text{ for } (x = v) \text{ have an element } (y = w) \text{ in } \Delta \text{ then } (x = v) \text{ is in } \Delta. \quad (3)$$

This can easily be translated to a reflexive and monotonic operator that eliminates additional assignments:

$$\rho_{\mathcal{A}}(\Delta) := \Delta \cup \{(x = v) \in \mathcal{A} \mid \forall S \in S_{x=v}^c : S \not\subseteq \mathcal{A} - \Delta\} \quad (4)$$

Fixed-point theory then tells us that the transitive closure of this operator is the unique minimal set Δ^* satisfying equation 3. As a consequence, the set A^* of all assignments that are not eliminated (i.e., $A^* := \mathcal{A} - \Delta^*$) is the unique maximal set satisfying item 2 of def. 4. If each variable has an assignment in A^* then A^* is the maximal arc-consistent assignment. If some variable has no assignment in A^* no arc-consistent assignment exists. In this case, no solution exists.

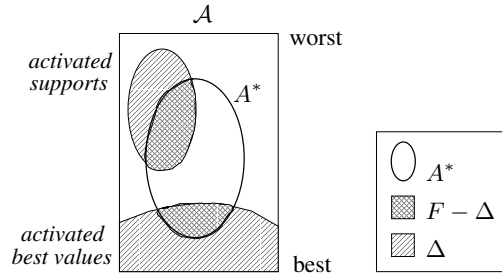


Figure 2: Activating best values and their supports.

5 Arc Consistency for Preferred Values

Maintaining arc consistency prunes the search tree as follows:

- P1 if the current CSP is not arc-consistent then a fail occurs.
- P2 if a value v is not arc-consistent for a variable x then the search procedure will not assign v to x inside the current subtree.

In general, it is desirable to eliminate as many values as possible and to determine A^* by applying an algorithm of the AC-family. The worst-case complexity is $O(|\mathcal{C}| \cdot |\mathcal{D}|^2)$ for binary constraints. If domains contain thousands or more elements as is the case for typical scheduling problems and for configuration problems with large catalogs, then maintaining arc consistency is no longer feasible in practice.

Interestingly, it is not necessary to eliminate all assignments of $\mathcal{A} - A^*$. [Schiex *et al.*, 1996] try to construct a small arc-consistent set A . If this succeeds then pruning rule P1 cannot be triggered. This can lead to saving time in search nodes that are arc-consistent. We now want to extend this policy to pruning rule P2 in the case that the original search procedure uses the given preferences between variables and values in order to select an assignment. In each search state, the search procedure will pick a $\prec_{\mathcal{X}}$ -best variable and try out its possible values starting with the best value v^* . If $x = v^*$ is not in the maximal arc-consistent set A^* then we must ensure that it will be eliminated when constructing the subset A . However, if $x = v^*$ is in A^* then we must ensure that it will be in the subset A .

Instead of doing a pure elimination process, we interleave construction and elimination steps and determine a set F of *activated assignments*, called *focus*, and a set Δ of *eliminated assignments*. Both sets will grow during the process and our purpose is to obtain $F - \Delta$ as arc-consistent set. This set is sufficient for our search procedure if some properties are satisfied: 1. If x is a $\prec_{\mathcal{X}}$ -best variable and v is a $\prec_x$ -best among the non-eliminated values of x then the search procedure might pick $x = v$ as next assignment. Hence, we have to activate it. 2. If an activated assignment does not have a support on a constraint c then it has to be eliminated. 3. The supports for the activated, but non-eliminated assignments must only contain activated and non-eliminated elements. 4. We need to guarantee that $F - \Delta$ contains at least one value per variable if an arc-consistent set exists. Since all variables can be reached from the sources via constraints, we activate best values for all sources of the directed constraint graph.

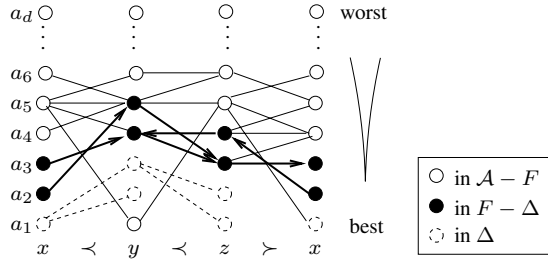


Figure 3: Example for a stable activation.

Definition 5 Let $\mathcal{A}$ be a set of initial assignments and F, Δ be subsets of $\mathcal{A}$ s.t. $\Delta \subseteq F$. We say that (F, Δ) is an $\prec$ -activation of $\mathcal{A}$ iff the following properties hold:

1. If x is a source and $(x = w) \in \Delta$ for all $w \in \mathcal{A}$ with $w \prec_x v$ then $x = v$ is in F .
2. If $(x = v) \in F$, $x \in X(c)$, and c has no support S for $x = v$ s.t. $S \subseteq \mathcal{A} - \Delta$ then $(x = v) \in \Delta$.

An $\prec$ -activation (F, Δ) is stable iff all $(x = v) \in F - \Delta$ have a support $S \subseteq F - \Delta$ on all constraints c containing x .

Proposition 1 Let $\mathcal{A}$ be a set of initial assignments and $\Delta^* := \rho_{\mathcal{A}}^*(\emptyset)$. Then $(\mathcal{A}, \Delta^*)$ is a stable activation.

Hence stable activations always exist and coincide with the maximal arc-consistent set in the worst case. Maintaining stable activations is sufficient to achieve pruning rules P1 and P2 for search procedures that always pick the best elements of $\mathcal{A} - \Delta$ (cf. figure 2):

Proposition 2 Let (F, Δ) be a stable $\prec$ -activation of $\mathcal{A}$. If there is no arc-consistent subset of $\mathcal{A}$ then there exists a variable x s.t. $F - \Delta$ does not contain an assignment for x . If there is an arc-consistent subset of $\mathcal{A}$ and A^* is the maximal arc-consistent subset of $\mathcal{A}$ then the following properties hold:

1. $F - \Delta$ is an arc-consistent subset of $\mathcal{A}$.
2. If x is a $\prec_X$ -best variable and $x = v$ is a $\prec_x$ -best assignment for x in A^* then $x = v$ is in $F - \Delta$.
3. If x is a $\prec_X$ -best variable, $x = v$ is a $\prec_x$ -best assignment for x in A^* , and $w \prec_x v$ then $x = w$ is in Δ .

If the search procedure chooses a $\prec$ -best assignment for a best variable in each step, then it will explore the same search tree independently of whether it maintains full arc consistency or whether it maintains stable activations. Since the first solution found by the standard search procedure is a preferred solution, we will still obtain a preferred solution first if we maintain stable activations.

We now discuss how to actually construct stable $\prec$ -activations. Initially, we activate the best assignments in $\mathcal{A}$ for all relevant variables by a monotonic operator ϕ :

$$\phi_{\mathcal{A}}(\Delta) := \{(x = v) \in \mathcal{A} \mid x \text{ is a source, } \forall w \in \mathcal{A} \text{ s.t. } w \prec_x v : (x = w) \in \Delta\} \quad (5)$$

If an activated assignment is not supported by a constraint it will be eliminated and replaced by the next best assignments until we reach a set of assignments that have a support in

each constraint. Unsupported assignments are eliminated by an operator δ , which restricts ρ to the focus F :

$$\delta_{\mathcal{A}}(F, \Delta) := \{(x = v) \in F \mid \exists c : \forall S \in \mathcal{S}_{x=v}^c : S \not\subseteq \mathcal{A} - \Delta\} \quad (6)$$

We can iterate both operators if we combine them into an operator σ that maps a pair of assignment sets to a pair of assignment sets s.t. $\sigma_{\mathcal{A}}((F, \Delta)) := (F \cup \phi_{\mathcal{A}}(\Delta), \Delta \cup \delta_{\mathcal{A}}(F, \Delta))$. We easily obtain a lattice on these pairs if we define union and intersection of two pairs by applying these operations to both elements. The operator $\sigma_{\mathcal{A}}$ is monotonic and reflexive in this lattice and its transitive closure $\sigma_{\mathcal{A}}^*((F, \Delta))$ is the smallest $\prec$ -activation (F^*, Δ^*) that extends (F, Δ) .

Next we discuss how an activation containing unsupported assignments can be extended to a stable one. We say that assignment $x = v$ is *unsupported* on c in (F, Δ) if c does not have a support S for $x = v$ s.t. $S \subseteq F - \Delta$. If no such assignment exists, we have found a stable activation. Otherwise, we pick an unsupported assignment, choose a support S for it, and activate the elements of this support. After this, we apply operator $\sigma_{\mathcal{A}}$ to produce a new activation:

Proposition 3 Let (F, Δ) be an $\prec$ -activation for $\mathcal{A}$. Suppose $x = v$ is unsupported on c in (F, Δ) . Then c has a support S for $x = v$ s.t. $S \subseteq \mathcal{A} - \Delta$ and $S \not\subseteq F - \Delta$ and $\sigma_{\mathcal{A}}^*((F \cup S, \Delta))$ is an activation that supports $x = v$ on c .

We can thus construct a stable activation as follows:

Definition 6 Let $\mathcal{A}$ be a set of initial assignments. Let $S_1, \dots, S_m$ be subsets of $\mathcal{A}$. We then define $(F_0, \Delta_0) := \sigma_{\mathcal{A}}^*((\emptyset, \emptyset))$ and $(F_i, \Delta_i) := \sigma_{\mathcal{A}}^*((F_{i-1} \cup S_i, \Delta_{i-1}))$ for $i = 1, \dots, m$. The sequence $S_1, \dots, S_m$ is called an *activation sequence* iff 1. $S_i \not\subseteq F_{i-1}$, 2. each S_i is a support on a constraint c for an assignment $(x = v)$ in $F_{i-1} - \Delta_{i-1}$, and 3. (F_m, Δ_m) is stable.

Since F_i has more elements than F_{i-1} and $(\mathcal{A}, \rho_{\mathcal{A}}^*(\emptyset))$ is a stable activation, we can show the following result, which permits the construction of stable activations:

Proposition 4 Let $\mathcal{A}$ be a set of initial assignments. There exists an activation sequence $S_1, \dots, S_m$ for $\mathcal{A}$.

Figure 3 gives an idea of the possible gains. Once the dashed assignments have been eliminated, we activate only two assignments per variable, i.e. six in total, whereas an AC-algorithm would activate each of the $3 \cdot |\mathcal{D}|$ assignments in $\mathcal{A}$. Hence, we obtain significant gains if domains are large and no elimination is required. The elimination of an assignment still costs $O(|\mathcal{D}|)$ checks.

6 Preferred Supports

Preferences between values and variables allow us to define preferred supports. A support $S \subseteq \mathcal{A} - \Delta$ for an assignment $x = v$ in constraint c can be considered a solution of the CSP $(X(c), \mathcal{D}, \{c, x = v\} \cup \{y \neq w \mid (y = w) \in \Delta\})$. We say that S is a *preferred support* for $x = v$ on c iff S is a preferred solution of this CSP. An assignment can have several preferred supports. In case of binary constraint networks, preferred supports are unique if all domain orders $\prec_x$ are total. In general, preferred supports are unique if additionally the order $\prec_X$ is total as well.

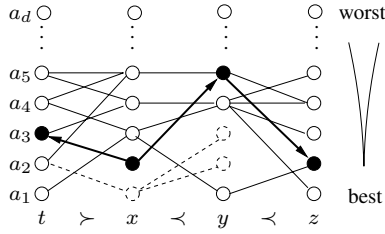


Figure 4: A preferred solution as stable activation.

Given a preferred solution, we cannot guarantee that each assignment has a preferred support in all constraints. However, if certain assignments in a solution S have preferred supports then S is preferred solution:

Proposition 5 *Let S be a solution of $\mathcal{P}$. Suppose that $\prec_{\mathcal{X}}$ and all $\prec_x$ are strict total orders. If each assignment $x = v$ to a source x of the direct constraint graph is a best assignment for x in the unique maximal arc-consistent set A^* and all activated assignments to input variables of a constraint c have a preferred support on c in S then S is a preferred solution of $\mathcal{P}$.*

When constructing a stable activation, we choose preferred supports since this may result in a preferred solution:

Proposition 6 *Suppose that $\prec_{\mathcal{X}}$ and all $\prec_x$ are strict total orders. Let $S_1, \dots, S_m$ be an activation sequence and let (F_i, Δ_i) be defined as in def. 6. If 1. each S_i is a preferred support for some $x = v$ on some constraint c s.t. x is an input variable of c or $x = v$ is unsupported on c in (F_{i-1}, Δ_{i-1}) and 2. $F_m - \Delta_m$ is a solution of $\mathcal{P}$ then $F_m - \Delta_m$ is a preferred solution of $\mathcal{P}$.*

The first condition in proposition 6 can be satisfied by always activating preferred supports. The second condition can be met by CSPs of a special structure. For example, consider a binary constraint network such that its directed constraint graph forms a tree (cf. figure 4). In this case, we only activate supports for assignments to input variables, but not for assignments to output variables, since each variable can only be an output variable of a single constraint. This result is, for example, interesting for certain car configuration problems and permits us to efficiently take into account certain user preferences.

Preferred solutions are also obtained in trivial cases:

Proposition 7 *Suppose that $\prec_{\mathcal{X}}$ and all $\prec_x$ are strict total orders. If there is a preferred solution containing all best values of all variables in $\mathcal{A}$ then it is equal to all activations produced by preferred supports.*

7 Algorithm Pref-AC

In this Section, we present Pref-AC, an algorithm for computing an arc-consistent set A from a binary constraint network $(\mathcal{X}, \mathcal{D}, \mathcal{C})$. This algorithm activates supports for unsupported elements and meets the requirements of definitions 5 and 6, and proposition 6. An algorithm for non-binary constraints can be derived from this one. Pref-AC follows the principles of lazy arc consistency [Schiex *et al.*, 1996] in order to

build an arc-consistent set of assignments that is not necessarily maximal. We base Pref-AC on AC-6 [Bessière, 1994] for keeping the discussion simple, although we could exploit the bi-directionality of supports [Bessière *et al.*, 1999].

AC-6 assigns an ordering of the values in the domain of every variable x_i , checks one support (the *first* one or *smallest* one with respect to the ordering) for each assignment $(x_i = a)$ on each constraint $c(x_i, x_j)$ to prove that $(x_i = a)$ is currently viable. When $(x_j = b)$ is found as the smallest support of $(x_i = a)$ on $c(x_i, x_j)$, $(x_i = a)$ is added to $CS(x_j, b)$, the list of assignments currently having $(x_j = b)$ as smallest support. If b is removed from the domain of x_j then AC-6 looks for the next support in the domain of x_j for each assignment $x_i = a$ in $CS(x_j, b)$.

Pref-AC uses the domain order $\prec_{x_i}$, a chosen linearization of $\prec_{x_i}$, when looking for supports in the domain of a variable x_i . Furthermore, Pref-AC seeks supports only for elements of A , the current set of activated and non-eliminated values (i.e., $F - \Delta$). Pref-AC uses following data structure:

- Each variable x_i has a fixed initial domain $\mathcal{D}_{x_i}$, ordered by $\prec_{x_i}$, containing all values from $\mathcal{D}$ except those that violate unary constraints on x_i . The set $\overline{\Delta}(x_i) := \mathcal{D}_{x_i} - \Delta(x_i)$ contains the values of $\mathcal{D}_{x_i}$ that have not yet been removed by arc consistency.
- A is the arc-consistent set built by the algorithm.
- For each $(x_i = a) \in A$, a set $CS(x_i, a)$ contains the assignments that are currently supported by $(x_i = a)$.
- The set **Pending** contains all the 4-tuples (x_i, a, x_j, b) such that a support for $(x_i = a)$ has to be sought on $c(x_i, x_j)$. If $b \neq \text{nil}$, this means that all values of x_j better than b have already been removed.

Pref-AC uses two subprocedures (Algorithm 1). Procedure **Activate** (x_i, a) adds an assignment $(x_i = a)$ to A , the set of current activated values. It initializes its data structure CS , and puts in **Pending** all the information needed to look for supports for $(x_i = a)$ on the constraints involving x_i . Function **BestSup** (x_i, a, x_j, b, B) looks for the best support for $(x_i = a)$ in B which is less preferred than b (we know there is no support in B better than or equal to b). B is $\overline{\Delta}(x_j)$ or A_{x_j} depending on the status of x_i (input variable or not).

Pref-AC works as follows. We start by initializing $\overline{\Delta}(x_i)$ to $\mathcal{D}_{x_i}$ for each x_i (line 1 of Algorithm 2). We activate the best value of the best variable (line 2). Then, 4-tuples (x_i, a, x_j, b) are taken from the **Pending** set (line 4), and if $(x_i = a)$ is still active (line 5), we must seek a (new) support c for it. If x_i is an input variable of c_{ij} , we seek the support in $\overline{\Delta}(x_j)$ because we have to ensure that it will be a preferred support (lines 6 to 7). Otherwise, we first seek a support c among the activated, but non-eliminated elements (line 8). If none exists, we seek a new best support c (line 10) following def. 6. If c exists, we activate it if not yet done, and store the fact that it supports $(x_i = a)$ (lines 11 to 13). If no support exists for $(x_i = a)$, we remove it from $\overline{\Delta}$ and A (line 14). If $\overline{\Delta}(x_i)$ is empty, a wipe out stops the procedure (line 15). If x_i is a source of the directed constraint graph and the best value of $\overline{\Delta}(x_i)$ is not in A_{x_i} ($(x_i = a)$ was the best of A_{x_i} and $\overline{\Delta}(x_i)$), the best value

Algorithm 1: Subprocedures

```

procedure Activate (in  $x_i$ : variable;  $a$ : value)
   $A_{x_i} \leftarrow A_{x_i} \cup \{a\}$ ;
   $CS(x_i, a) \leftarrow \emptyset$ ;
  Pending  $\leftarrow \{(x_i, a, x_j, \text{nil}) \mid c(x_i, x_j) \in C\}$ ;

function BestSup (in  $x_i$ ;  $a$ ;  $x_j$ ;  $b$ ;  $B$ ): value
  /* returns the best value in  $B$  supporting  $(x_i = a)$ , or nil if
  not found */;
  if  $b = \text{nil}$  then  $b \leftarrow \text{best}_{<_{x_j}}(B)$ ;
  else
    if  $b > \max(B)$  then return nil ;
     $b \leftarrow \text{next}_{<_{x_j}}(b, B)$ ;
  while  $(b \neq \text{nil})$  do
    if  $(a, b) \in c(x_i, x_j)$  then return  $b$ ;
    else  $b \leftarrow \text{next}_{<_{x_j}}(b, B)$ ;
  return nil;

```

of $\overline{\Delta}(x_i)$ has to be activated (lines 16 to 18). Finally, we have to put in **Pending** the information needed to propagate the deletion of $(x_i = a)$ (line 19).

Algorithm 2: Pref-AC

```

procedure Pref-AC()
1  $\overline{\Delta}(x_i) \leftarrow \mathcal{D}_{x_i}, \forall x_i \in \mathcal{X}$ ; Pending  $\leftarrow \emptyset$ ;
2 for each source  $x$  do Activate $(x, \text{best}_{<_x}(\overline{\Delta}(x)))$ ;
3 while Pending  $\neq \emptyset$  do
4   pick  $(x_i, a, x_j, b)$  from Pending ;
5   if  $(x_i = a) \in A$  then
6     if  $x_i$  is an input variable of  $c_{ij}$  then
7        $c \leftarrow \text{BestSup}(x_i, a, x_j, b, \overline{\Delta}(x_j))$ ;
8       else
9          $c \leftarrow \text{BestSup}(x_i, a, x_j, b, A_{x_j})$ ;
10       $c \leftarrow \text{BestSup}(x_i, a, x_j, b, \overline{\Delta}(x_j) - A_{x_j})$ ;
11   if  $c \neq \text{nil}$  then
12     if  $c \notin A_{x_j}$  then Activate $(x_j, c)$ ;
13     put  $(x_i = a)$  in  $CS(x_j, c)$ ;
14   else
15      $\overline{\Delta}(x_i) \leftarrow \overline{\Delta}(x_i) - \{a\}$ ;  $A \leftarrow A - \{(x_i = a)\}$ ;
16     if  $\overline{\Delta}(x_i) = \emptyset$  then return false ;
17     if  $x_i$  is a source then
18       if  $\text{best}_{<_{x_i}}(\overline{\Delta}(x_i)) \notin A_{x_i}$  then
19         Activate $(\text{best}_{<_{x_i}}(\overline{\Delta}(x_i)))$ ;
20   for each  $(x_j, b) \in CS(x_i, a)$  do
21     put  $(x_j, b, x_i, a)$  in Pending ;
22 return true ;

```

If Pref-AC terminates with an empty set of pending propagations (line 3), A only contains assignments for which a support has been activated on each constraint (because of line 12), and not deleted (because of line 19). In addition, this support is the preferred support for this assignment on this constraint if the variable was an input variable for the constraint (because of line 7 and the way **BestSup** works). We know A contains at least an assignment per variable because of the activation of supports. Therefore, A is an arc-consistent set. And because of lines 16-18, we know that for each x_i which

is a source of $\mathcal{X}$, the $<_{x_i}$ -best of $A_{x_i}^*$ is in A_{x_i} .

The space complexity of Pref-AC is the same as AC-6, namely $O(|\mathcal{C}| \cdot |\mathcal{D}|)$, since in the worst case, all the values are activated and have a support stored on each constraint. The time complexity is also bounded above by that of AC-6, namely $O(|\mathcal{C}| \cdot |\mathcal{D}|^2)$, since for each value in each domain, we perform at most $|\mathcal{D}|$ constraint checks on each constraint.

8 Conclusion

We have shown that it is sufficient to maintain an arc-consistent set for the preferred choices of a search procedure. This can significantly speed up constraint solving if the variable and value ordering heuristics of the search procedure can be described by preferences as is the case for typical configuration problems. We developed an algorithm called Pref-AC for achieving this reduction and are currently testing it for configuration problems with large domains.

Many real-world applications of constraint programming suffer from insufficient propagation, since most approaches support only bound consistency for crucial constraints such as precedences of activities, resources, and rest-time rules. Adapting algorithm Pref-AC to those constraints provides an interesting future perspective for improving constraint solving in areas such as scheduling, personnel planning, and other logistics applications. Future work will be devoted to improve Pref-AC such that it directly finds a preferred solution if the problem structure permits this (e.g. by activating preferred supports that are common to several variables).

Acknowledgements

We would like to thank Jean-Charles Régin and Olivier Lhomme for very helpful discussions.

References

- [Bessière *et al.*, 1999] C. Bessière, E.C. Freuder, and J.C. Régin. Using constraint metaknowledge to reduce arc consistency computation. *Artificial Intelligence*, 107:125–148, 1999.
- [Bessière, 1994] C. Bessière. Arc-consistency and arc-consistency again. *Artificial Intelligence*, 65:179–190, 1994.
- [Doyle, 2002] Jon Doyle. Preferences: Some problems and prospects. In *AAAI-02 Workshop on Preferences in AI and CP: Symbolic Approaches*. AAAI Press, 2002.
- [Junker and Mailharro, 2003] Ulrich Junker and Daniel Mailharro. Preference programming: Advanced problem solving for configuration. *AI-EDAM*, 17(1), 2003.
- [Junker, 2002] Ulrich Junker. Preference-based search and multi-criteria optimization. In *AAAI-02*, pages 34–40, Menlo Park, CA, 2002. AAAI Press.
- [Schiex *et al.*, 1996] T. Schiex, J.C. Régin, C. Gaspin, and G. Verfaillie. Lazy arc consistency. In *AAAI-96*, pages 216–221, 1996.

Amalgams of Constraint Satisfaction Problems

Andrei A. Bulatov

Oxford University Computing Laboratory
Oxford, UK
andrei.bulatov@comlab.ox.ac.uk

Eugeny S. Skvortsov

Ural State University, Ekaterinburg, Russia
Skvortsov@dc.ru

Abstract

Many of standard practical techniques of solving constraint satisfaction problems use various decomposition methods to represent a problem as a combination of smaller ones. We study a general method of decomposing constraint satisfaction problems in which every constraint is represented as a disjunction of two or more simpler constraints defined, possibly, on smaller sets of values. We call a problem an amalgam if it can be decomposed in this way. Some particular cases of this construction have been considered in [Cohen *et al.*, 1997; 2000b; 2000a] including amalgams of problems with disjoint sets of values, and amalgams of independent problems. In this paper, we concentrate on constraint classes determined by relational clones, and study amalgams of such classes in the general case of arbitrary finite sets of values. We completely characterise amalgams of this form solvable in polynomial time and provide efficient algorithms.

1 Introduction

In the Constraint Satisfaction Problem (CSP) the aim is to find an assignment to a set of variables subject specified constraints. CSP provides a generic approach to various combinatorial problems frequently appearing in artificial intelligence and computer science, including the propositional satisfiability problem, in which the variables must be assigned Boolean values, graph-theoretical problems, scheduling problems, temporal and spatial reasoning, database theory, and many others.

The general CSP is NP-complete [Montanari, 1974]. However, the time complexity of many practical problems can be considerably decreased by representing a problem as a combination of smaller problems. Such decomposition methods have been intensively studied and implemented in most of existing constraint solvers. Most of them deals with the hypergraph associated with a problem instance (see, e.g. [Gottlob *et al.*, 2000; Dechter and Pearl, 1989; Freuder, 1990]), and to date are highly developed. Another possibility, decomposition of constraints themselves [Cohen *et al.*, 1997; 2000b;

2000a], remains almost uninvestigated. The present paper focuses on this class of decomposition methods.

In many cases of interest we may restrict the form of allowed constraints by specifying a constraint language, that is a set of allowed constraints. Every constraint is specified by a relation, so, a constraint language is just a set of relations over the set of values. One of the most natural ways to decompose such a restricted CSP is to represent every its constraint as a disjunction of two or more simpler constraints on, possibly, smaller sets of values. We say that a constraint language, Γ , is the *amalgam* of Γ_1, Γ_2 if every constraint in Γ is the disjunction of a constraint from Γ_1 and a constraint from Γ_2 . In this case we call Γ_1, Γ_2 the *components* of Γ .

The main research direction in the study of amalgams is, of course, developing efficient algorithms solving the corresponding constraint satisfaction problem. However, as we shall see the complexity of the CSP arising from an amalgam is not determined automatically by the complexity of its components. Therefore, the first research problem we deal with is the *tractability problem*: under which conditions the problem arising from an amalgam is tractable. (A problem is called tractable if it can be solved in polynomial time.) Then we tackle the *algorithmic problem*: design efficient algorithms for tractable amalgams.

As is naturally expected, algorithms for amalgams tend to use algorithms for their components, especially if ones are already known. Unfortunately, in general, the connection between constraint languages and their amalgam cannot be expressed by usual constraint techniques, and strongly depends on properties of particular constraint languages. Thus, solutions to both research problems are expected to be nontrivial.

Amalgams have been introduced in [Cohen *et al.*, 1997], though another name was used. In [Cohen *et al.*, 2000a], amalgams were considered in the simplest case when the sets of values for languages Γ_1, Γ_2 are disjoint. In this case, for any amalgam, the corresponding problem is trivially reducible to the problems over the components, that solves both the algorithmic problem and the tractability problem: an amalgam gives rise to a tractable problem if and only its components do. Certain properties of interaction of constraint languages may yield a reduction of an amalgam to its components. In [Cohen *et al.*, 1997] and later in [Cohen *et al.*, 2000b], several such properties, so-called *independence* of constraint languages, have been identified. A number of

previously unknown tractable constraint languages have been represented as amalgams of very simple and well studied independent constraint languages.

It has been shown in [Jeavons, 1998b] that the tractability of the constraint satisfaction problem arising from a constraint language implies the tractability of the problem for the *relational clone* generated by the language. Therefore, a reasonable strategy is to concentrate on relational clones rather than arbitrary constraint languages. In this paper we solve the tractability and algorithmic problems for amalgams of relational clones. We completely characterise tractable amalgams and provide efficient algorithms in this case. The characterisation criteria is stated in terms of, first, properties of components of an amalgam, and second, the tractability of a certain constraint language on a 2-element set of values. The latter language reflects the interaction of the components. Thus, both the characterisation and the reducing algorithms use Schaefer's Dichotomy Theorem for Boolean constraints [Schaefer, 1978].

Throughout the paper we heavily use the algebraic technique for CSP developed in [Jeavons, 1998b; Jeavons *et al.*, 1998].

2 Preliminaries

2.1 Basic Definitions

Let A be a finite set. The set of all n -tuples of elements of A is denoted A^n . A subset of A^n is called an n -ary *relation* on A , and any set of finitary relations on A is called a *constraint language* on A .

Definition 1 Let Γ be a constraint language on a set A . $\text{CSP}(\Gamma)$ is the combinatorial decision problem whose instance is a triple $\mathcal{P} = (V; A; \mathcal{C})$ in which V is a set of variables, and $\mathcal{C}$ is a set of constraints, that is pairs of the form $C = \langle s, \varrho \rangle$ where s is a list of variables of length m_C , called the constraint scope, and $\varrho \in \Gamma$ is an m_C -ary relation, called the constraint relation. The question is whether there exists a solution to $\mathcal{P}$, that is a mapping $\varphi: V \rightarrow A$ such that $\varphi(s) \in \varrho$ for all $\langle s, \varrho \rangle \in \mathcal{C}$.

Example 1 An instance of GRAPH q -COLORABILITY consists of a graph G . The question is whether the vertices of G can be labelled with q colours so that adjacent vertices are assigned different colours.

This problem corresponds to $\text{CSP}(\{\neq_A\})$ where A is a q -element set (of colours) and $\neq_A = \{(a, b) \in A^2 \mid a \neq b\}$ is the disequality relation on A .

A constraint language Γ is said to be *tractable* [NP-complete] if the problem $\text{CSP}(\Gamma)$ is tractable [NP-complete].

Now we introduce the central notion of the paper.

Definition 2 Let Γ_A, Γ_B be constraint languages on sets A, B respectively. The amalgam of Γ_A, Γ_B is defined to be the constraint language on $A \cup B$

$$\Gamma_A \dot{\bowtie} \Gamma_B = \{\varrho_A \cup \varrho_B \mid \varrho_A \in \Gamma_A, \varrho_B \in \Gamma_B, \varrho_A, \varrho_B \text{ are of the same arity}\}.$$

Example 2 Let $A = \{a, c, d\}, B = \{b, c, d\}$, and $\Gamma_A = \{\varrho_A\}, \Gamma_B = \{\varrho_B\}$ where ϱ_A, ϱ_B are partial orders:

$$\varrho_A = \begin{pmatrix} a & c & d & c & a & a \\ a & c & d & d & d & c \end{pmatrix}, \quad \varrho_B = \begin{pmatrix} b & c & d & c & d & d \\ b & c & d & b & b & c \end{pmatrix}.$$

Columns of the matrices represent tuples of the relations. Then $\Gamma_A \dot{\bowtie} \Gamma_B = \{\varrho_A \cup \varrho_B\}$ where $\varrho_A \cup \varrho_B$ is a quasiorder.

Example 3 ([Cohen *et al.*, 2000b]) Let $\mathbb{Z}$ be the set of all integers, and Γ the set of congruences of the form $x \equiv a \pmod{m}$ treated as unary relations. The Chinese Remainder Theorem implies that Γ is tractable. The amalgam $\Gamma \dot{\bowtie} \Gamma$ consists of expressions of the form $x \equiv a \pmod{m} \vee x \equiv b \pmod{n}$, and by results of [Cohen *et al.*, 2000b] is tractable. For example, this means that we are able to recognise in polynomial time the consistency of the system

$$\begin{aligned} x_1 &\equiv 1 \pmod{4} \vee x_1 \equiv 0 \pmod{3} \\ x_2 &\equiv 2 \pmod{3} \vee x_2 \equiv 2 \pmod{5} \\ x_3 &\equiv 5 \pmod{7}. \end{aligned}$$

We are concerned with the following two problems.

Problem 1 (tractability problem) When is $\text{CSP}(\Gamma_1 \dot{\bowtie} \Gamma_2)$ tractable?

Problem 2 (algorithmic problem) Find a polynomial time algorithm for tractable $\text{CSP}(\Gamma_1 \dot{\bowtie} \Gamma_2)$.

In fact, properties of the amalgam do not strongly depend neither on properties of the original constraint languages nor on the way they interact. For example, if Γ_1, Γ_2 contains no relations of the same arity then their amalgam is empty. We, therefore, should restrict the class of constraint languages to be studied.

2.2 Relational Clones

For any problem in $\text{CSP}(\Gamma)$, there may be some sets of variables whose possible values subject to certain constraints which are not elements of Γ . These constraints are said to be *implicit* and arise from interaction of constraints specified in the problem [Jeavons, 1998a].

To describe implicit constraint relations we make use of the natural correspondence between relations and predicates: for an n -ary relation ϱ on a set A , the n -ary predicate P_ϱ is true on a tuple $\mathbf{a}$ if and only if $\mathbf{a} \in \varrho$. Usually, we will not distinguish a relation and the corresponding predicate, and freely use both terminology. An existential first order formula $\exists y_1, \dots, y_m \Phi(x_1, \dots, x_n, y_1, \dots, y_m)$ is said to be *primitive positive* (pp-) if its quantifier-free part Φ is a conjunction of atomic formulas.

Definition 3 A relation is an implicit relation of a constraint language Γ on a set A if it can be expressed by a pp-formula involving relations from Γ and the equality relation $=_A$.

A constraint language Γ is said to be a relational clone if it contains all its implicit constraint relations.

The relational clone $\langle \Gamma \rangle$ of all implicit constraint relations of Γ is called the relational clone generated by Γ .

Example 4 The intersection of relations of the same arity, and Cartesian product are expressible via pp-formulas:

- Let ϱ, σ be n -ary relations. Then $P_{\varrho \cap \sigma}(x_1, \dots, x_n) = P_\varrho(x_1, \dots, x_n) \wedge P_\sigma(x_1, \dots, x_n)$.
- Let ϱ be n -ary, and σ m -ary relations. Then $P_{\varrho \times \sigma}(x_1, \dots, x_n) \wedge P_\sigma(x_{n+1}, \dots, x_{n+m})$.

The notion of a relational clone considerably simplifies the analysis of constraint satisfaction problems in view of the following result that links the complexity of a constraint language and the relational clone it generates.

Theorem 1 ([Jeavons, 1998b]) *Let Γ be a constraint language on a finite set, and $\Gamma' \subseteq \langle \Gamma \rangle$ finite. Then $\text{CSP}(\Gamma')$ is polynomial time reducible to $\text{CSP}(\Gamma)$.*

This result motivates restricting Problems 1,2 to the class of relational clones.

Example 3 (continuation) Reconsider the constraint language Γ from Example 3. Results of [Cohen *et al.*, 2000b] implies that $\langle \Gamma \rangle \bowtie \langle \Gamma \rangle$ is also tractable. This larger amalgam includes, e.g., constraints of the form $x \equiv a \pmod{m} \vee y \equiv b \pmod{n}$, which are not members of $\Gamma \bowtie \Gamma$.

2.3 Invariance Properties of Constraints

Another advantage of considering relational clones is that they often admit a concise description in terms of algebraic invariance properties [Pöschel and Kalužnin, 1979; Jeavons, 1998b]. An (m -ary) operation f on a set A preserves an n -ary relation ϱ on A (or ϱ is invariant under f , or f is a polymorphism of ϱ) if for any $(a_{11}, \dots, a_{n1}), \dots, (a_{1m}, \dots, a_{nm}) \in \varrho$ the tuple $(f(a_{11}, \dots, a_{1m}), \dots, f(a_{n1}, \dots, a_{nm}))$ belongs to ϱ . For a given set of operations, C , the set of all relations invariant under every operation from C is denoted by $\text{Inv } C$. Conversely, for a set of relations, Γ , the set of all operations preserving every relation from Γ is denoted by $\text{Pol } \Gamma$. Every relational clone can be represented in the form $\text{Inv } C$ for a certain set of operations C [Pöschel and Kalužnin, 1979]. Therefore, in view of Theorem 1 the complexity of a finite constraint language depends only on its polymorphisms.

We need operations of some particular types that give rise to tractable problem classes.

Definition 4 *Let A be a finite set. An operation f on A is called*

- a constant operation if there is $c \in A$ such that $f(x_1, \dots, x_n) = c$, for any $x_1, \dots, x_n \in A$;
- a semilattice operation¹, if it is binary and satisfies the following three conditions: (a) $f(x, x) = x$, (b) $f(x, f(y, z)) = f(f(x, y), z)$, (c) $f(x, y) = f(y, x)$, for any $x, y, z \in A$;
- a majority operation if it is ternary, and $f(x, x, y) = f(x, y, x) = f(y, x, x) = x$, for any $x, y \in A$;
- affine if $f(x, y, z) = x - y + z$, for any $x, y, z \in A$, where $+$, $-$ are the operations of an Abelian group.

Proposition 1 ([Jeavons *et al.*, 1998; 1997]) *If Γ is a constraint language on a finite set, and $\text{Pol } \Gamma$ contains an operation of one of the following types: constant, semilattice, affine, majority; then Γ is tractable.*

The complexity of constraint languages on a 2-element set is completely characterised in [Schaefer, 1978]. This outstanding result is known as Schaefer's Dichotomy Theorem. By making use of Proposition 1 the algebraic version of Schaefer's theorem can be derived [Jeavons *et al.*, 1997].

¹Note that in some earlier papers [Jeavons, 1998b; Jeavons *et al.*, 1998] the term *ACI operation* is used.

Theorem 2 (Schaefer, [Schaefer, 1978]) *A constraint language Γ on a 2-element set is tractable if and only if $\text{Pol } \Gamma$ contains one of the operations listed in Proposition 1. Otherwise Γ is NP-complete.*

3 Tractable amalgams

In this section we give a complete solution of Problems 1,2 for amalgams of relational clones. Throughout the section A, B are finite sets, $D = A \cap B$, and R_A, R_B are relational clones on A, B respectively. First, we reduce Problems 1,2 to the case when $|D| = 1$ and prove NP-completeness results in this case. Then we concentrate on tractable cases and present a solving algorithm for these cases.

3.1 The result

We assume that R_A, R_B contain empty relations of any arity, because otherwise $\text{Pol}(R_A \bowtie R_B)$ contains a constant operation, and therefore $R_A \bowtie R_B$ is tractable by Proposition 1.

The case when D is empty was completely investigated in [Cohen *et al.*, 2000a].

Proposition 2 ([Cohen *et al.*, 2000a]) *If $D = \emptyset$ then $\text{CSP}(R_A \bowtie R_B)$ is polynomial time reducible to $\text{CSP}(R_A)$, $\text{CSP}(R_B)$. Hence, $R_A \bowtie R_B$ is tractable if and only if both R_A, R_B are tractable.*

So, we assume D to be non-empty.

Proposition 3 ([Jeavons *et al.*, 1997; Jeavons, 1998b]) *If f is a unary polymorphism of a constraint language Γ then the problem class $\text{CSP}(f(\Gamma))$ where $f(\Gamma) = \{f(\varrho) \mid \varrho \in \Gamma\}$, $f(\varrho) = \{(f(a_1), \dots, f(a_n)) \mid (a_1, \dots, a_n) \in \varrho\}$, is polynomial time equivalent to $\text{CSP}(\Gamma)$.*

By $f|_C$ we denote the restriction of a (unary) operation f onto a set C . The following statement is straightforward.

Proposition 4 (1) *If f is a unary polymorphism of $R_A \bowtie R_B$ then $f|_A, f|_B$ are polymorphisms of R_A, R_B , respectively.*

(2) $\text{CSP}(R_A \bowtie R_B)$ is polynomial time equivalent to $\text{CSP}(f|_A(R_A) \bowtie f|_B(R_B))$.

(3) *If the amalgam $R_A \bowtie R_B$ is tractable then there is a polymorphism f of $R_A \bowtie R_B$ such that $|f(D)| = 1$. Otherwise, $R_A \bowtie R_B$ is NP-complete.*

Therefore, we may restrict ourselves to the case $|D| = 1$; let $D = \{c\}$. In this case we need more notation and terminology.

Definition 5 *The relational clone $R_A [R_B]$ is said to be monolithic if $\bar{R}_A = \langle R_A \cup \{D\} \rangle$ [$\bar{R}_B = \langle R_B \cup \{D\} \rangle$] contains no unary relation E with $c \notin E$.*

Let R_A^+ denote the relational clone generated by the set $R_A \cup \{\varrho \cup \{\bar{c}\} \mid \varrho \in R_A\}$ where $\bar{c} = (c, \dots, c)^2$. Obviously, $\bar{R}_A \subseteq R_A^+$. The clone R_B^+ is defined analogously.

²We do not specify the length of the tuple $\bar{c} = (c, \dots, c)$, because it is always clear from the context.

In the following definition we introduce an auxiliary relational clone on the 2-element set $Z = \{A, B\}$ that describes interaction of the components of the amalgam. For an $(n\text{-ary})$ relation $\varrho \in R_A$, let $\tilde{\varrho} = \{(a_1, \dots, a_n, A) \mid (a_1, \dots, a_n) \in \varrho_A\} \cup \{(c, \dots, c, B)\}$. We also define the corresponding predicate $\tilde{P}_\varrho(x_1, \dots, x_n, y)$ which is true if and only if $(x_1, \dots, x_n, y) \in \tilde{\varrho}$. The variable y will be called *distinguished*.

Definition 6 The meta-clone $\mathbb{R}_A$ is defined to be the set of all relations on Z expressible through a pp-formula of the form $\exists x_1, \dots, x_n z_1, \dots, z_k \Phi(x_1, \dots, x_n, y_1, \dots, y_m, z_1, \dots, z_k)$ where Φ is a conjunction of predicates $\tilde{P}_\varrho$ for $\varrho \in R_A^+$, and $y_1, \dots, y_m, z_1, \dots, z_k$ are the distinguished variables.

The meta-clone $\mathbb{R}_B$ is defined in a similar way.

Lemma 1 Meta-clones $\mathbb{R}_A, \mathbb{R}_B$ are relational clones on Z .

We now in a position to state the main result of this paper.

Theorem 3 Let $|D| = 1$. Then

1. if R_A is monolithic then $R_A \bowtie R_B$ is tractable if and only if R_B^+ is tractable;

2. if both R_A, R_B are not monolithic then $R_A \bowtie R_B$ is tractable if and only if R_A^+, R_B^+ , and $\mathbb{R}_A \cup \mathbb{R}_B$ are tractable.

It is not hard to see that $R_A^+, R_B^+ \subseteq \langle R_A \bowtie R_B \rangle$. Therefore, by Theorem 1, R_B^+ is tractable whenever $R_A \bowtie R_B$ is tractable. The converse inclusion follows from Lemma 2. Then, it can be proved by straightforward calculation that $\text{CSP}(\mathbb{R}_A \cup \mathbb{R}_B)$ is polynomial time reducible to $\text{CSP}(R_A \bowtie R_B)$. Finally, in Sections 3.2, 3.3, we show how to reduce $\text{CSP}(R_A \bowtie R_B)$ to $\text{CSP}(\mathbb{R}_A \cup \mathbb{R}_B)$, $\text{CSP}(R_A^+)$, and $\text{CSP}(R_B^+)$.

Lemma 2 If R_A is monolithic then $\text{CSP}(R_A \bowtie R_B)$ is polynomial time reducible to $\text{CSP}(R_B^+)$.

Proof: Let R_A be monolithic. We set out the proof in several claims.

CLAIM 1. For any $a \in A - \{c\}$, there is a unary polymorphism f_a of $\overline{R}_A$ such that $f_a(a) = c$.

Let $g_1, \dots, g_k$ be the list of all unary polymorphisms of $\overline{R}_A$, and $E = \{a_1 = g_1(a), \dots, a_k = g_k(a)\}$. Suppose that $c \notin E$. Since R_A is monolithic, $E \notin \overline{R}_A$. Therefore, there is a polymorphism $h(x_1, \dots, x_n)$ such that $h(b_1, \dots, b_n) \notin E$ for some $b_i \in E$, say, $b_i = a_{j_i}$. The operation $h'(x) = h(g_{j_1}(x), \dots, g_{j_n}(x))$ is also a polymorphism of $\overline{R}_A$ [Pöschel and Kalužnin, 1979], and $h'(a) \notin E$, a contradiction.

CLAIM 2. $\overline{R}_A$ is invariant under the constant operation c .

Since $D = \{c\} \in \overline{R}_A$, for any $a \in A - \{c\}$, we have $f_a(c) = c$. This implies $|f_{a_1}(A)| \leq |A| - 1$ for any $a_1 \in A - \{c\}$. Then choosing an element $a_2 \in f_{a_1}(A) - \{c\}$ we get $|f_{a_2}(f_{a_1}(A))| \leq |A| - 2$. Continuing the process we eventually obtain an operation $f(x) = f_{a_1}(\dots f_{a_1}(x) \dots)$ such that $|f(A)| = 1$. As $f(c) = c$, f is the required constant operation.

Claim 2 implies that if $\varrho \in \overline{R}_A$ is non-empty then $\bar{c} \in \varrho$.

CLAIM 3. The operation g on $A \cup B$ where $g(x) = c$ if $x \in A$ and $g(x) = x$ if $x \in B$ is a polymorphism of $R_A \bowtie R_B$.

Take $\varrho \in R_A \bowtie R_B$ and $\mathbf{d} \in \varrho = \varrho_A \cup \varrho_B$, $\varrho_A \in R_A$, $\varrho_B \in R_B$. If $\mathbf{d} \in \varrho_B$ then $g(\mathbf{d}) = \mathbf{d} \in \varrho$. If $\mathbf{d} \in \varrho_A$ then $\varrho_A \neq \emptyset$, $\bar{c} \in \varrho_A \subseteq \varrho$, and therefore $g(\mathbf{d}) = \bar{c} \in \varrho$.

Finally, for any $\varrho = \varrho_A \cup \varrho_B \in R_A \bowtie R_B$, $g(\varrho) = \varrho_B$ or $g(\varrho) = \varrho_B \cup \{\bar{c}\}$. This means that $g(R_A \bowtie R_B) = R_B^+$, and by Proposition 3, $\text{CSP}(R_A \bowtie R_B)$ is polynomial time equivalent to $\text{CSP}(R_B^+)$. $\square$

3.2 The Split Problem

We transform a problem over an amalgam so that the interaction of its components becomes more transparent.

Lemma 3 If $R_A [R_B]$ is not monolithic then there is a relation $\sigma_A \in R_A [\sigma_B \in R_B]$ of arity $l_A [l_B]$ such that $\bar{c} \notin \sigma_A [\bar{c} \notin \sigma_B]$.

Proof: Let $E \in \overline{R}_A$ be such that $c \notin E$. Since E belongs to the clone generated by $R_A \cup \{D\}$, it can be expressed by a pp-formula

$$P_E(x) = \exists y_1, \dots, y_n, z_1, \dots, z_m$$

$$\Phi(x, y_1, \dots, y_n, z_1, \dots, z_m) \wedge P_D(z_1) \wedge \dots \wedge P_D(z_m)$$

where Φ uses predicates from R_A . As is easily seen, the relation $P_\sigma(x, z_1, \dots, z_m) = \exists y_1, \dots, y_n \Phi(x, y_1, \dots, y_n, z_1, \dots, z_m)$ satisfies the required conditions. $\square$

Let $\mathcal{P} = (V; A \cup B; \mathcal{C})$ be a problem instance from $\text{CSP}(R_A \bowtie R_B)$. Fix a representation $\varrho = \varrho_A \cup \varrho_B$, $\varrho_A \in R_A$, $\varrho_B \in R_B$ for every constraint relation ϱ . The *split problem* $\tilde{\mathcal{P}}$ is defined to be the triple $(\tilde{V}; A \cup B; \tilde{\mathcal{C}})$ (see Fig. 1) in which

- $\tilde{V} = V_A \cup V_B \cup V'_A \cup V'_B$ where for every $v \in V$ there are $v^A \in V_A$ and $v^B \in V_B$, and for every constraint $C \in \mathcal{C}$ there are $u^1_C, \dots, u^{l_A}_C \in V'_A$ and $w^1_C, \dots, w^{l_B}_C \in V'_B$; let us denote $\bar{u}_C, \bar{w}_C$ the sequences $u^1_C, \dots, u^{l_A}_C, w^1_C, \dots, w^{l_B}_C$ respectively;
- $\tilde{\mathcal{C}} = \mathcal{C}_A \cup \mathcal{C}_B \cup \mathcal{C}'$ where for every $C = \langle s, \varrho \rangle \in \mathcal{C}$, $s = (v_1, \dots, v_n)$, we include 3 constraints
 - $\langle (v^A_1, \dots, v^A_n, \bar{u}_C), \varrho'_A \rangle \in \mathcal{C}_A$, $\varrho'_A = (\varrho \times \sigma_A) \cup \{\bar{c}\}$;
 - $\langle (v^B_1, \dots, v^B_n, \bar{w}_C), \varrho'_B \rangle \in \mathcal{C}_B$, $\varrho'_B = (\varrho \times \sigma_B) \cup \{\bar{c}\}$;
 - $\langle (\bar{u}_C, \bar{w}_C), \sigma \rangle \in \mathcal{C}'$ where σ is the $l_A + l_B$ -ary relation $(\sigma_A \times \{\bar{c}\}) \cup (\{\bar{c}\} \times \sigma_B)$.

The following lemma is straightforward.

Lemma 4 $\mathcal{P}$ is equivalent to $\tilde{\mathcal{P}}$.

On the one hand, the restrictions of $\tilde{\mathcal{P}}$ on $V_A \cup V'_A, V_B \cup V'_B$ are problems from $\text{CSP}(R_A^+)$, $\text{CSP}(R_B^+)$ respectively. On the other hand, the implicit constraint relation arising on the set V'_A in this problem can be treated as a relation on the 2-element set $\{\sigma_A, D^{l_A}\}$, and correspond, in fact, to the relations from $\mathbb{R}_A$. Therefore, the meta-clones describe the interaction of two parts of the split problem.

3.3 Solving the Split Problem

Let us assume that $R_A^+, R_B^+, \mathbb{R}_A \cup \mathbb{R}_B$ are tractable, and R_A, R_B are not monolithic. By Theorem 2, there is either (1) constant, or (2) semilattice, or (3) majority, or (4) affine operation f such that every relation from $\mathbb{R}_A \cup \mathbb{R}_B$ is invariant under f . We show that under this assumption the split problem

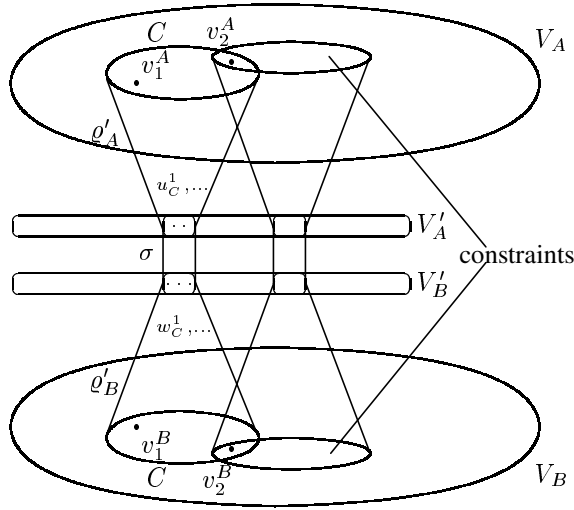


Figure 1:

$\tilde{\mathcal{P}}$ can be reduced to problems from $\text{CSP}(R_A^+)$, $\text{CSP}(R_B^+)$ in polynomial time.

We divide $\tilde{\mathcal{P}}$ into two subproblems $\tilde{\mathcal{P}}_A = (V_A \cup V'_A; A; \mathcal{C}_A)$ and $\tilde{\mathcal{P}}_B = (V_B \cup V'_B; B; \mathcal{C}_B)$ from $\text{CSP}(R_A^+)$, $\text{CSP}(R_B^+)$ respectively. Recall also that $\mathcal{C} = \{C_1, \dots, C_q\}$. Then set

$$\begin{aligned} S_A &= \{(\varphi(u_{C_1}^1), \dots, \varphi(u_{C_1}^{l_A}), \varphi(u_{C_2}^1), \dots, \varphi(u_{C_q}^{l_A})) \mid \\ &\quad \varphi \text{ is a solution to } \tilde{\mathcal{P}}_A\}, \\ S_B &= \{(\varphi(w_{C_1}^1), \dots, \varphi(w_{C_1}^{l_B}), \varphi(w_{C_2}^1), \dots, \varphi(w_{C_q}^{l_B})) \mid \\ &\quad \varphi \text{ is a solution to } \tilde{\mathcal{P}}_B\}. \end{aligned}$$

Let also mappings $\psi_A: \sigma_A \cup \{\bar{c}\} \rightarrow Z$, $\psi_B: \sigma_B \cup \{\bar{c}\} \rightarrow Z$ be defined as follows

$$\psi_A(\mathbf{d}) = \begin{cases} A, & \text{if } \mathbf{d} \in \sigma_A, \\ B, & \text{if } \mathbf{d} = \bar{c}, \end{cases} \quad \psi_B(\mathbf{d}) = \begin{cases} A, & \text{if } \mathbf{d} = \bar{c}, \\ B, & \text{if } \mathbf{d} \in \sigma_B. \end{cases}$$

Then $\bar{S}_A = \psi_A(S_A) = \{(\psi_A(a_{C_1}^1, \dots, a_{C_1}^{l_A}), \dots, \psi_A(a_{C_q}^1, \dots, a_{C_q}^{l_A})) \mid (a_{C_1}^1, \dots, a_{C_q}^{l_A}) \in S_A\}$ is a q -ary relation from $\mathbb{R}_A$, and $\bar{S}_B = \psi_B(S_B) = \{(\psi_B(b_{C_1}^1, \dots, b_{C_1}^{l_B}), \dots, \psi_B(b_{C_q}^1, \dots, b_{C_q}^{l_B})) \mid (b_{C_1}^1, \dots, b_{C_q}^{l_B}) \in S_B\}$ is a q -ary relation from $\mathbb{R}_B$. A solution to $\tilde{\mathcal{P}}$ exists if and only if there are solutions $\varphi_A \in S_A$, $\varphi_B \in S_B$ such that $\psi_A \varphi_A = \psi_B \varphi_B$ or, equivalently, if $\bar{S}_A \cap \bar{S}_B \neq \emptyset$. The i th component of a tuple $\mathbf{d}$ will be denoted by $\mathbf{d}[i]$. Fix tuples $\mathbf{a} \in \sigma_A$, $\mathbf{b} \in \sigma_B$.

CASE 1. $\mathbb{R}_A \cup \mathbb{R}_B$ is invariant under a constant operation.

Without loss of generality we may assume that the constant operation maps Z to A . Then either $\bar{S}_A$ [$\bar{S}_B$] is empty, or $(A, \dots, A) \in \bar{S}_A$ [$(A, \dots, A) \in \bar{S}_B$]. If $\bar{S}_A = \emptyset$ or $\bar{S}_B = \emptyset$ then S_A or S_B is empty, consequently, $\tilde{\mathcal{P}}$ has no solution. If $\bar{S}_A, \bar{S}_B \neq \emptyset$ then $\tilde{\mathcal{P}}_A$ has a solution $\varphi_A \in S_A$ such that $\varphi_A(u_C^i) = \mathbf{a}[i]$ for all $C \in \mathcal{C}$, $1 \leq i \leq l_A$, and $\tilde{\mathcal{P}}_B$ has a solution $\varphi_B \in S_B$ with $\varphi_B(w_C^j) = \mathbf{b}[j]$ for all $C \in \mathcal{C}$, $1 \leq j \leq l_B$. As is easily seen, $\psi_A \varphi_A = \psi_B \varphi_B$.

We need some additional notation. Let ϱ be an n -ary relation, and $I = \{i_1, \dots, i_k\} \subseteq \{1, \dots, n\}$; then $\pi_I(\varrho)$ denotes the k -ary relation $\{\pi_I(\mathbf{d}) \mid \mathbf{d} \in \varrho\}$ where $\pi_I(\mathbf{d}) = (\mathbf{d}[i_1], \dots, \mathbf{d}[i_k])$. For a problem instance $\mathcal{P} = (W; D; \mathcal{C})$ and a constraint $\langle s, \varrho \rangle$, we denote the problem instance $(W; D; \mathcal{C} \cup \{\langle s, \varrho \rangle\})$ by $\mathcal{P} \cup \{\langle s, \varrho \rangle\}$.

CASE 2. $\mathbb{R}_A \cup \mathbb{R}_B$ is invariant under a semilattice operation f .

Without loss of generality we may assume that $f(A, A) = f(A, B) = f(B, A) = A$, $f(B, B) = B$. Then [Jeavons *et al.*, 1997] any relation $\varnothing \neq \varrho \in \mathbb{R}_A \cup \mathbb{R}_B$ contains the tuple $\max_\varrho$ where $\max_\varrho[i] = A$ if $A \in \pi_{\{i\}}(\varrho)$, and $\max_\varrho[i] = B$ otherwise. We employ the following algorithm.

Until no new constraint added, **for each** $C \in \mathcal{C}$ **do**

- **Solve** the problems $\tilde{\mathcal{P}}_A \cup \{\langle \bar{u}_C, \sigma_A \rangle\}$, $\tilde{\mathcal{P}}_B \cup \{\langle \bar{w}_C, \{\bar{c}\} \rangle\}$.
- **If** one of them has no solution **then replace** $\tilde{\mathcal{P}}_A$ with $\tilde{\mathcal{P}}_A \cup \{\langle \bar{u}_C, \{\bar{c}\} \rangle\}$ and $\tilde{\mathcal{P}}_B$ with $\tilde{\mathcal{P}}_B \cup \{\langle \bar{w}_C, \sigma_B \rangle\}$.
- **Solve** the problems $\tilde{\mathcal{P}}_A \cup \{\langle \bar{u}_C, \{\bar{c}\} \rangle\}$, $\tilde{\mathcal{P}}_B \cup \{\langle \bar{w}_C, \sigma_B \rangle\}$.
- **If** one of them has no solution **then replace** $\tilde{\mathcal{P}}_A$ with $\tilde{\mathcal{P}}_A \cup \{\langle \bar{u}_C, \sigma_A \rangle\}$ and $\tilde{\mathcal{P}}_B$ with $\tilde{\mathcal{P}}_B \cup \{\langle \bar{w}_C, \{\bar{c}\} \rangle\}$.

Notice that this algorithm is quite analogous to *establishing 1-consistency* [Jeavons *et al.*, 1998; Dechter and van Beek, 1997; Cooper, 1989]. Let $\tilde{\mathcal{P}}'_A, \tilde{\mathcal{P}}'_B$ be the obtained problems, S'_A, S'_B defined for $\tilde{\mathcal{P}}'_A, \tilde{\mathcal{P}}'_B$ in the same way as S_A, S_B , and $\bar{S}'_A = \psi_A(S'_A)$, $\bar{S}'_B = \psi_B(S'_B)$. Clearly, $\bar{S}_A \cap \bar{S}_B \neq \emptyset$ if and only if $\bar{S}'_A \cap \bar{S}'_B \neq \emptyset$. Moreover, since $\pi_{\{i\}} \bar{S}'_A = \pi_{\{i\}} \bar{S}'_B$, for any i , we have $\max_{\bar{S}'_A} = \max_{\bar{S}'_B} \in \bar{S}'_A \cap \bar{S}'_B$ whenever $\bar{S}'_A, \bar{S}'_B \neq \emptyset$. Thus, $\tilde{\mathcal{P}}$ has a solution if and only if $\tilde{\mathcal{P}}'_A, \tilde{\mathcal{P}}'_B$ have solutions.

CASE 3. $\mathbb{R}_A \cup \mathbb{R}_B$ is invariant under a majority operation f .

In this case any $(n$ -ary) relation $\varrho \in \mathbb{R}_A \cup \mathbb{R}_B$ is 2-decomposable [Jeavons *et al.*, 1998], that is every n -tuple $\mathbf{d}$ such that, for any $1 \leq i, j \leq n$, $(\mathbf{d}[i], \mathbf{d}[j]) \in \pi_{\{i,j\}} \varrho$, belongs to ϱ . Therefore, $\bar{S}_A \cap \bar{S}_B \neq \emptyset$ if and only if there is a tuple $\mathbf{d}$ such that $(\mathbf{d}[i], \mathbf{d}[j]) \in \pi_{\{i,j\}}(\bar{S}_A) \cap \pi_{\{i,j\}}(\bar{S}_B)$, for any $1 \leq i, j \leq q$.

The latter condition is encoded by the constraint satisfaction problem $(W; Z; \mathcal{C}'')$ in which $W = \{z_1, \dots, z_q\}$, and for any $1 \leq i, j \leq q$, there is the constraint $\langle (z_i, z_j), \pi_{\{i,j\}}(\bar{S}_A) \cap \pi_{\{i,j\}}(\bar{S}_B) \rangle$. Since all the constraint relations are in $\mathbb{R}_A \cup \mathbb{R}_B$, this problem can be solved in polynomial time (see [Jeavons *et al.*, 1998]). We just have to find relations of the form $\pi_{\{i,j\}}(\bar{S}_A)$, $\pi_{\{i,j\}}(\bar{S}_B)$. To this end we consider the problems

$$\begin{aligned} \tilde{\mathcal{P}}_A^{i,j}(A, A) &= \tilde{\mathcal{P}}_A \cup \{\langle \bar{u}_{C_i}, \sigma_A \rangle, \langle \bar{u}_{C_j}, \sigma_A \rangle\}, \\ \tilde{\mathcal{P}}_A^{i,j}(A, B) &= \tilde{\mathcal{P}}_A \cup \{\langle \bar{u}_{C_i}, \sigma_A \rangle, \langle \bar{u}_{C_j}, \{\bar{c}\} \rangle\}, \\ \tilde{\mathcal{P}}_A^{i,j}(B, A) &= \tilde{\mathcal{P}}_A \cup \{\langle \bar{u}_{C_i}, \{\bar{c}\} \rangle, \langle \bar{u}_{C_j}, \sigma_A \rangle\}, \\ \tilde{\mathcal{P}}_A^{i,j}(B, B) &= \tilde{\mathcal{P}}_A \cup \{\langle \bar{u}_{C_i}, \{\bar{c}\} \rangle, \langle \bar{u}_{C_j}, \{\bar{c}\} \rangle\}, \end{aligned}$$

and include a tuple (X, Y) in $\pi_{\{i,j\}}(\bar{S}_A)$ if and only if the problem $\tilde{\mathcal{P}}_A^{i,j}(X, Y)$ has a solution. The relation $\pi_{\{i,j\}}(\bar{S}_B)$ can be found in the same way.

CASE 4. $\mathbb{R}_A \cup \mathbb{R}_B$ is invariant under an affine operation f .

This case is, in fact, impossible. To show this, notice that the relation $\varrho = (\sigma_A \times A^{\uparrow A}) \cup \{\bar{c}\}$ belongs to R_A^+ . Then the relation

$$\tau = \begin{pmatrix} A & A & B \\ A & B & B \end{pmatrix}$$

is expressible by the pp-formula $P_\tau(x_1, x_2) = \exists y_1, y_2, z \tilde{P}_\varrho(y_1, y_2, z) \wedge \tilde{P}_{\sigma_A}(y_1, x_1) \wedge \tilde{P}_{\sigma_A}(y_2, x_2)$, and, therefore, belongs to $\mathbb{R}_A$. However, f does not preserve τ .

Example 5 Let $X = \{0, 1, 2, 3\}$, $Y = \{\bar{0}, \bar{1}, \bar{2}, \bar{3}\}$, $D = \{c\}$. Consider the relations on $A = X \cup D$, $B = Y \cup D$:

$$\varrho_A = \begin{pmatrix} c & c & c & c & c & c & c & c & c & c & c & c \\ 0 & 0 & 2 & 2 & 1 & 1 & 3 & 3 & 0 & 1 & 2 & 3 \\ 1 & 3 & 1 & 3 & 0 & 2 & 0 & 2 & c & c & c & c \end{pmatrix},$$

$$\varrho_B = \begin{pmatrix} \bar{0} & \bar{0} & \bar{2} & \bar{2} & \bar{1} & \bar{1} & \bar{3} & \bar{3} & \bar{0} & \bar{0} & \bar{2} & \bar{2} & \bar{1} & \bar{1} & \bar{3} & \bar{3} & \bar{0} & \bar{1} & \bar{2} & \bar{3} \\ c & c & c & c & c & c & c & c & c & c & c & c & c & c & c & c & c & c & c & c \\ \bar{0} & \bar{2} & \bar{0} & \bar{2} & \bar{1} & \bar{3} & \bar{1} & \bar{3} & \bar{1} & \bar{3} & \bar{0} & \bar{2} & \bar{0} & \bar{2} & c & c & c & c \end{pmatrix}.$$

We are going to show that the amalgam $\{\varrho_A\} \bowtie \{\varrho_B\}$ is tractable. To this end we notice that $\varrho_A [\varrho_B]$ is invariant with respect to $f_A, h_A [f_B, g_B, h_B]$ such that

- $f_A(x, y) = x$ if $x = c$, $f_A(x, y) = y$ if $x \in X$;
 $f_B(x, y) = x$ if $x \in Y$, $f_B(x, y) = y$ if $x = c$;
- $g_B(x, y) = x$ if $x = c$, $g_B(x, y) = y$ if $x \in Y$;
- $h_A(x, y, z) = x -_X y +_X z$ if $x, y, z \in X$, $h_A(x, y, z) = x$ otherwise;
- $h_B(x, y, z) = x -_Y y +_Y z$ if $x, y, z \in Y$, $h_B(x, y, z) = x$ otherwise

where $+_X, +_Y$ denote addition modulo 4 in X, Y respectively. We define the relational clones R_A, R_B to be $\text{Inv}(\{f_A, h_A\})$, $\text{Inv}(\{f_B, g_B, h_B\})$. Clearly, $\varrho_A \in R_A$, $\varrho_B \in R_B$. We show that $R_A \bowtie R_B$ is tractable.

As is easily seen, the operations f_A, h_A preserve the relation X , and f_B, g_B, h_B preserve Y . Therefore, $X \in R_A \subseteq \bar{R}_A$, $Y \in R_B \subseteq \bar{R}_B$, hence, R_A, R_B are not monolithic. It is straightforward to show that every relation from $\mathbb{R}_A \cup \mathbb{R}_B$ is invariant with respect to the semilattice operation f where $f(A, B) = f(B, A) = f(B, B) = B$, $f(A, A) = A$.

By Schaefer's Theorem, $\mathbb{R}_A \cup \mathbb{R}_B$ is tractable, therefore, to show the tractability of $R_A \bowtie R_B$ we have to show the tractability of R_A^+, R_B^+ . For any $\varrho \in R_A$, the operations f_A, h_A preserve the relation $\varrho \cup \{\bar{c}\}$, and, for any $\varrho \in R_B$, f_B, g_B, h_B preserve $\varrho \cup \{\bar{c}\}$. Hence, $R_A^+ = R_A$, $R_B^+ = R_B$.

We sketch a solving algorithm for $\text{CSP}(R_A)$ and leave to the reader verifying its soundness. Let $\mathcal{P} = (V; A; \mathcal{C}) \in \text{CSP}(R_A)$.

- **Set** $W = \{v \in V \mid \text{for any } \langle s, \varrho \rangle \in \mathcal{C} \text{ such that } v \in s, c \in \pi_{\{v\}}(\varrho)\}$, and $U = V - W$.
- Every constraint relation of $\mathcal{P}|_U = (U; X; \mathcal{C}|_U)$ where, for any $\langle s, \varrho \rangle \in \mathcal{C}$, there is $\langle s \cap U, \pi_{s \cap U}(\varrho) \rangle \in \mathcal{C}|_U$ is invariant with respect to $x -_X y +_X z$. **Find** a solution $\psi: U \rightarrow X$ to $\mathcal{P}|_U$ by making use of standard algorithms from number theory.
- **Set** $\varphi(v) = \psi(v)$ if $v \in U$ and $\varphi(v) = c$ otherwise. The mapping $\varphi: V \rightarrow A$ is a solution to $\mathcal{P}$.

The algorithm for $\text{CSP}(R_B)$ is analogous.

References

- [Cohen *et al.*, 1997] D.A. Cohen, P.G. Jeavons, and M. Koubarakis. Tractable disjunctive constraints. In *Proceedings 3rd International Conference on Constraint Programming—CP'97 (Linz, October 1997)*, volume 1330 of *Lecture Notes in Computer Science*, pages 478–490. Springer-Verlag, 1997.
- [Cohen *et al.*, 2000a] D.A. Cohen, P.G. Jeavons, and R.L. Gault. New tractable constraint classes from old. In *Proceedings 6th International Conference on Principles and Practice of Constraint Programming—CP'2000*, volume 1894 of *Lecture Notes in Computer Science*, pages 160–171. Springer-Verlag, 2000.
- [Cohen *et al.*, 2000b] D.A. Cohen, P.G. Jeavons, P. Jonsson, and M. Koubarakis. Building tractable disjunctive constraints. *Journal of the ACM*, 47:826–853, 2000.
- [Cooper, 1989] M.C. Cooper. An optimal k-consistency algorithm. *Artificial Intelligence*, 41:89–95, 1989.
- [Dechter and Pearl, 1989] R. Dechter and J. Pearl. Tree clustering for constraint networks. *Artificial Intelligence*, 38:353–366, 1989.
- [Dechter and van Beek, 1997] R. Dechter and P. van Beek. Local and global relational consistency. *Theoretical Computer Science*, 173(1):283–308, 1997.
- [Freuder, 1990] E.C. Freuder. Complexity of k -tree structured constraint satisfaction problems. In *Proceedings of 8th National Conference on Artificial Intelligence AAAI-90*, pages 4–9, 1990.
- [Gottlob *et al.*, 2000] G. Gottlob, L. Leone, and F. Scarcello. A comparison of structural CSP decomposition methods. *Artificial Intelligence*, 124(2):243–282, 2000.
- [Jeavons *et al.*, 1997] P.G. Jeavons, D.A. Cohen, and M. Gyssens. Closure properties of constraints. *Journal of the ACM*, 44:527–548, 1997.
- [Jeavons *et al.*, 1998] P.G. Jeavons, D.A. Cohen, and M.C. Cooper. Constraints, consistency and closure. *Artificial Intelligence*, 101(1-2):251–265, 1998.
- [Jeavons, 1998a] P.G. Jeavons. Constructing constraints. In *Proceedings 4th International Conference on Constraint Programming—CP'98 (Pisa, October 1998)*, volume 1520 of *Lecture Notes in Computer Science*, pages 2–16. Springer-Verlag, 1998.
- [Jeavons, 1998b] P.G. Jeavons. On the algebraic structure of combinatorial problems. *Theoretical Computer Science*, 200:185–204, 1998.
- [Montanari, 1974] U. Montanari. Networks of constraints: Fundamental properties and applications to picture processing. *Information Sciences*, 7:95–132, 1974.
- [Pöschel and Kalužnin, 1979] R. Pöschel and L.A. Kalužnin. *Funktionen- und Relationenalgebren*. DVW, Berlin, 1979.
- [Schaefer, 1978] T.J. Schaefer. The complexity of satisfiability problems. In *Proceedings 10th ACM Symposium on Theory of Computing (STOC'78)*, pages 216–226, 1978.

On a generalization of triangulated graphs for domains decomposition of CSPs

Assef Chmeiss*

CRIL-CNRS FRE 2499
Université d'Artois - IUT de Lens
Rue de l'Université - SP 16
62307 Lens Cedex, France
chmeiss@cril.univ-artois.fr

Philippe Jégou and Lamia Keddar

LSIS - UMR CNRS 6168
Université d'Aix-Marseille III
Avenue Escadrille Normandie-Niemen
13397 Marseille cedex 20, France
{philippe.jegou, lamia.keddar}@univ.u-3mrs.fr

Abstract

In [Jégou, 1993], a decomposition method has been introduced for improving search efficiency in the area of Constraint Satisfaction Problems. This method is based on properties of micro-structure of CSPs related to properties of triangulated graphs. This decomposition allows to transform an instance of CSP in a collection of sub-problems easier to solve, and then gives a natural and efficient way for a parallel implementation [Habbas *et al.*, 2000].

In this paper, we present a generalization of this approach, which is based on a generalization of triangulated graphs. This generalization allows to define the level of decomposition which can be fixed by a graph parameter. The larger this parameter is, the more level of decomposition that is the number of sub-problems is. As a consequence, we can then define the level of decomposition, with respect to the nature of the parallel configuration used (the number of processors).

First experiments reported here show that this extension increases significantly the advantage of the basic decomposition, already shown in [Habbas *et al.*, 2000].

1 Introduction

Constraint-satisfaction problems (CSPs) involve the assignment of values to variables which are subject to a set of constraints. Examples of CSPs are map coloring, conjunctive queries in a relational databases, line drawings understanding, pattern matching in production rules systems, combinatorial puzzles. . . CSP is known to be a NP-complete problem. So, during last twenty years, many works have been proposed to optimize the classical Backtrack procedure, as constraint propagation, intelligent backtracking or network decompositions.

In [Jégou, 1993], another approach has been proposed which is based on the decomposition of the "micro-structure" of the CSP. The micro-structure of a CSP is the graph defined

by the compatible relations between variable-value pairs: vertices are these pairs, and edges are defined by pairs of compatible vertices. Given the micro-structure of a CSP and using graph properties, the method realizes a pre-processing to simplify the problem with a decomposition of the domains of variables. For a CSP $\mathcal{P}$, the decomposition generates a collection of subproblems $\mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_k$, which is equivalent to the initial problem $\mathcal{P}$. Each subproblem $\mathcal{P}_i$ has the size of its domains less or equal to the size of domains of $\mathcal{P}$, so the time complexity of each $\mathcal{P}_i$ is necessarily less than for the initial CSP. Moreover, given this collection of subproblems, we can separately solve them, or recursively apply again the decomposition.

This decomposition allows to transform an instance of CSP in a collection of subproblems the domains size of which is less than domains size of the CSP. So, these sub-problems are theoretically easier to solve. Moreover, this decomposition defines and then gives a natural and efficient way for a parallel implementation to solve CSP. In [Habbas *et al.*, 2000], an experimental analysis of this approach is presented, realized on a parallel configuration. These experiments show clearly the efficiency of the approach: the decomposition generally offers better results than the results obtained to solving the problems without decomposition. This advantage is related to the natural parallelization of the approach since each subproblem is solved independently. This fact is clear for consistent CSPs. Moreover, if we consider inconsistent CSPs, the results are more interesting: without parallelization, solving the problem after its decomposition is clearly better as solving it without decomposition.

In this paper, we present a generalization of this approach. This approach is based on a generalization of triangulated graphs. This generalization allows to define the level of decomposition which can be fixed by a graph parameter. The larger this parameter is, the more level of decomposition (that is the number of sub-problems) is. As a consequence, we can then define the level of decomposition, with respect to the power of the parallel configuration used.

To define this generalization, we exploit a generalization of triangulated graphs called CSG^k graphs which has been introduced in [Chmeiss and Jégou, 1997]. This class of graphs possesses the same kind of properties as triangulated graphs do. The most important here is the fact that the CLIQUE problem has a polynomial time complexity on these graphs.

*This work has been supported in part by the IUT de Lens, the CNRS and the Region Nord/Pas-de-Calais under the TACT Programme

First experiments reported here show that this extension increases significantly the advantage of the basic decomposition, already shown in [Habbas *et al.*, 2000]. Indeed, we present experiments with two levels of decomposition. The basic decomposition, and the second level (CSG² graphs). The results show that for the second level of decomposition we obtain better results.

The section 2 recalls the decomposition method while section 3 presents the generalization of the decomposition. Experimental results and a discussion about these result are presented in the section 4.

2 Preliminaries

2.1 Constraint Satisfaction Problems

A finite CSP (Constraint Satisfaction Problem) is defined as four tuple $\mathcal{P} = (X, D, C, R)$. X is a set of n variables $x_1, x_2, \dots, x_n$. D is a set of finite domains $D_1, D_2, \dots, D_n$, and C is a set of m constraints. Here we only consider binary CSPs, that is the constraints are defined on pairs of variables $\{x_i, x_j\}$ and will be denoted C_{ij} . To each constraint C_{ij} , we associate a subset of the cartesian product $(D_i \times D_j)$ which is denoted R_{ij} and specifies which values of the variables are compatible with each other. R is the set of all R_{ij} . A *solution* is an assignment of values to variables which satisfies all the constraints. For a CSP $\mathcal{P}$, the pair (X, C) is a graph called the *constraint graph*. Given a CSP, the problem is either to know if there exists any solution, to find solution, or to find all solutions. The first problem is known to be NP-complete.

CSPs are normally solved by different versions of backtrack search. In this case, if d is the size of domains (maximum number of values in domains D_i), the theoretical time complexity of search is then bounded by the size of the search space, that is d^n . Consequently, many works tried to improve the search efficiency. At present time, it seems that the most efficient algorithm is the procedure MAC which has been introduced in [Sabin and Freuder, 1994]. Other approaches use decomposition techniques based on structural properties of the CSP. These methods exploit the fact that the tractability of CSPs is intimately connected to the topological structure of their underlying constraint graph. Moreover, these methods give an upper bound to the complexity of the problem. As example of such decompositions, we have tree-clustering scheme [Dechter and Pearl, 1989]. The time complexity of this method is bounded by d^k where k is induced by the topological features of the constraint graph. Intuitively, more the constraint graph is dense, more the value k is large. For example, if the constraint network is a complete graph, then $k = n$. So, the complexity of these decomposition methods is the same as for classical backtracking, that is d^n . So, in [Jégou, 1993], Jégou proposed an alternative way to decompose CSPs which is not based on the constraint graph but on the *micro-structure* of a CSP. In the sequel, we called this decomposition scheme *TR-Decomposition*.

2.2 TR-Decomposition

Given a binary¹ CSP $\mathcal{P} = (X, D, C, R)$ such that (X, C) is a complete graph, $\mu(\mathcal{P}) = (X_D, C_R)$ is called *micro-structure* of $\mathcal{P}$ and it is a n -partite graph where $X_D = \{(x_i, a) : x_i \in X \text{ and } a \in D_i\}$ and $C_R = \{(x_i, a), (x_j, b) : (a, b) \in R_{ij}\}$.

Note that if (X, C) is not a complete graph, for pairs of variables x_i and x_j such that the constraint C_{ij} does not belong to set C , we consider the universal relation between these variables, that is $R_{ij} = D_i \times D_j$ (all pairs of values are compatible). The decomposition is based on a basic property of the micro-structure:

Theorem 1 Given a CSP $\mathcal{P}$ and its micro-structure $\mu(\mathcal{P})$, then $(a_1, \dots, a_n) \in \text{Solutions}(\mathcal{P})$, that is $(a_1, \dots, a_n)$ is a solution of $\mathcal{P}$, iff $\{(x_1, a_1), \dots, (x_n, a_n)\}$ is a n -clique of $\mu(\mathcal{P})$.

In other words, transforming a CSP as its micro-structure is clearly a polynomial reduction from CSP to *CLIQUE*, which is the problem of finding a clique of a given size belongs to a graph. As a consequence of this fact, as CSP, the *CLIQUE* problem is NP-Complete. Since the problem of finding a n -clique is NP-hard, TR-Decomposition exploits the fact that *triangulated graphs* constitute a polynomial class of instances for this problem. We recall briefly basic properties of these graphs. Given a graph $G = (V, E)$ and an ordering $\sigma = [v_1, \dots, v_n]$ of V , the *successors* of a vertex v_i are the elements of the set $N^+(v_i) = \{v_j \in N(v_i) : i < j\}$ where $N(v) = \{u \in V : \{u, v\} \in E\}$; a vertex v_i is *simplicial* in σ if $N^+(v_i)$ is a clique. The ordering $\sigma = [v_1, \dots, v_n]$ of V is a *perfect elimination ordering* if, for $i = 1, 2, \dots, n$, the vertex v_i is simplicial in the ordering σ . A graph $G = (V, E)$ is triangulated if and only if V admits a perfect elimination ordering. Triangulated graphs satisfies desirable properties. A triangulated graph on n vertices has at most n maximal cliques (a clique is maximal iff it is not included in an other clique) [Fulkerson and Gross, 1965]. Moreover, the problem of finding all maximal cliques in a triangulated graph is in $O(n + m)$ if n is the number of vertices and m the number of edges [Gavril, 1972].

Given the micro-structure of any CSP, it is not possible to immediately use these properties because any micro-structure is not necessary a triangulated graph. So, in adding edges in E , we can build a new graph $\Gamma(G) = (V, E')$ which is triangulated. This addition of edges is called *triangulation*, and can be realized in a linear time in the size of the graph. So, after the triangulation, all maximal cliques of G can be found in linear time. Applied to a CSP $\mathcal{P} = (X, D, C, R)$, the triangulation of its micro-structure $\mu(\mathcal{P}) = (X_D, C_R)$ produces the graph $\Gamma(\mu(\mathcal{P}))$, the set of maximal cliques of which is $Y_1, \dots, Y_k$ ($k \leq d \cdot n$). Every Y_i is a set of values, and then defines a subproblem of $\mathcal{P}$. More formally, given a binary CSP $\mathcal{P} = (X, D, C, R)$, its micro-structure $\mu(\mathcal{P}) = (X_D, C_R)$, and Y a subset of X_D . The CSP induced by Y on

¹Note that in [Hamadi, 1996], TR-Decomposition has been generalized to General CSPs, that is CSPs where constraints can be defined on more than two variables.

$\mathcal{P}$, denoted $\mathcal{P}(Y)$ is defined by:

- $D_Y = \{D_{Y,1}, \dots, D_{Y,n}\} : D_{Y,i} = \{a \in D_i : (x_i, a) \in Y\}$
- $R_{Y,ij} = \{(a, b) \in R_{ij} : (x_i, a), (x_j, b) \in Y\}$
- $\mathcal{P}(Y) = (X, D_Y, C, R_Y)$

Finally, to ensure the validity of the decomposition, presents the theorem below:

Theorem 2 [Jegou, 1993] Given a binary CSP $\mathcal{P}$, its micro-structure $\mu(\mathcal{P})$ and $\{Y_1, \dots, Y_k\}$, the set of the maximal cliques of $\Gamma(\mu(\mathcal{P}))$, then

$$\text{Solutions}(\mathcal{P}) = \bigcup_{i=1}^k \text{Solutions}(\mathcal{P}(Y_i)).$$

To summarize the approach, we recall the algorithm:

Algorithm: TR-Decomposition

Begin

- 1 Build the micro-structure $\mu(\mathcal{P})$
- 2 Triangulation($\mu(\mathcal{P}), \Gamma(\mu(\mathcal{P})), \sigma$)
- 3 MaximalCliques($\Gamma(\mu(\mathcal{P})), \sigma, \{Y_1, \dots, Y_k\}$)
- 4 **for** all Y_i **do**
- 5 **if** Y_i is a covering of all the domains in D
- 6 **then** Solve($\mathcal{P}(Y_i)$)
- 7 **else** $\mathcal{P}(Y_i)$ has no solution
- endfor**

End; { TR-Triangulation }

The steps 1, 2 and 3 can be realized in a linear time w.r.t. the size of the problem $\mathcal{P}$. The procedure *Triangulation* returns a perfect elimination ordering σ which allows to the procedure *MaximalCliques* to find sets Y_i . The step 4, that is the iteration **for**, can be replaced by a parallel execution on independent subproblems $\mathcal{P}(Y_i)$. In this case, if the CSP $\mathcal{P}$ is consistent, the cost is related to the cost of solving the easiest subproblem, while it is the cost of the hardest subproblem for inconsistent CSPs. A such implementation has been realized in [Habbas *et al.*, 2000]. In step 5, a set Y_i is not a covering of all the domains in D if there is a subdomain $D_{Y_i,j}$ induced by Y_i which is the empty set. This subproblem is then trivially inconsistent. In step 6, to solve $\mathcal{P}(Y_i)$, we can use any classical search method such that standard backtracking, Forward-Checking or MAC [Sabin and Freuder, 1994]. Finally, note that the quality of decomposition depends on the quality of the triangulation step.

The last point is the most important. Nevertheless, it's well known that the problem of finding an optimal (w.r.t. the size of the largest induced clique) triangulation is NP-Hard (see [Arnborg *et al.*, 1987]). Several algorithms have been proposed for triangulation. In all cases, the aim is to minimize either the number of added edges or the size of cliques in G' . Three classes of approaches are usable. The first one consists in finding an optimal triangulation which produces a graph the maximum size clique of which has minimum size over all triangulations. [Shoikhet and Geiger, 1997] propose a such algorithm the time complexity of which is exponential. The second is to find a minimal triangulation, that is a triangulation computing a set E_1 such as if for any other triangulation $E_2 \subset E_1$ the graph $G' = (V, E \cup E_2)$ is not

triangulated. Note that a triangulation can be minimal and not optimal. See [Berry, 1999] who presents an algorithm in $O(n(n+m))$. The last one consist in using a triangulation consuming a linear time which does not guarantee the minimality. Nevertheless, the purpose of this paper is not to propose a good triangulation, but to exploit a generalization of triangulated graphs to generalize this decomposition. One of the aims of this generalization is to avoid the problem of finding optimal triangulations.

3 Generalizing TR-Decomposition

3.1 A first basic property

To generalize TR-Decomposition, we can extend theorem 2. Consider a CSP $\mathcal{P} = (X, D, C, R)$ and its micro-structure $\mu(\mathcal{P}) = (X_D, C_R)$. Consider a graph $\Delta(\mu(\mathcal{P})) = (X_D, \delta(C_R))$ where $C_R \subseteq \delta(C_R)$. Consider now the set $\{Z_1, \dots, Z_j\}$ of maximal cliques of $\Delta(\mu(\mathcal{P}))$. Every Z_i is a set of values, and then, as in section 2.2, defines a subproblem of $\mathcal{P}$ which is the CSP induced by Z_i on $\mathcal{P}$ (it is denoted $\mathcal{P}(Z_i)$).

Theorem 3 Given a binary CSP $\mathcal{P}$, its micro-structure $\mu(\mathcal{P})$ and $\{Z_1, \dots, Z_j\}$, the set of the maximal cliques of $\Delta(\mu(\mathcal{P}))$, then $\text{Solutions}(\mathcal{P}) = \bigcup_{i=1}^j \text{Solutions}(\mathcal{P}(Z_i))$.

Proof. Given a binary CSP $\mathcal{P}$, its micro-structure $\mu(\mathcal{P})$ and by theorem 1, we know that each solution of $\mathcal{P}$ is a clique of size n . If we consider $\Delta(\mu(\mathcal{P})) = (X_D, \delta(C_R))$ where $C_R \subseteq \delta(C_R)$, each clique of $\mu(\mathcal{P})$ is included in a maximal clique Z_i of $\Delta(\mu(\mathcal{P}))$. Therefore, each solution of $\mathcal{P}$ appears in the associated subproblem $\mathcal{P}(Z_i)$.

While TR-Decomposition is limited to triangulated graphs, theorem 3 allows to define a larger class of decomposition of domains. Nevertheless, because the number of maximal cliques can be exponential in an arbitrary graph, and then, the number of induced subproblems, we must consider graphs $\Delta(\mu(\mathcal{P}))$ which possess a limited number of maximal cliques as triangulated graphs do. So, we exploit here a class of graphs which is a generalization of triangulated graphs.

3.2 A generalization of triangulated graphs

In [Chmeiss and Jegou, 1997], a new class of graphs, called *CSG^k graphs* has been introduced. These graphs which generalize triangulated graphs, possess the same kind of properties as triangulated graphs, e.g. hereditary property of subgraphs, existence of an elimination scheme, polynomial time recognizing algorithm, and polynomial class of graph for CLIQUE problem. Informally, a *CSG⁰* graph is a complete graph, a *CSG¹* graph is a triangulated graph, and a *CSG²* graph is a graph allowing an elimination vertex scheme where the successors of every vertex induce triangulated graphs. More generally, *CSG^k* graphs are defined inductively.

Definition [Chmeiss and Jegou, 1997] The class of *CSG⁰* Graphs is the class of complete graphs. Given $k > 0$, the class of *CSG^k* graphs is the class of graphs $G = (V, E)$ such that there exists an ordering $\sigma = [v_1, \dots, v_n]$ of V such that

for $i = 1, 2, \dots, n$, the graph $G(N^+(v_i))$ is a CSG^{k-1} Graph. The ordering σ is then called a CSG^k scheme.

It's easy to see that CSG^1 Graphs is the class of triangulated graphs. As for triangulated graphs, there exists a polynomial time algorithm R-CSG^k for recognizing CSG^k graphs; $\forall k \geq 2$, its time complexity is $O((n^2)^{k-1}(n+m))$. In the context of decomposition of micro-structure, the number of maximal cliques is an important criteria. CSG^k graphs inherit of a generalization of the property of Fulkerson and Gross [Fulkerson and Gross, 1965] related to this number since, a CSG^k Graphs with n vertices have less than n^k maximal cliques. Moreover, a CSG^2 graph with n vertices and m edges has at most $n+m$ maximal cliques. Finding these cliques is easy, since, given a constant value k the CLIQUE problem is polynomial on this class of graphs since the time complexity of finding the clique of maximum size in an undirected CSG^k graph is bounded by $O(n^{2(k-1)}(n+m))$.

3.3 TR^k -Decomposition

CSG^k graphs can be exploited as a way to generalizing of TR -Decomposition, defining TR^k -Decomposition as decompositions corresponding to the class of associated graphs. For $k = 1$, TR^k -Decomposition is exactly TR -Decomposition, while for $k = 2$, TR^k -Decomposition, that is TR^2 -Decomposition, it's the decomposition induced by CSG^2 graphs. More generally, TR^k -Decomposition is summarized as:

Algorithm: TR^k -Decomposition

Begin

```

1  Build the micro-structure  $\mu(\mathcal{P})$ 
2   $k$ -Triangulation( $\mu(\mathcal{P}), \Delta^k(\mu(\mathcal{P})), \sigma$ )
3   $k$ -MaximalCliques( $\Delta^k(\mu(\mathcal{P})), \sigma, \{Z_1, \dots, Z_j\}$ )
4  for all  $Z_i$  do
5    if  $Z_i$  is a covering of all the domains in  $D$ 
6    then Solve( $\mathcal{P}(Z_i)$ )
7    else  $\mathcal{P}(Z_i)$  has no solution
  endfor
```

End;

Note that :

- The procedure k -Triangulation produces a graph $\Delta^k(\mu(\mathcal{P}))$ which is a CSG^k graph and returns an ordering σ which allows to the procedure k -MaximalCliques to find all the maximal cliques Z_i of $\Delta^k(\mu(\mathcal{P}))$.
- As for TR -Decomposition the step 4, that is the iteration **for**, can be replaced by a parallel execution on independent subproblems $\mathcal{P}(Z_i)$. In this case, if the CSP $\mathcal{P}$ is consistent, the cost is related to the cost of solving the easiest subproblem, while it is the cost of the hardest subproblem for inconsistent CSPs.
- For steps 5 and 6, remarks given for TR -Decomposition hold.
- Finally, the quality of decomposition depends on the quality of the k -triangulation step.

Because of the time complexity of managing CSG^k graphs, that is $O(n^{2(k-1)}(n+m))$, it seems reasonable to focus our attention on small values of k . So, in the sequel we limit our study to CSG^2 graphs. Therefore, the number of maximal cliques will be bounded now by $k \leq d \cdot n + d^2 \cdot n^2$. Nevertheless, two problems must be addressed here. Finding an efficient algorithm for 2-triangulation, and an operational algorithm to find maximal cliques of a CSG^2 graph.

For 2-triangulation, we can imagine an algorithm the time complexity of which is theoretically bounded by $O(nm(n+m))$. This algorithm consider an ordering on vertices which is obtained by the Maximum Cardinality Search of Tarjan and Yannakakis. Given this ordering, we consider vertices v_i , and then, we run a triangulation on $G(N^+(v_i))$. This first step is in $O(n(n+m))$ but this approach does not guarantee to get a 2-triangulation. Indeed, given two vertices v_i and v_j , with $i < j$, the triangulation of $G(N^+(v_j))$ which is realized after the triangulation of $G(N^+(v_i))$ can add edges that restore the new subgraph $G(N^+(v_i))$ not triangulated. So, a second step of verification and repairing the subgraphs $G(N^+(v_i))$ in adding new edges must be realized. This additionnal work induces a theoretical complexity of $O(nm(n+m))$. Nevertheless, experiments show that this additionnal work will be rarely reached in practical cases (see section 4).

4 Experiments

Our aim in this section is to show the usefulness of the generalized decomposition method comparing to the MAC (Maintaining Arc Consistency) algorithm and the decomposition method by triangulation of the micro-structure and to give an idea about the efficiency of the proposed method. For the generalized decomposition method we will consider the TR^2 -Decomposition, i.e. $k = 2$.

As mentionned in section 3, we will not focus on the triangulation and 2-triangulation algorithms because it is not the principal objective of this paper. In experiments, we will use the Tarjan and Yannakakis's triangulation algorithm [Tarjan *et al.*, 1984]. This algorithm requires a precomputed elimination ordering on the vertices. Then, it adds edges, with respect to the ordering, which maintain the chordality property (i.e. when a vertex is treated, its successors must form a clique). Also, for the 2-triangulation, we will use an algorithm based on the Tarjan and Yannakakis's triangulation one as described in section 3.3. In practice, we have observed that the additionnal step is rarely realized and then the observed time of the 2-triangulation is in $n(n+m)$. So, for CSPs of great size, the CPU time used to decompose the problem is generally not significant with respect to the time of solving the CSP.

We have experimented these methods on random CSPs generated at the phase transition. These CSPs are generated according to the model B generator [Prosser, 1996]. Such a generator admits four parameters:

- the number N of variables
- the common size of the initial domains D
- the proportion $p1$ of constraints in the network (or the number $C = p1 * N \cdot (N - 1)/2$ of constraints)

< N, D, T, C >	#SubProblems			#checks			#nodes		
	TR	TR ²		MAC	TR	TR ²	MAC	TR	TR ²
< 20, 5, 6, 106 >	4	14	Sat (29/50)	7943	5260	2695	23	22	20
			Unsat (21/50)	15219	11888	5355	15	12	6
< 20, 5, 11, 47 >	3	12	Sat (31/50)	2295	1710	1119	21	20	20
			Unsat (19/50)	2892	2189	1500	2	1	1
< 20, 10, 30, 130 >	5	40	Sat (10/50)	101141	72138	21279	57	46	26
			Unsat (40/50)	322397	262212	63768	133	111	27
< 20, 10, 50, 60 >	6	42	Sat (39/50)	20816	10227	5372	28	22	20
			Unsat (11/50)	48294	35760	16044	18	15	6
< 20, 20, 130, 150 >	7	109	Sat (31/50)	8598276	2312841	377929	2641	720	130
			Unsat (19/50)	19178630	16179016	1672898	5825	4958	517
< 20, 20, 201, 85 >	11	131	Sat (17/50)	768460	236799	48223	193	70	27
			Unsat (33/50)	1611125	1187395	252259	339	253	50
< 30, 5, 5, 190 >	3	15	Sat (39/50)	21672	14142	6954	48	39	32
			Unsat (11/50)	67897	60267	19846	72	86	23
< 30, 5, 10, 80 >	3	13	Sat (32/50)	4778	3663	2243	32	31	30
			Unsat (18/50)	6997	6145	4876	6	6	7
< 30, 10, 30, 190 >	5	37	Sat (14/50)	875070	429786	176796	410	214	101
			Unsat (36/50)	1774848	1472370	422599	742	619	178
< 30, 10, 50, 90 >	5	39	Sat (37/50)	51855	32627	11648	49	42	32
			Unsat (13/50)	116232	91314	54609	50	40	20
< 30, 15, 90, 155 >	6	78	Sat (18/50)	2549883	1177789	497667	769	364	158
			Unsat (32/50)	6717878	5442121	1778683	1846	1513	465
< 30, 15, 115, 110 >	7	87	Sat (16/50)	1130136	285589	66599	323	101	42
			Unsat (34/50)	1517131	1314240	408847	371	329	95
< 40, 5, 6, 210 >	3	14	Sat (19/50)	37505	23780	11327	76	60	47
			Unsat (31/50)	62690	51937	26762	58	50	24
< 40, 5, 4, 341 >	3	14	Sat (21/50)	116682	85003	42475	138	111	70
			Unsat (29/50)	291075	272612	119047	244	230	101
< 40, 10, 55, 105 >	5	40	Sat (22/50)	75832	44089	18412	70	54	42
			Unsat (28/50)	142332	126415	74768	48	48	23
< 40, 10, 35, 205 >	5	43	Sat (16/50)	1566087	965645	233136	758	491	137
			Unsat (34/50)	3467369	2810709	1194567	1468	1202	477

Table 1: experimental results

- the proportion $p2$ of forbidden pairs of values in a constraint (or the number $T = p2 * D * D$ of forbidden pairs)

In the following, we will use C and T instead of $p1$ and $p2$. C is called the *density* of the constraint graph and T the *tightness* of the constraints. We have analysed decomposition using the procedure MAC as basic search algorithm (the used heuristic is dom/deg [Bessière and Régin, 1996]). That is MAC has been used before decomposition on the initial problem and after decomposition on each induced subproblem.

In order to have a credible idea on the performance of these methods, we consider several classes of problems. We vary two parameters (the number of variables N and the domains size D). In order to deal with classes near the phase transition, we search the appropriate values of the *tightness* (T) and the *density* (C).

For each value of N , we vary D from small domains size to larger ones. So, we can observe the behavior of the decomposition when D grows. Also, we have remarked that the performance of the decomposition method was not the same for satisfiable and unsatisfiable problems. So, we have chosen to present the results separately.

Results are reported in three tables. Each table represents results for a given number of variables N . As measures of comparison, we used the number of consistency checks (#checks) performed by each of the methods and the number of visited nodes (#nodes). We mention that, the execution time is proportional to the number of checks.

In the tables, results are organized as following : the first column contains the classes of problems represented by the four parameters < N, D, T, C >. For each class, we give results of the different methods for satisfiable (Sat) and unsatisfiable (Unsat) problems separately. So, we have two lines per class. The second and the third columns give the number of subproblems induced by the TR-Decomposition and the TR²-Decomposition respectively. The fourth column indicates the number of satisfiable and unsatisfiable problems over the 50 randomly generated CSPs. The next three columns give the number of consistency checks for MAC, TR-Decomposition and the TR²-Decomposition respectively. The last three columns present the number of visited nodes by the different methods.

As the subproblems induced by the decomposition are independent, we can envisage a parallel implementation. In other words, we can imagine one process by subproblem. So, for satisfiable problems we consider the faster subproblem (the firstly solved subproblem since we have the response the it is satisfiable) while we consider the hardest subproblem for unsatisfiable ones.

Tables show, that decomposition improve the MAC algorithm on all classes. The improvement is more significant for larger domains. Also, the performance of MAC is improved more significantly for denser constraint graphs. If we look at table 1 where $N = 20$, we remark that the TR²-Decomposition is five times faster than MAC for the < 20, 10, 30, 130 >. For classes with $D = 20$ the improvement is more significant. This remark is confirmed when

the constraint graph is denser. For example, for the class $< 20, 20, 130, 150 >$ and for satisfiable problems, the performance ratio between TR^2 -Decomposition and MAC is about 1 to 22. In other words, for each consistency check performed by TR^2 -Decomposition the algorithm MAC performs 22 checks for satisfiable problems. For unsatisfiable problems, the performance ratio is about 1 to 12.

This overall view on experiments shows that the generalized decomposition method is promising and results are encouraging. Finally, we would like to mention the importance of the triangulation (2-triangulation) phase. We think that the efficiency of the decomposition method is related to the quality of the triangulation (2-triangulation). We believe that other triangulation algorithms might offer a better decomposition.

5 Conclusion

In this paper, we have presented a generalization of the decomposition of micro-structure, introduced in [Jegou, 1993]. This decomposition allows to transform an instance of CSP in a collection of sub-problems easier to solve, and then gives a natural and efficient way for a parallel implementation [Habbas *et al.*, 2000]. While the original method is based on triangulated graphs, our generalization is based on a generalization of triangulated graphs called CSG^k graphs. For a given value k , we get a special sub-class of graphs. E.g. CSG^0 graphs are complete graphs and CSG^1 graphs are triangulated graphs. The value k allows to define a particular level of decomposition. So, we have introduced TR^k -Decomposition as the generalized decomposition based on CSG^k graphs.

There are two motivations for this generalization of the decomposition:

- to extend the level of decomposition,
- to be able to fix this level to give an optimal exploitation for the parallel configuration used.

Indeed, it has been experimentally observed in [Habbas *et al.*, 2000] that the first level of decomposition, that is TR^1 -Decomposition, outperforms classical algorithms, and that the more the number of sub-problems is, the more the efficiency of TR^1 -Decomposition is.

Our experimental results confirm these facts. We have studied two levels of decomposition, for $k = 1$ and $k = 2$ and the experiments show that TR^2 -Decomposition outperforms TR^1 -Decomposition. Moreover, the number of sub-problems is greater for $k = 2$ than for $k = 1$.

We have limited our experiments to $k = 2$ because the time complexity of managing CSG^k graphs is related to the value k . More precisely, $O(n^{2(k-1)}(n+m))$ is the time complexity to recognize these graphs. So in future works, we will try to propose efficient algorithms to realize decomposition.

References

- [Arnborg *et al.*, 1987] S. Arnborg, D.G. Corneil and A. Proskurowski. Complexity of finding embeddings in a k -tree. *SIAM J. Disc. Meth.*, 8:277–284, 1987.
- [Bessière and Régin, 1996] C. Bessière and J-C. Régin. MAC and combined heuristics: Two reasons to forsake FC (and CBJ?) on hard problems. In *Proceedings of CP'96, LNCS 1118*, pages 61–75, Cambridge, MA, USA, 1996.
- [Berry, 1999] A. Berry. A Wide-range Efficient Algorithm for Minimal Triangulation. In *Proceedings of SODA'99 SIAM Conference*, USA, 1999.
- [Chmeiss and Jegou, 1997] A. Chmeiss and P. Jegou. A Generalization of Chordal Graphs and the Maximum Clique Problem. *Information Processing Letters*, 62(2):61–66, 1997.
- [Dechter and Pearl, 1989] R. Dechter and J. Pearl. Tree Clustering for Constraint Networks. *Artificial Intelligence*, 38:353–366, 1989.
- [Fulkerson and Gross, 1965] D.R. Fulkerson and O. Gross. Incidence matrices and interval graphs. *Pacific J. Math.*, 15:835–855, 1965.
- [Gavril, 1972] F. Gavril. Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph. *SIAM J. Comput.*, 1(2):180–187, 1972.
- [Habbas *et al.*, 2000] Z. Habbas, D. Singer and R. Krajecki. Domain Decomposition for Parallel Resolution of Constraints Satisfaction Problems with OpenMP. In *Proceedings of the Second European Workshop on OpenMP*, Edinburgh, Scotland, UK, 2000.
- [Hamadi, 1996] R. Hamadi. A domain decomposition method to solve n -ary CSPs. In *Proceedings of the International Workshop on Constraint-Based Reasoning, FLAIRS-Constraints-96*, pages 84–88, Key West Florida, USA, 1996.
- [Jegou, 1993] P. Jegou. Decomposition of domains based on the structure of finite constraints satisfaction problems. In *Proceedings of the National Conference on Artificial Intelligence - AAAI'93*, pages 731–736, Washington DC, USA, July 1993.
- [Prosser, 1996] P. Prosser. An empirical study of phase transition in binary constraint satisfaction problems. *Artificial Intelligence*, 81:81–109, 1996.
- [Sabin and Freuder, 1994] D. Sabin and E. Freuder. Contradicting conventional wisdom in constraint satisfaction. In *Proceedings of ECAI'94*, pages 125–129, Amsterdam, Holland, 1994.
- [Shoikhet and Geiger, 1997] K. Shoikhet and D. Geiger. A practical algorithm for finding optimal triangulation. In *Proceedings of AAAI'97*, pages 185–190, Providence, USA, 1997.
- [Tarjan *et al.*, 1984] R. Tarjan and M. Yannakakis. Simple Linear-Time Algorithms to test Chordality of graphs, test acyclicity of hypergraph and selectively reduce acyclic hypergraph. *SIAM Journal on Computing*, 13(3):566–579, 1984.

A Maximal Tractable Class of Soft Constraints

David Cohen

Computer Science Department
RHUL, UK
d.cohen@rhul.ac.uk

Martin Cooper

IRIT
University of Toulouse III, France
cooper@irit.fr

Peter Jeavons

OUCL
University of Oxford, UK
p.jeavons@comlab.ox.ac.uk

Andrei Krokhin

Computer Science Department
University of Warwick, UK
ak@dcsc.warwick.ac.uk

Abstract

Many optimization problems can be expressed using some form of soft constraints, where different measures of desirability are associated with different combinations of domain values for specified subsets of variables. In this paper we identify a class of soft binary constraints for which the problem of finding the optimal solution is tractable. In other words, we show that for any given set of such constraints, there exists a polynomial time algorithm to determine the assignment having the best overall combined measure of desirability. This tractable class includes many commonly-occurring soft constraints, such as “as near as possible” or “as soon as possible after”, as well as crisp constraints such as “greater than”.

1 Introduction

The constraint satisfaction framework is widely acknowledged as a convenient and efficient way to model and solve a wide variety of problems arising in Artificial Intelligence, including planning and scheduling, image processing and natural language understanding.

In the standard framework a *constraint* is usually taken to be a predicate, or relation, specifying the allowed combinations of values for some fixed collection of variables: we will refer to such constraints here as *crisp* constraints. A number of authors have suggested that the usefulness of the constraint satisfaction framework could be greatly enhanced by extending the definition of a constraint to include also *soft* constraints, which allow different measures of desirability to be associated with different combinations of values [Bistarelli *et al.*, 1999]. In this extended framework a constraint can be seen as a *function*, mapping each possible combination of values to a measure of desirability or undesirability. Finding a solution to a set of constraints then means finding an assignment of values to all of the variables which has the best overall combined desirability measure.

Example 1.1 For example, consider an optimization problem where we have $2n$ variables, $v_1, v_2, \dots, v_{2n}$, and we wish to assign each variable an integer value in the range $1, 2, \dots, n$, subject to the following restrictions:

- Each variable v_i should be assigned a value that is as close as possible to $i/2$.
- Each pair of variables v_i, v_{2i} should be assigned a pair of values that are as similar as possible.

To model this situation we might impose the following soft constraints:

- A unary constraint on each v_i specified by a function ψ_i , where $\psi_i(x) = |x - i/2|^r$ for some $r \geq 1$.
- A binary constraint on each pair v_i, v_{2i} specified by a function δ_r , where $\delta_r(x, y) = |x - y|^r$ for some $r \geq 1$.

We would then seek an assignment to all of the variables which minimizes the sum of these constraint functions,

$$\sum_{i=1}^{2n} \psi_i(v_i) + \sum_{i=1}^n \delta_r(v_i, v_{2i}).$$

□

The cost of allowing additional flexibility in the specification of constraints, in order to model requirements of this kind, is generally an increase in computational difficulty. In the case of crisp constraints there has been considerable progress in identifying classes of constraints which are *tractable*, in the sense that there exists a polynomial time algorithm to determine whether or not any collection of constraints from such a class can be simultaneously satisfied [Feder and Vardi, 1998; Jeavons *et al.*, 1997]. In the case of soft constraints there has been a detailed investigation of the tractable cases for Boolean problems (where each variable has just 2 possible values) [Creignou *et al.*, 2001], but very little investigation of the tractable cases over larger domains¹, even though there are many significant results in the literature on combinatorial optimization which are clearly relevant to this question [Nemhauser and Wolsey, 1988].

In this paper we make use of the idea of a *submodular function* [Nemhauser and Wolsey, 1988; Topkis, 1978] to identify a general class of soft constraints for which there exists a polynomial time solution algorithm. Submodular functions are usually defined as real-valued functions on Boolean tuples ($\equiv$ sets) [Nemhauser and Wolsey, 1988], but we consider

¹The only previous work we have been able to find on non-Boolean tractable soft constraints is [Khatib *et al.*, 2001], which describes a family of tractable soft temporal constraints.

the more general case of functions on tuples over an arbitrary finite domain. We also allow our functions to take infinite values. By establishing a new decomposition result for this general class of binary submodular functions (Theorem 4.4), we obtain a cubic time algorithm to find the optimal assignment for any set of soft constraints defined by such functions.

We give a number of examples to illustrate the many different forms of soft constraint that can be defined using binary submodular functions, and we also show that this class is *maximal*, in the sense that no other form of binary constraint can be added without sacrificing tractability.

2 Definitions

To identify a tractable class of soft constraints we will need to restrict the set of functions that are used to specify constraints. Such a restricted set of possible functions will be called a soft constraint *language*.

Definition 2.1 Let D and E be fixed sets. A *soft constraint language over D with evaluations in E* is defined to be a set of functions, Γ , such that each $\phi \in \Gamma$ is a function from D^k to E , for some $k \in \mathbb{N}$, where k is called the arity of ϕ .

For any given choice of soft constraint language, Γ , we define an associated soft constraint satisfaction problem, which we will call $\text{sCSP}(\Gamma)$, as follows.

Definition 2.2 Let Γ be a soft constraint language over D with evaluations in E . An instance $\mathcal{P}$ of $\text{sCSP}(\Gamma)$ is a triple $\langle V, D, C \rangle$, where:

- V is a finite set of *variables*, which must be assigned values from the set D .
- C is a set of *soft constraints*. Each $c \in C$ is a pair $\langle \sigma, \phi \rangle$ where: σ is a list of variables, called the *scope* of c ; and ϕ is an element of Γ of arity $|\sigma|$, called the *evaluation function* of c .

Note that, for any constraint $c = \langle \sigma, \phi \rangle$, the arity of the constraint is given by $|\sigma|$, the length of the constraint scope. The evaluation function ϕ will be used to specify some measure of desirability or undesirability associated with each possible tuple of values over σ .

To complete the definition of a soft constraint satisfaction problem we need to define how the evaluations obtained from each evaluation function are combined and compared, in order to define what constitutes an optimal overall solution. Several alternative mathematical approaches to this issue have been suggested in the literature:

- In the semiring based approach [Bistarelli *et al.*, 1999], the set of possible evaluations, E , is assumed to be an algebraic structure equipped with two binary operations, satisfying the axioms of a semiring.
- In the valued CSP approach [Bistarelli *et al.*, 1999], the set of possible evaluations E is assumed to be a totally ordered algebraic structure with a top and bottom element and a single monotonic binary operation known as *aggregation*.

For our purposes in this paper we require the same properties as the valued CSP approach, with the additional requirement that the aggregation operator has a partial inverse (so that any evaluation can be “subtracted” from any larger evaluation). For concreteness, we shall simply assume throughout this paper that the set of evaluations E is either the set of non-negative integers together with infinity, or else the set of non-negative real numbers together with infinity. Hence, for any two evaluations $\rho_1, \rho_2 \in E$, the aggregation of ρ_1 and ρ_2 is given by $\rho_1 + \rho_2 \in E$, and when $\rho_1 \geq \rho_2$ we also have $\rho_1 - \rho_2 \in E$. (Note that we set $\infty - \infty = \infty$).

The elements of the set E will be used to represent different measure of undesirability, or *penalties*, associated with different combinations of values. This allows us to complete the definition of a soft constraint satisfaction problem with the following simple definition of a solution to an instance.

Definition 2.3 For any soft constraint satisfaction problem instance $\mathcal{P} = \langle V, D, C \rangle$, an *assignment* for $\mathcal{P}$ is a mapping t from V to D . The *evaluation* of an assignment t , denoted $\Phi_{\mathcal{P}}(t)$, is given by the sum (i.e., aggregation) of the evaluations for the restrictions of t onto each constraint scope, that is,

$$\Phi_{\mathcal{P}}(t) = \sum_{\langle \{v_1, v_2, \dots, v_k\}, \phi \rangle \in C} \phi(t(v_1), t(v_2), \dots, t(v_k)).$$

A *solution* to $\mathcal{P}$ is an assignment with the smallest possible evaluation, and the question is to find a solution.

Example 2.4 For any standard constraint satisfaction problem instance $\mathcal{P}$ with crisp constraints, we can define a corresponding soft constraint satisfaction problem instance $\hat{\mathcal{P}}$ in which the range of the evaluation functions of all the constraints is the set $\{0, \infty\}$. For each crisp constraint c of $\mathcal{P}$, we define a corresponding soft constraint $\hat{c}$ of $\hat{\mathcal{P}}$ with the same scope; the evaluation function of $\hat{c}$ maps each tuple allowed by c to 0, and each tuple disallowed by c to ∞ .

In this case the evaluation of an assignment t for $\hat{\mathcal{P}}$ equals the minimal possible evaluation, 0, if and only if t satisfies all of the crisp constraints in $\mathcal{P}$. $\square$

Note that the problem of finding a solution to a soft constraint satisfaction problem is an NP optimization problem, that is, it lies in the complexity class NPO (see [Creignou *et al.*, 2001] for a formal definition of this class). If there exists a polynomial-time algorithm which finds a solution to all instances of $\text{sCSP}(\Gamma)$, then we shall say that $\text{sCSP}(\Gamma)$ is *tractable*. On the other hand, if there is a polynomial-time reduction from some NP-complete problem to $\text{sCSP}(\Gamma)$, then we shall say that $\text{sCSP}(\Gamma)$ is *NP-hard*.

Example 2.5 Let Γ be a soft constraint language over D , where $|D| = 2$. In this case $\text{sCSP}(\Gamma)$ is a class of Boolean soft constraint satisfaction problems.

If we restrict Γ even further, by only allowing functions with range $\{0, \infty\}$, as in Example 2.4, then $\text{sCSP}(\Gamma)$ corresponds precisely to a standard Boolean crisp constraint satisfaction problem. Such problems are sometimes known as

GENERALIZED SATISFIABILITY problems [Schaefer, 1978]. The complexity of $\text{sCSP}(\Gamma)$ for such restricted sets Γ has been completely characterised, and the six tractable cases have been identified [Schaefer, 1978; Creignou *et al.*, 2001].

Alternatively, if we restrict Γ by only allowing functions with range $\{0, 1\}$, then $\text{sCSP}(\Gamma)$ corresponds precisely to a standard Boolean maximum satisfiability problem, in which the aim is to satisfy the maximum number of crisp constraints. Such problems are sometimes known as MAX-SAT problems [Creignou *et al.*, 2001]. The complexity of $\text{sCSP}(\Gamma)$ for such restricted sets Γ has been completely characterised, and the three tractable cases have been identified (see Theorem 7.6 of [Creignou *et al.*, 2001]).

We note, in particular, that when Γ contains just the single binary function ϕ_{XOR} defined by

$$\phi_{XOR}(x, y) = \begin{cases} 0 & \text{if } x \neq y \\ 1 & \text{otherwise} \end{cases}$$

then $\text{sCSP}(\Gamma)$ corresponds to the MAX-SAT problem for the exclusive-or predicate, which is known to be NP-hard (see Lemma 7.4 of [Creignou *et al.*, 2001]). $\square$

Example 2.6 Let Γ be a soft constraint language over D , where $|D| \geq 3$, and assume that Γ contains just the set of all unary functions, together with the single binary function ϕ_{EQ} defined by

$$\phi_{EQ}(x, y) = \begin{cases} 0 & \text{if } x = y \\ 1 & \text{otherwise} \end{cases}$$

Even in this very simple case it can be shown that $\text{sCSP}(\Gamma)$ is NP-hard [Cohen *et al.*, 2002], by reduction from the MINIMUM MULTITERMINAL CUT problem [Dahlhaus *et al.*, 1994]. $\square$

The examples above indicate that generalizing the constraint satisfaction framework to include soft constraints does indeed increase the computational complexity, in general. For example, the standard 2-SATISFIABILITY problem is tractable, but the soft constraint satisfaction problem involving only the single binary Boolean function, ϕ_{XOR} , defined at the end of Example 2.5, is NP-hard. Similarly, the standard constraint satisfaction problem involving only crisp unary constraints and equality constraints is clearly trivial, but the soft constraint satisfaction problem involving only soft unary constraints and a soft version of the equality constraint, specified by the function ϕ_{EQ} defined at the end of Example 2.6, is NP-hard.

However, in the next two sections we will show that it is possible to identify a large class of functions for which the corresponding soft constraint satisfaction problem is tractable.

3 Generalized interval functions

We begin with a rather restricted class of binary functions, with a very special structure.

Definition 3.1 Let D be a totally ordered set. A binary function, $\phi : D^2 \rightarrow E$ will be called a *generalized interval function* on D if it has the following form:

$$\phi(x, y) = \begin{cases} 0 & \text{if } x < a \vee y > b; \\ \rho & \text{otherwise} \end{cases}$$

for some $a, b \in D$ and some $\rho \in E$. Such a function will be denoted $\eta_{[a,b]}^\rho$.

We can explain the choice of name for these functions by considering the unary function $\eta_{[a,b]}^\rho(x, x)$. This function returns the value ρ if and only if its argument lies in the interval $[a, b]$; outside of this interval it returns the value 0.

We shall write Γ_{GI} to denote the set of all generalized interval functions on D , where $D = \{1, 2, \dots, M\}$ with the usual ordering.

The main result of this section is Corollary 3.6, which states that $\text{sCSP}(\Gamma_{GI})$ is tractable. To establish this result we first define a weighted directed graph² associated with each instance of $\text{sCSP}(\Gamma_{GI})$.

Definition 3.2 Let $\mathcal{P} = \langle V, \{1, \dots, M\}, C \rangle$ be an instance of $\text{sCSP}(\Gamma_{GI})$. We define the weighted directed graph $G_{\mathcal{P}}$ as follows.

- The vertices of $G_{\mathcal{P}}$ are as follows:
 $\{S, T\} \cup \{v_d \mid v \in V, d \in \{0, 1, \dots, M\}\}$.
- The edges of $G_{\mathcal{P}}$ are defined as follows:
 - For each $v \in V$, there is an edge from S to v_M with weight ∞ ;
 - For each $v \in V$, there is an edge from v_0 to T with weight ∞ ;
 - For each $v \in V$ and each $d \in \{1, 2, \dots, M-2\}$, there is an edge from v_d to v_{d+1} with weight ∞ ;
 - For each constraint $\langle \langle v, w \rangle, \eta_{[a,b]}^\rho \rangle \in C$, there is an edge from w_b to v_{a-1} with weight ρ . These edges are called “constraint edges”.

Example 3.3 Let $\mathcal{P} = \langle \{x, y, z\}, \{1, 2, 3, 4\}, C \rangle$ be an instance of $\text{sCSP}(\Gamma_{GI})$ with the following four constraints:

$$\begin{aligned} c_1 &= \langle \langle y, x \rangle, \eta_{[3,4]}^3 \rangle & c_3 &= \langle \langle z, y \rangle, \eta_{[1,3]}^7 \rangle \\ c_2 &= \langle \langle y, z \rangle, \eta_{[4,3]}^2 \rangle & c_4 &= \langle \langle z, z \rangle, \eta_{[2,4]}^\infty \rangle \end{aligned}$$

The corresponding weighted directed graph $G_{\mathcal{P}}$, is shown in Figure 1. $\square$

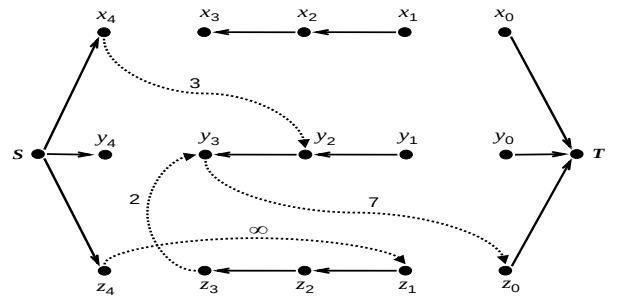


Figure 1: The graph $G_{\mathcal{P}}$ associated with the instance $\mathcal{P}$ defined in Example 3.3.

²This construction was inspired by a similar construction for certain Boolean constraints described in [Khanna *et al.*, 2000].

Any set of edges C in the graph $G_{\mathcal{P}}$ whose removal leaves the vertices S and T disconnected will be called a *cut*. If every edge in C is a constraint edge, then C will be called a *proper cut*. The weight of a cut C is defined to be the sum of the weights of all the edges in C .

Example 3.4 Consider the graph $G_{\mathcal{P}}$ shown in Figure 1. The set $\{\langle y_3, z_0 \rangle\}$ is a proper cut in $G_{\mathcal{P}}$ with weight 7, which is minimal with respect to inclusion. The set $\{\langle x_4, y_2 \rangle, \langle z_3, y_3 \rangle\}$ is a proper cut in $G_{\mathcal{P}}$ with weight 5, which is again minimal with respect to inclusion. $\square$

Proposition 3.5 Let $\mathcal{P}$ be any instance of $\text{sCSP}(\Gamma_{GI})$, and let $G_{\mathcal{P}}$ be the corresponding weighted directed graph.

1. For each minimal proper cut in $G_{\mathcal{P}}$ with weight Φ , there is an assignment for $\mathcal{P}$ with evaluation Φ .
2. For each assignment t for $\mathcal{P}$ with evaluation Φ , there is a proper cut in $G_{\mathcal{P}}$ with weight Φ .

Proof:

1. Let C be any minimal proper cut of the graph $G_{\mathcal{P}}$, and let C_S be the component of $G_{\mathcal{P}} \setminus C$ connected to S . Define the assignment t_C as follows:

$$t_C(v) = \min\{d \mid v_d \in C_S\}$$

(Note that t_C is well-defined because C_S always contains v_M , and never contains v_0 , by construction.)

By the construction of $G_{\mathcal{P}}$, it follows that:

$$t_C(v) > d \Leftrightarrow v_d \notin C_S$$

Now consider any constraint $c = \langle \langle v, w \rangle, \eta_{[a,b]}^{\rho} \rangle$ of $\mathcal{P}$, and its associated edge e in $G_{\mathcal{P}}$. By the definition of generalized interval constraint and the choice of t_C , $\eta_{[a,b]}^{\rho}(t_C(v), t_C(w)) = \rho$ if and only if $v_{a-1} \notin C_S$ and $w_b \in C_S$, and hence if and only if e joins a vertex in C_S to a vertex not in C_S . Since C is minimal, this happens if and only if $e \in C$. Hence, the total weight of the cut C is equal to the evaluation of t_C .

2. Conversely, let t be an assignment to $\mathcal{P}$, and let K be the set of constraints in $\mathcal{P}$ with a non-zero evaluation on t .

Now consider any path from S to T in $G_{\mathcal{P}}$. If we examine, in order, the constraint edges of this path, and assume that each of the corresponding constraints evaluates to 0, then we obtain a sequence of assertions of the following form:

$$\begin{array}{ll} (v_{i_0} > M) \quad \vee \quad (v_{i_1} < a_1) \\ (v_{i_1} > b_2) \quad \vee \quad (v_{i_2} < a_2) & \text{for some } b_2 \geq a_1 \\ \vdots \\ (v_{i_{k-1}} > b_k) \quad \vee \quad (v_{i_k} < a_k) & \text{for some } b_k \geq a_{k-1} \\ (v_{i_k} > b_{k+1}) \quad \vee \quad (v_{i_{k+1}} < 1) & \text{for some } b_{k+1} \geq a_k \end{array}$$

Since the second disjunct of each assertion contradicts the first disjunct of the next, these assertions cannot all hold simultaneously, so one of the corresponding constraints must in fact give a non-zero evaluation on t . Hence, every path from S to T includes at least one edge

corresponding to a constraint from K , and so the edges corresponding to the set K form a cut in $G_{\mathcal{P}}$. Furthermore, by the choice of K , the weight of this cut is equal to the evaluation of t . $\square$

Hence, by using a standard efficient algorithm for the MINIMUM WEIGHTED CUT problem [Goldberg and Tarjan, 1988], we can find an optimal assignment in cubic time.

Corollary 3.6 The time complexity of $\text{sCSP}(\Gamma_{GI})$ is $O(n^3|D|^3)$, where n is the number of variables.

4 Submodular functions

In this section we will consider a rather more general class of functions, as described in [Topkis, 1978].

Definition 4.1 Let D be a totally ordered set. A function, $\phi : D^k \rightarrow E$ is a *submodular function* on D if, for all $\langle a_1, \dots, a_k \rangle, \langle b_1, \dots, b_k \rangle \in D^k$, we have

$$\begin{aligned} \phi(\min(a_1, b_1), \dots, \min(a_k, b_k)) + \phi(\max(a_1, b_1), \dots, \max(a_k, b_k)) \\ \leq \phi(a_1, \dots, a_k) + \phi(b_1, \dots, b_k). \end{aligned}$$

It is easy to check that all unary functions and all generalized interval functions are submodular. For binary functions, the definition of submodularity can be simplified, as follows.

Remark 4.2 Let D be a totally ordered set. A binary function, $\phi : D^2 \rightarrow E$ is submodular if and only if, for all $u < x, v < y \in D$, we have:

$$\phi(u, v) + \phi(x, y) \leq \phi(u, y) + \phi(x, v)$$

Example 4.3 Let D be the set $\{1, 2, \dots, M\}$ with the usual ordering, and consider the binary function π_M , defined by $\pi_M(x, y) = M^2 - xy$.

Note that, for any $u < x, v < y \in D$, we have:

$$\begin{aligned} \pi_M(u, v) + \pi_M(x, y) &= 2M^2 - uv - xy \\ &= 2M^2 - uy - xv - (x - u)(y - v) \\ &\leq \pi_M(u, y) + \pi_M(x, v). \end{aligned}$$

Hence, by Remark 4.2, the function π_M is submodular. $\square$

It follows from Definition 4.1 that the sum of any two submodular functions is submodular. This suggests that in some cases it may be possible to express a submodular function as a sum of simpler submodular functions. For example, for any unary function $\psi : D \rightarrow E$ we have

$$\psi(x) \equiv \sum_{d \in D} \eta_{[d,d]}^{\psi(d)}(x, x).$$

The main result of this section is Theorem 4.4, which states that any *binary* submodular function can also be expressed as a sum of generalized interval functions.

Theorem 4.4 Let D be a totally ordered finite set. A binary function, $\phi : D^2 \rightarrow E$ is submodular if and only if it can be expressed as a sum of generalized interval functions on D .

Proof: By the observations already made, any function ϕ which is equal to a sum of generalized interval functions is clearly submodular.

To establish the converse, we use induction on the *tightness* of ϕ , that is, the number of tuples for which the value of ϕ is non-zero. Details are given in [Cohen *et al.*, 2002]. $\square$

Example 4.5 Consider the binary function π_M on $D = \{1, 2, \dots, M\}$, defined in Example 4.3. When $M = 3$, the values of π_3 are given by the following table:

π_3	1	2	3
1	8	7	6
2	7	5	3
3	6	3	0

Note that:

$$\begin{pmatrix} 8 & 7 & 6 \\ 7 & 5 & 3 \\ 6 & 3 & 0 \end{pmatrix} = \begin{pmatrix} 6 & 6 & 6 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 \\ 3 & 3 & 3 \\ 0 & 0 & 0 \end{pmatrix} + \begin{pmatrix} 2 & 0 & 0 \\ 2 & 0 & 0 \\ 2 & 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 0 \end{pmatrix}.$$

Hence,

$$\begin{aligned} \pi_3(x, y) &= \eta_{[1,1]}^6(x, x) + \eta_{[2,2]}^3(x, x) + \\ &\quad \eta_{[1,1]}^2(y, y) + \eta_{[2,2]}^1(y, y) + \eta_{[2,2]}^1(x, y) + \\ &\quad \eta_{[2,1]}^1(x, y) + \eta_{[3,2]}^1(x, y) + \eta_{[3,1]}^1(x, y). \end{aligned}$$

In general, for arbitrary values of M , we have $\pi_M(x, y) =$

$$\sum_{d=1}^{M-1} \left(\eta_{[d,d]}^{M(M-d)}(x, x) + \eta_{[d,d]}^{M-d}(y, y) + \sum_{e=1}^{M-1} \eta_{[d+1,e]}^1(x, y) \right)$$

We remark that this decomposition is not unique - other decompositions exist, including the symmetric decomposition $\pi_M(x, y) = \pi'_M(x, y) + \pi'_M(y, x)$, where $\pi'_M(x, y) =$

$$\sum_{d=1}^{M-1} \left(\eta_{[d,d]}^{\frac{M^2-d^2}{2}}(x, x) + \eta_{[d+1,d]}^{\frac{1}{2}}(x, y) + \sum_{e=1}^{d-1} \eta_{[d+1,e]}^1(x, y) \right)$$

$\square$

Combining Theorem 4.4 with Corollary 3.6, gives:

Corollary 4.6 For any finite soft constraint language Γ on a finite totally ordered set D , if Γ contains only unary or binary submodular functions, then the time complexity of $\text{sCSP}(\Gamma)$ is $O(n^3|D|^3)$.

The next result shows that the tractable class identified in Corollary 4.6 is maximal.

Theorem 4.7 Let Γ be the set of all binary submodular functions on a totally ordered finite set D , with $|D| \geq 2$.

For any binary function $\psi \notin \Gamma$, $\text{sCSP}(\Gamma \cup \{\psi\})$ is NP-hard.

Proof: The proof is by reduction from $\text{sCSP}(\{\phi_{XOR}\})$ to $\text{sCSP}(\Gamma \cup \{\psi\})$, where ϕ_{XOR} is the binary function defined in Example 2.5. It was pointed out in Example 2.5 that $\text{sCSP}(\{\phi_{XOR}\})$ corresponds to the MAX-SAT problem for the exclusive-or predicate, which is known to be NP-hard [Creignou *et al.*, 2001]. Hence $\text{sCSP}(\Gamma \cup \{\psi\})$ is also NP-hard. Details of the reduction are given in [Cohen *et al.*, 2002]. $\square$

5 Applications

In this section we give a number of examples to illustrate the wide range of soft constraints which can be shown to be tractable using the results obtained in the previous sections.

Definition 5.1 For any k -ary relation R on a set D , we define an associated function, $\phi_R : D^k \rightarrow E$, as follows:

$$\phi_R(x_1, x_2, \dots, x_k) = \begin{cases} 0 & \text{if } \langle x_1, x_2, \dots, x_k \rangle \in R \\ \infty & \text{otherwise.} \end{cases}$$

By Corollary 4.6, any collection of crisp constraints, where each constraint is specified by a relation R for which ϕ_R is unary or binary submodular, can be solved in polynomial time, even when combined with other soft constraints that are also unary or binary submodular.

Example 5.2 The constraint programming language CHIP incorporates a number of constraint solving techniques for arithmetic and other constraints. In particular, it provides a constraint solver for a restricted class of crisp constraints over natural numbers, referred to as *basic constraints* [van Hentenryck *et al.*, 1992]. These basic constraints are of two kinds, which are referred to as “domain constraints” and “arithmetic constraints”. The domain constraints described in [van Hentenryck *et al.*, 1992] are unary constraints which restrict the value of a variable to some specified finite subset of the natural numbers. The arithmetic constraints described in [van Hentenryck *et al.*, 1992] have one of the following forms:

$$\begin{aligned} aX &\neq b & aX &\leq bY + c \\ aX &= bY + c & aX &\geq bY + c \end{aligned}$$

where variables are represented by upper-case letters, and constants by lower case letters, all constants are non-negative real numbers and a is non-zero.

For each of these crisp constraints the associated function given by Definition 5.1 is unary or binary submodular, hence, by Corollary 3.6, any problem involving constraints of this form can be solved in cubic time. Moreover, any other soft constraints with unary or binary submodular evaluation functions can be added to such problems without sacrificing tractability (including the examples below). $\square$

Now assume, for simplicity, that $D = \{1, 2, \dots, M\}$.

Example 5.3 Consider the binary linear function λ defined by $\lambda(x, y) = ax + by + c$, where $a, b \in \mathbb{R}^+$.

This function is submodular and hence, by Corollary 3.6, any collection of such binary linear soft constraints over the discrete set D can be solved in polynomial time. $\square$

Example 5.4 The Euclidean length function $\sqrt{x^2 + y^2}$ is submodular, and can be used to express the constraint that a 2-dimensional point $\langle x, y \rangle$ is “as close to the origin as possible”.

Example 5.5 The following functions are all submodular:

- $\delta_r(x, y) = |x - y|^r$, where $r \in \mathbb{R}, r \geq 1$.
The function δ_r can be used to express the constraint that: “The values assigned to the variables x and y should be as similar as possible”.
- $\delta_r^+(x, y) = (\max(x - y, 0))^r$, where $r \in \mathbb{R}, r \geq 1$.
The function δ_r^+ can be used to express the constraint that: “The value of x is either less than or as near as possible to y ”.
- $\delta_r^\geq(x, y) = \begin{cases} |x - y|^r & \text{if } x \geq y \\ \infty & \text{otherwise} \end{cases}$
where $r \in \mathbb{R}, r \geq 1$.
The function $\delta_r^\geq$ can be used to express the temporal constraint that: “ x occurs as soon as possible after y ”.

Example 5.6 Reconsider the optimization problem defined in Example 1.1. Since ψ_i is unary, and δ_r is binary submodular (Example 5.5), this problem can be solved in cubic time, using the methods developed in this paper.

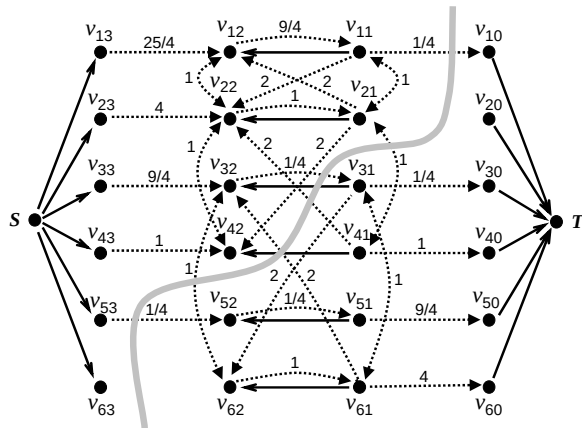
Let $\mathcal{P}$ be the instance with $n = 3$ and $r = 2$. The values of δ_2 are given by the following table:

δ_2	1	2	3
1	0	1	4
2	1	0	1
3	4	1	0

Hence,

$$\delta_2(x, y) = \eta_{[2,1]}^1(x, y) + \eta_{[2,1]}^1(y, x) + \eta_{[3,2]}^1(x, y) + \eta_{[3,2]}^1(y, x) + \eta_{[3,1]}^2(x, y) + \eta_{[3,1]}^2(y, x)$$

Using this decomposition for δ_2 , we can construct the graph $G_{\mathcal{P}}$ corresponding to the instance $\mathcal{P}$, as follows.



The minimum weight of any cut in this graph is $\frac{11}{4}$, and hence the optimal evaluation of any assignment for $\mathcal{P}$ is $\frac{11}{4}$.

One of the several possible cuts with this weight is indicated by the gray line across the graph, which corresponds to the solution $v_1 = 1, v_2 = 1, v_3 = 2, v_4 = 2, v_5 = 3, v_6 = 3$.

References

- [Bistarelli et al., 1999] S. Bistarelli, U. Montanari, F. Rossi, T. Schiex, G. Verfaillie, and H. Fargier. Semiring-based CSPs and valued CSPs: Frameworks, properties, and comparison. *Constraints*, 4:199–240, 1999.
- [Cohen et al., 2002] D. Cohen, M. Cooper, P. Jeavons, and A. Krokhin. A tractable class of soft constraints. Technical Report CSD-TR-02-14, Computer Science Department, Royal Holloway, University of London, Egham, Surrey, UK, December 2002.
- [Creignou et al., 2001] N. Creignou, S. Khanna, and M. Sudan. *Complexity Classifications of Boolean Constraint Satisfaction Problems*, volume 7 of *SIAM Monographs on Discrete Mathematics and Applications*. 2001.
- [Dahlhaus et al., 1994] E. Dahlhaus, D.S. Johnson, C.H. Papadimitriou, P.D. Seymour, and M. Yannakakis. The complexity of multiterminal cuts. *SIAM Journal on Computing*, 23(4):864–894, 1994.
- [Feder and Vardi, 1998] T. Feder and M.Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: A study through Datalog and group theory. *SIAM Journal of Computing*, 28:57–104, 1998.
- [Goldberg and Tarjan, 1988] A. Goldberg and R.E. Tarjan. A new approach to the maximum flow problem. *Journal of the ACM*, 35:921–940, 1988.
- [Jeavons et al., 1997] P.G. Jeavons, D.A. Cohen, and M. Gyssens. Closure properties of constraints. *Journal of the ACM*, 44:527–548, 1997.
- [Khanna et al., 2000] S. Khanna, M. Sudan, L. Trevisan, and D. Williamson. The approximability of constraint satisfaction problems. *SIAM Journal on Computing*, 30(6):1863–1920, 2000.
- [Khatib et al., 2001] L. Khatib, P. Morris, R. Morris, and F. Rossi. Temporal constraint reasoning with preferences. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI-01)*, pages 322–327, Seattle, USA, 2001.
- [Nemhauser and Wolsey, 1988] G.L. Nemhauser and L.A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley & Sons, 1988.
- [Schaefer, 1978] T.J. Schaefer. The complexity of satisfiability problems. In *Proceedings 10th ACM Symposium on Theory of Computing, STOC’78*, pages 216–226, 1978.
- [Topkis, 1978] D.M. Topkis. Minimizing a submodular function on a lattice. *Operations Research*, 26:305–321, 1978.
- [van Hentenryck et al., 1992] P. van Hentenryck, Y. Deville, and C-M. Teng. A generic arc-consistency algorithm and its specializations. *Artificial Intelligence*, 57:291–321, 1992.

Reasoning about soft constraints and conditional preferences: complexity results and approximation techniques*

C. Domshlak
Dept. of Computer Science
Cornell University
Ithaca, NY, USA
dcarmel@cs.cornell.edu

F. Rossi and K. B. Venable
Dept. of Mathematics
University of Padova
Padova, Italy
{frossi,kvenable}@math.unipd.it

T. Walsh
Cork Constraint Computation Centre
University College Cork
Cork, Ireland
tw@4c.ucc.ie

Abstract

Many real life optimization problems contain both hard and soft constraints, as well as qualitative conditional preferences. However, there is no single formalism to specify all three kinds of information. We therefore propose a framework, based on both CP-nets and soft constraints, that handles both hard and soft constraints as well as conditional preferences efficiently and uniformly. We study the complexity of testing the consistency of preference statements, and show how soft constraints can faithfully approximate the semantics of conditional preference statements whilst improving the computational complexity.

1 Introduction and Motivation

Representing and reasoning about preferences is an area of increasing interest in theoretical and applied AI. In many real life problems, we have both hard and soft constraints, as well as qualitative conditional preferences. For example, in a product configuration problem, the producer may have hard and soft constraints, while the user has a set of conditional preferences. Until now, there has been no single formalism which allows all these different kinds of information to be specified efficiently and reasoned with effectively. For example, soft constraint solvers [Bistarelli *et al.*, 1997; Schiex *et al.*, 1995] are most suited for reasoning about the hard and soft constraints, while CP-nets [Boutilier *et al.*, 1999] are most suited for representing qualitative conditional preference statements. In this paper, we exploit a connection between these two approaches, and define a framework based on both CP-nets and soft constraints which can efficiently handle both constraints and preferences.

*This research was partially funded by AFOSR, grant F49620-01-1-0076 (Intelligent Information Systems Institute, Cornell University) and F49620-01-1-0361 (MURI grant on Cooperative Control of Distributed Autonomous Vehicles in Adversarial Environments), DARPA, F30602-00-2-0530 (Controlling Computational Cost: Structure, Phase Transitions and Randomization) and F30602-00-2-0558 (Configuring Wireless Transmission and Decentralized Data Processing for Generic Sensor Networks), the Italian MIUR projects NAPOLI and COVER, the ASI (Italian Space Agency) project ARISCOM and Science Foundation Ireland.

Soft constraints [Bistarelli *et al.*, 1997; Schiex *et al.*, 1995] are one of the main methods for dealing with preferences in constraint optimization. Each assignment to the variables of a constraint is annotated with a level of its desirability, and the desirability of a complete assignment is computed by a combination operator applied to the “local” preference values. Whilst soft constraints are very expressive, and have a powerful computational machinery, they are not good at modeling and solving the sort of conditional preference statements that occur in the real world. Moreover, soft constraints are based on *quantitative* measures of preference, which tends to make preference elicitation more difficult.

Qualitative user preferences have been widely studied in decision-theoretic AI [Doyle and Thomason, 1999]. Of particular interest are CP-nets [Boutilier *et al.*, 1999]. These model statements of qualitative and conditional preference such as “I prefer a red dress to a yellow dress”, or “If the car is convertible, I prefer a soft top to a hard top”. These are interpreted under the *ceteris paribus* (that is, “all else being equal”) assumption. Preference elicitation in such a framework is intuitive, independent of the problem constraints, and suitable for naive users. However, the Achilles heel of CP-nets and other sophisticated qualitative preference models [Lang, 2002] is the complexity of reasoning with them [Domshlak and Brafman, 2002; Boutilier *et al.*, 2002].

Motivated by a product configuration application [Sabin and Weigel, 1998], we have developed a framework to reason simultaneously about qualitative conditional preference statements and hard and soft constraints. In product configuration, the producer has hard (e.g., component compatibility) and soft (e.g., supply time) constraints, while the customer has preferences over the product features. We first investigate the complexity of reasoning about qualitative preference statements, addressing in particular preferential consistency. To tackle the complexity of preference reasoning, we then introduce two approximation schemes based on soft constraints.

To the best of our knowledge, this work provides the first connection between the CP-nets and soft constraints machinery. In addition, for product configuration problems or any problem with both hard and soft quantitative constraints as well as qualitative conditional preferences, this framework lets us treat the three kinds of information in a unifying environment. Finally, we compare the two approximations in terms of both expressivity and complexity.

2 Formalisms for Describing Preferences

2.1 Soft constraints

There are many formalisms for describing *soft constraints*. We use the c-semi-ring formalism [Bistarelli *et al.*, 1997], which is equivalent to the valued-CSP formalism when total orders are used [Bistarelli *et al.*, 1996], as this generalizes many of the others. In brief, a soft constraint associates each instantiation of its variables with a value from a partially ordered set. We also supply operations for combining ($\times$) and comparing ($+$) values. A semi-ring is a tuple $\langle A, +, \times, 0, 1 \rangle$ such that: A is a set and $0, 1 \in A$; $+$ is commutative, associative and 0 is its unit element; $\times$ is associative, distributes over $+$, 1 is its unit element and 0 is its absorbing element. A *c-semi-ring* is a semi-ring $\langle A, +, \times, 0, 1 \rangle$ in which $+$ is idempotent, 1 is its absorbing element and $\times$ is commutative.

Let us consider the relation $\leq$ over A such that $a \leq b$ iff $a + b = b$. Then $\leq$ is a partial order, $+$ and $\times$ are monotone on $\leq$, 0 is its minimum and 1 its maximum, $\langle A, \leq \rangle$ is a complete lattice and, for all $a, b \in A$, $a + b = \text{lub}(a, b)$. Moreover, if $\times$ is idempotent: $+$ distributes over $\times$; $\langle A, \leq \rangle$ is a complete distributive lattice and $\times$ its glb. Informally, the relation $\leq$ compares semi-ring values and constraints. When $a \leq b$, we say that b is *better than* a . Given a semi-ring $S = \langle A, +, \times, 0, 1 \rangle$, a finite set D (variable domains) and an ordered set of variables V , a *constraint* is a pair $\langle \text{def}, \text{con} \rangle$ where $\text{con} \subseteq V$ and $\text{def} : D^{|\text{con}|} \rightarrow A$. A constraint specifies a set of variables, and assigns to each tuple of values of these variables an element of the semi-ring.

A *soft constraint satisfaction problem* (SCSP) is given by a set of soft constraints. For example, a classical CSP is an SCSP with the c-semi-ring $S_{\text{CSP}} = \langle \{\text{false}, \text{true}\}, \vee, \wedge, \text{false}, \text{true} \rangle$, a fuzzy CSP [Schiex, 1992] is an SCSP with the c-semi-ring $S_{\text{FCSP}} = \langle [0, 1], \max, \min, 0, 1 \rangle$, and probabilistic and weighted CSPs are SCSPs with the c-semi-rings $S_{\text{prob}} = \langle [0, 1], \max, \times, 0, 1 \rangle$ and $S_{\text{weight}} = \langle \mathbb{R}, \min, +, 0, +\infty \rangle$, respectively. A solution to an SCSP is a complete assignment to its variables. The preference value associated with a solution is obtained by multiplying the preference values of the projections of the solution to each constraint. One solution is better than another if its preference value is higher in the partial order. Finding an optimal solution for an SCSP is an NP-complete problem. On the other hand, given two solutions, checking whether one is preferable is easy: we compute the semi-ring values of the two solutions and compare the resulting values.

2.2 CP-nets

Soft constraints are the main tool for representing and reasoning about preferences in constraint satisfaction problems. However, they require the choice of a semi-ring value for each variable assignment in each constraint. They are therefore a *quantitative* method for expressing preferences. In many applications, it is more natural for users to express preferences via generic qualitative (usually partial) preference relations over variable assignments. For example, it is often more intuitive for the user to say “I prefer red wine to white wine”, rather than “Red wine has preference 0.7 and white wine has

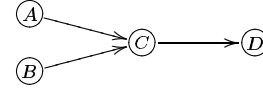


Figure 1: The CP-net graph for the example.

preference 0.4”. Of course, the former statement provides less information, but it does not require careful selection of preference values to maintain consistency. Moreover, soft constraints do not naturally represent conditional preferences, as in “If they serve meat, then I prefer red wine to white wine”. It is easy to see that both qualitative statements and conditions are essential ingredients in many applications.

CP-nets [Boutilier *et al.*, 1999] are a graphical model for compactly representing conditional and qualitative preference relations. They exploit conditional preferential independence by structuring a user’s preferences under the *ceteris paribus* assumption. Informally, CP-nets are sets of *conditional ceteris paribus* (CP) preference statements. For instance, the statement “I prefer red wine to white wine if meat is served.” asserts that, given two meals that differ *only* in the kind of wine served *and* both containing meat, the meal with a red wine is preferable to the meal with a white wine. Many philosophers (see [Hansson, 2001] for an overview) and AI researchers [Doyle and Wellman, 1994], have argued that most of our preferences are of this type.

CP-nets bear some similarity to Bayesian networks, as both utilize directed acyclic graphs where each node stands for a domain variable, and assume a set of features $\mathbf{F} = \{X_1, \dots, X_n\}$ with finite, discrete domains $\mathcal{D}(X_1), \dots, \mathcal{D}(X_n)$ (these play the same role as variables in soft constraints). During preference elicitation, for each feature X_i , the user is asked to specify a set of *parent* features $\text{Pa}(X_i)$ that can affect her preferences over the values of X_i . This information is used to create the graph of the CP-net in which each node X_i has $\text{Pa}(X_i)$ as its immediate predecessors. Given this structural information, the user is asked to explicitly specify her preference over the values of X_i for *each complete assignment* on $\text{Pa}(X_i)$, and this preference is assumed to take the form of total [Boutilier *et al.*, 1999] or partial [Boutilier *et al.*, 2002] order over $\mathcal{D}(X)$. These conditional preferences over the values of X_i are annotated with the node X_i in the CP-net. For example, consider a CP-net with the graph given in Figure 1, and with the preference statements as follows: $a \succ \bar{a}$, $b \succ \bar{b}$, $(a \wedge b) \vee (\bar{a} \wedge \bar{b}) : c \succ \bar{c}$, $(a \wedge \bar{b}) \vee (\bar{a} \wedge b) : \bar{c} \succ c$, $c : d \succ \bar{d}$, $\bar{c} : \bar{d} \succ d$. Here, statement $a \succ \bar{a}$ represents the unconditional preference of the user for $A = a$ over $A = \bar{a}$, while statement $c : d \succ \bar{d}$ represents that the user prefers $D = d$ to $D = \bar{d}$, given that $C = c$.

Several types of queries can be asked about CP-nets. First, given a CP-net N , one might be interested in finding an optimal assignment to the features of N . For acyclic CP-nets, such a query is answerable in linear time [Boutilier *et al.*, 1999]. Second, given a CP-net N and a pair of complete assignments α and β , one might be interested in determining whether $\alpha \succ \beta$, i.e. α is preferred to β . Unfortunately, this query is NP-hard even for acyclic CP-nets [Domshlak and Brafman, 2002] though some tractable special cases do exist.

3 Consistency and Satisfiability

Given a set of preference statements Ω extracted from a user, we might be interested in testing *consistency* of the induced preference relation. In general, there is no single notion of preferential consistency [Hansson, 2001]. In [Boutilier *et al.*, 1999], a CP-net N was considered consistent iff the partial ordering $\succ$ induced by N is *asymmetric*, i.e. there exist at least one total ordering of the outcomes consistent with $\succ$. However, in many situations, we can ignore cycles in the preference relation, as long as these do not prevent a rational choice, i.e. there exist an outcome that is not dominated by any other outcome. In what follows, we refer to this as *satisfiability*¹. It is easy to see that satisfiability is strictly weaker than asymmetry, and that asymmetry implies satisfiability. We will consider two cases: When the set Ω of preference statements induces a CP-net and, more generally, when preferences can take any form (and may not induce a CP-net).

When Ω defines an acyclic CP-net, the partial order induced by Ω is asymmetric [Boutilier *et al.*, 1999]. However, for cyclic CP-nets, asymmetry is not guaranteed. In the more general case, we are given a set Ω of conditional preference statements without any guarantee that they define a CP-net. Let the *dependence graph* of Ω be defined similarly to the graphs of CP-nets: the nodes stand for problem features, and a directed arc goes from X_i to X_j iff Ω contains a statement expressing preference on the values of X_j conditioned on the value of X_i . For example, the set $\Omega = \{a : b \succ \bar{b}, a \wedge c : \bar{b} \succ b\}$ does not induce a CP-net (the two conditionals are not mutually exclusive), and the preference relation induced by Ω is not asymmetric, despite the fact that the dependence graph of Ω is acyclic.

Note that while asymmetry implies satisfiability, the reverse does not hold in general. For example, the set Ω above is not asymmetric, but it is satisfiable (the assignment $a\bar{b}$ is undominated). Given such a satisfiable set of statements, we can prompt the user with one of the undominated assignments without further refinement of its preference relation. Theorem 1 shows that, in general, determining satisfiability of a set of statements is NP-complete. On the other hand, even for CP-nets, determining asymmetry is not known to be in NP [Domshlak and Brafman, 2002].

Theorem 1 SATISFIABILITY of a set of conditional preference statements Ω is NP-complete.

Proof: Membership in NP is straightforward, as an assignment is a polynomial-size witness that can be checked for non-dominance in time linear in the size of Ω . To show hardness, we reduce 3-SAT to our problem: Given a 3-cnf formula F , for each clause $(x \vee y \vee z) \in F$ we construct the conditional preference statement: $\bar{x} \wedge \bar{y} : z \succ \bar{z}$. This set of conditional preferences is satisfiable iff the original original formula F is satisfiable. $\square$

¹In preference logic [Hansson, 2001], these notions of “consistency as satisfiability” and “consistency as asymmetry” correspond to the notions of *eligibility* and *restrictable eligibility*, respectively. However, we will use the former terms as they seem more intuitive.

While testing satisfiability is hard in general, Theorem 2 presents a wide class of statement sets that can be tested for satisfiability in polynomial time.

Theorem 2 A set of conditional preference statements Ω , whose dependency graph is acyclic and has bounded node in-degree can be tested for satisfiability in polynomial time.

Proof: The proof is constructive, and the algorithm is as follows: First, for each feature $X \in \mathbf{V}$, we construct a table T_X with an entry for each assignment $\pi \in \mathcal{D}(Pa(X))$, where each entry $T_X[\pi]$ contains all the values of X that are not dominated given Ω and π . Subsequently, we remove all the empty entries. For example, let A , B and C be a set of boolean problem features, and let $\Omega = \{c \succ \bar{c}, a : b \succ \bar{b}, a \wedge c : \bar{b} \succ b\}$. The corresponding table will be as follows:

Feature	π	Values
T_A	$\emptyset$	$\{a, \bar{a}\}$
T_C	$\emptyset$	$\{c\}$
T_B	$a \wedge \bar{c}$	$\{b\}$
	$\bar{a} \wedge \bar{c}$	$\{b, \bar{b}\}$
	$\bar{a} \wedge c$	$\{b, \bar{b}\}$

Observe that the entry $T_B[a \wedge c]$ has been removed, since, given $a \wedge c$, b and $\bar{b}$ are dominated according to the statements $a \wedge c : \bar{b} \succ b$ and $a : b \succ \bar{b}$, respectively. Since the in-degree of each node X in the dependence graph of Ω is bounded by a constant k (i.e. $|Pa(X)| \leq k$), these tables take space and can be constructed in time $O(n2^k)$. Given such tables for all the features in $\mathbf{V}$, we traverse the dependence graph of Ω in a topological order of its nodes, and for each node X being processed we remove all the entries in T_X that are not “supported” by (already processed) $Pa(X)$: An entry $T_X[\pi]$ is not supported by $Pa(X)$ if there exists a feature $Y \in Pa(X)$ such that the value provided by π to Y appears in no entry of T_Y . For instance, in our example, the rows corresponding to $a \wedge \bar{c}$ and $\bar{a} \wedge \bar{c}$ will be removed, since $\bar{c}$ does not appear in the (already processed) table of C . Now, if the processing of a feature X results in $T_X = \emptyset$, then Ω is not satisfiable. Otherwise, any assignment to $\mathbf{V}$ consistent with the processed tables will be non-dominated with respect to Ω . $\square$

Note that, for sets of preference statements with cyclic dependence graphs, SATISFIABILITY remains hard even if the in-degree of each node is bounded by $k \geq 6$, since 3-SAT remains hard even if each variable participates in at most three clauses of the formula (the proof of Theorem 1). However, when at most one condition is allowed in each preference statement, and the features are boolean, then SATISFIABILITY can be reduced to 2-SAT, and thus tested in polynomial time. Further study of additional tractable cases is clearly of both theoretical and practical interest.

4 Approximating CP-nets with Soft Constraints

In addition to testing consistency and determining preferentially optimal outcomes, we can be interested in the *preferential comparison* of two outcomes. Unfortunately, determining dominance between a pair of outcomes with respect to a set of qualitative preferential statements under the *ceteris paribus*

assumption is PSPACE-complete in general [Lang, 2002], and is NP-hard even for acyclic CP-nets [Domshlak and Brafman, 2002]. However, given a set Ω of preference statements, instead of using a preference relation $\succ$ induced by Ω , one can use an approximation $\gg$ of $\succ$, achieving tractability while sacrificing precision to some degree. Clearly, different approximations $\gg$ of $\succ$ are not equally good, as they can be characterized by the precision with respect to $\succ$, time complexity of generating $\gg$, and time complexity of comparing outcomes with respect to $\gg$. In addition, it is vital that $\gg$ faithfully extends $\succ$ (i.e. $\alpha \succ \beta$ should entail $\alpha \gg \beta$). We call this *information preserving*. Another desirable property of approximations is that of preserving the *ceteris paribus* property (we call this the *cp-condition* for short).

For acyclic CP-nets, two approximations that are information preserving have been introduced, both comparing outcomes in time linear in the number of features. The first is based on the relative position of the features in the CP-net graph [Boutilier *et al.*, 2002]. This approximation does not require any preprocessing of the CP-net. However, it is problematic when there are hard constraints. The second, based on UCP-nets [Boutilier *et al.*, 2001], can be used as a quantitative approximation of acyclic CP-nets. UCP-nets resemble weighted CSPs, and thus they can be used in constraint optimization using the soft constraints machinery. However, generating UCP-nets is exponential in the size of CP-net node's Markov family², and thus in the CP-net node out-degree. An additional related work is described in [McGeachie and Doyle, 2002], where a numerical value function is constructed using graph-theoretic techniques by examining the graph of the preference relation induced by a set of preference statements. Note that this framework is also computationally hard, except for some special cases.

Here we study approximating CP-nets via soft constraints (SCSPs). This allows us to use the rich machinery underlying SCSPs to answer comparison queries in linear time. Moreover, this provides us a uniform framework to combine user preferences with both hard and soft constraints. Given an acyclic CP-net, we construct a corresponding SCSP in two steps. First, we build a constraint graph, which we call *SC-net*. Second, we compute the preferences and weights for the constraints in the SC-net, and this computation depends on the actual semi-ring framework being used. Here we present and discuss two alternative semi-ring frameworks, based on *min+* and *SLO* (Soft constraint Lexicographic Ordering) semi-rings, respectively. In both cases, our computation of preferences and weights ensures information preserving and satisfies the *cp-condition*. We illustrate the construction of the SCSP using the example in Figure 2, which continues our running example from Figure 1.

Given a CP-net N , the corresponding SC-net N_c has two types of nodes: First, each feature $X \in N$ is represented in N_c by a node V_X that stands for a SCSP variable with $\mathcal{D}(V_X) = \mathcal{D}(X)$. Second, for each feature $X \in N$, such that $|Pa(X)| \geq 2$, we have a node $V_{Pa(X)} \in N_c$, with $\mathcal{D}(V_{Pa(X)}) = \prod_{Y \in Pa(X)} \mathcal{D}(Y)$. Edges in N_c correspond to

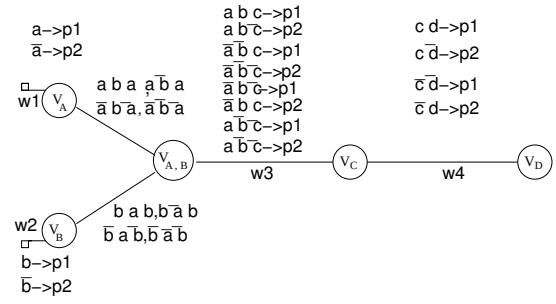


Figure 2: An SC-net.

hard and soft constraints, where the latter are annotated with weights. Each node V_X corresponding to an “independent feature” $X \in N$ has an incoming (source-less) soft constraint edge (e.g., see V_A and V_B). For each node V_X corresponding to a “single-parent” feature $X \in N$ with $Pa(X) = \{Y\}$, we have a soft constraint edge between X and Y (e.g., see V_D). Finally, for each node V_X such that $|Pa(X)| \geq 2$, we have (i) hard constraint edges between $V_{Pa(X)}$ and each $Y \in Pa(X)$ to ensure consistency (e.g., the edges between $V_{A,B}$ and both V_A and V_B), and (ii) a soft constraint edge between $V_{Pa(X)}$ and V_X (e.g., the edge between $V_{A,B}$ and V_C).

To assign preferences to variable assignments in each soft constraint, each soft constraint c (between $V_{Pa(X)}$ and V_X) is associated with two items: w_c , a real number which can be interpreted as a weight (will be defined in the next section), and $P_c = \{p_1, \dots, p_{|\mathcal{D}(V_X)|}\}$, a set of reals which can be interpreted as “quantitative levels of preference”. We will see in the next section how to generate the preference for each assignment to the variables of c , depending on the chosen semiring. In any case, each preference will be obtained by combining (via multiplication over naturals) the weight of the constraint w_c and one of the elements of P_c .

4.1 Weighted soft constraints

The weighted SCSP is based on the *min+* semi-ring $SWCSP = \langle R_+, \min, +, +\infty, 0 \rangle$. We assign preferences using real positive numbers (or penalties) and prefer assignments with smaller total penalty (i.e. the sum of all local penalties). In a soft constraint c on $V_{Pa(X)}$ and V_X , there are $|\mathcal{D}(V_X)|$ penalties. Without loss of generality, we assume they range between 0 and $|\mathcal{D}(V_X)| - 1$, that is, $p_1 = 0, \dots, p_{|\mathcal{D}(V_X)|} = |\mathcal{D}(V_X)| - 1$. In our example, since all variables are binary, there are only two penalties i.e., $p_1 = 0$ and $p_2 = 1$, in all the constraints.

To ensure the *cp-condition*, similar to [Boutilier *et al.*, 2001], we need to ensure that each variable dominates its children. We therefore set the minimum penalty on a variable to be greater than the sum of the maximum penalties of the children. In Figure 3 we show the pseudocode for the algorithm to compute the weights. In this code, $w(V_X)$ represents the weight of the soft constraint c between $V_{Pa(X)}$ and V_X .

Considering our example, let $\{D, C, B, A\}$ be the reverse topological ordering obtained in line 2. Therefore, the first soft constraint to be processed is the one between V_C and V_D . Since D has no children in N , in line 5 we assign $w(V_D)$ to

²Markov family of a node X contains X , its parents and children, and the parents of its children.

1. Next, we process the soft constraint between $V_{A,B}$ and V_C : V_D is the only child of V_C , hence $w(V_C) = w(V_D) \times \mathcal{D}(V_D) = 1 \times 2 = 2$. Subsequently, since V_C is the only child of both V_A and V_B , we assign $w(V_A) = w(V_B) = w(V_C) \times |\mathcal{D}(V_C)| = 2 \times 2 = 4$.

Now, consider two outcomes $o_1 = abcd$ and $o_2 = a\bar{b}cd$. The total penalty of o_1 is $(w(V_A) \times p_1) + (w(V_B) \times p_1) + (w(V_C) \times p_1) + (w(V_D) \times p_1) = 0$, since $p_1 = 0$, while the total penalty of o_2 is $(w(V_A) \times p_1) + (w(V_B) \times p_2) + (w(V_C) \times p_2) + (w(V_D) \times p_1) = (4 \times 1) + (2 \times 1) = 6$ since $p_2 = 1$. Therefore, we can conclude that o_1 is better than o_2 since $\min(0, 6) = 0$.

We now prove that our algorithm for weight computation ensures the cp-condition on the resulting set of soft constraints, and this also implies preserving the ordering information with respect to the original CP-net.

Theorem 3 *The SC-net based weighted SCSP N_c , generated from an acyclic CP-net N , is an information preserving approximation of N , i.e. for each pair of outcomes α, β we have $\alpha \succ \beta \Rightarrow \alpha >_{\min+} \beta$.*

Proof: Due to the CP-net semantics, it is enough to show that, for each variable $X \in N$, each assignment $\mathbf{u}$ on $Pa(X)$, and each pair of values $x_1, x_2 \in \mathcal{D}(X)$, if CP-net specifies that $u : x_1 \succ x_2$, then we have $x_1 \mathbf{u} \mathbf{y} >_{\min+} x_2 \mathbf{u} \mathbf{y}$, for all assignments $\mathbf{y}$ on $\mathbf{Y} = \mathbf{V} - \{X\} \cup Pa(X)$. By definition, $x_1 \mathbf{u} \mathbf{y} >_{\min+} x_2 \mathbf{u} \mathbf{y}$ iff $\sum_{s \in S} p'((x_1 \mathbf{u} \mathbf{y})|_s) < \sum_{s \in S} p'((x_2 \mathbf{u} \mathbf{y})|_s)$, where S is the set of soft constraints of N_c and notation $(x_1 \mathbf{u} \mathbf{y})|_s$ stands for the projection on the outcome on constraint s . The constraints on which $x_1 \mathbf{u} \mathbf{y}$ differs from $x_2 \mathbf{u} \mathbf{y}$ are: constraint c on $V_{Pa(X)}$ and V_X , and all the constraints $t_i \in T$ on $V_{Pa(B_i)}$ and V_{B_i} such that $X \in Pa(B_i)$ (in what follows, we denote the children of X by $\mathcal{B} = \{V_{B_1}, \dots, V_{B_h}\}$). Thus, we can rewrite the above inequality as $p'((x_1 \mathbf{u} \mathbf{y})|_c) + \sum_{t_i \in T} p'((x_1 \mathbf{u} \mathbf{y})|_{t_i}) < p'((x_2 \mathbf{u} \mathbf{y})|_c) + \sum_{t_i \in T} p'((x_2 \mathbf{u} \mathbf{y})|_{t_i})$. By construction of N_c we have $p'(\pi_c(x_1 \mathbf{u} \mathbf{y})|_c) = w_c \times p(x_1 \mathbf{u}) < p'(\pi_c(x_2 \mathbf{u} \mathbf{y})|_c) = w_c \times p(x_2 \mathbf{u})$ and thus $x_1 \mathbf{u} \mathbf{y} >_{\min+} x_2 \mathbf{u} \mathbf{y}$ iff $w_c p(x_2 \mathbf{u}) - w_c p(x_1 \mathbf{u}) > \sum_{t_i \in T} p'((x_1 \mathbf{u} \mathbf{y})|_{t_i}) - \sum_{t_i \in T} p'((x_2 \mathbf{u} \mathbf{y})|_{t_i})$. In particular, this will hold if $w_c (\min_{x, x' \in \mathcal{D}(X)} |p(x'z) - p(xz)|) > \sum_{t_i \in T} w_{t_i} (\max_{x, x', z, b} |p(x'zb) - p(xzb)|)$ where z is the assignment to all parents of $\mathcal{B}$ other than X . Observe, that the maximum in the right term is obtained when $p(x'zb) = |\mathcal{D}(\mathcal{B})| - 1$ and $p(xzb) = 0$. On the other hand, $\min_{x, x' \in \mathcal{D}(X)} |p(x'u) - p(xu)| = 1$. In other words: $w_c > \sum_{t_i \in T} w_{t_i} (|\mathcal{D}(B_i)| - 1)$

Input : Acyclic CP-net N

1. Construct the SC-net N_c without weights.
2. Order variables of N in a reverse topological ordering.
3. **foreach** $X \in N$ **do**
4. **if** X has no successors in N **then**
5. $w(V_X) = 1$
6. **else**
7. $w(V_X) = \sum_{Y \text{ s.t. } X \in Pa(Y)} w(V_Y) \cdot |\mathcal{D}(V_Y)|$
8. **return** N_c

Figure 3: Algorithm for weight computation.

must hold. But this is ensured by the algorithm, setting (in line 7) $w_c = \sum_{t_i \in T} w_{t_i} (|\mathcal{D}(B_i)|)$. $\square$

Theorem 4 (Complexity) *Given an acyclic CP-net N with the node in-degree bounded by a constant, the construction of the corresponding SC-net based weighted SCSP N_c is polynomial in the size of N .*

Proof: If the CP-net has n nodes then the number of vertices V of the derived SC-net is at most $2n$. In fact, in the SC-net a node representing a feature appears at most once and there is at most one node representing its parents. If the number of edges of the CP-net is e , then the number of edges E in the SC-net (including hard and soft edges) is at most $e + n$, since each edge in the CP-net corresponds to at most one constraint, and each feature in the CP-net generates at most one new soft constraints. Topological sort can be performed in $O(V + E)$, that is, $O(2n + e + n) = O(e + n)$. Then, for each node, that is, $O(V)$ times, at most V children must be checked to compute the new weight value, leading to a number of checks which is $O(V^2) = O(n^2)$. Each check involves checking a number of assignments which is exponential in the number of parents of a node. Since we assume that the number of parents of a node is limited by a constant, this exponential is still a constant. Thus the total time complexity is $O(V^2)$ (or $O(n^2)$ if we consider the size of the CP-net). $\square$

Let us compare in more details the original preference relation induced by the CP-net and this induced by its min+ semi-ring based SC-net. The comparison is summarized in the following table, where $\sim$ denotes incomparability. Notice that Theorem 3 shows that ordering information is preserved by the approximation.

CP-nets $\Rightarrow$ min+		min+ $\Rightarrow$ CP-nets	
$\prec$	$<$	$<$	$\prec, \sim$
$\succ$	$>$	$>$	$\succ, \sim$
$\sim$	$<, >, =$	$=$	$=, \sim$

Since the min+ approximation is a total ordering, it is a linearization of the original partial ordering. In compensation, however, preferential comparison is now linear time.

4.2 SLO soft constraints

We also consider a different semi-ring to approximate CP-nets via soft constraints. The SLO c-semi-ring is defined as follows: $S_{SLO} = \langle A, \max_s, \min_s, \mathbf{MAX}, \mathbf{0} \rangle$, where A is the set of sequences of n integers from 0 to $\mathbf{MAX}$, $\mathbf{MAX}$ is the sequence of n elements all equal to $\mathbf{MAX}$, and $\mathbf{0}$ is the sequence of n elements all equal to 0. The additive operator, $\max_s$ and the multiplicative operator, $\min_s$ are defined as follows: given $s = s_1 \dots s_n$ and $t = t_1 \dots t_n$, $s_i = t_i, i = 1 \leq k$ and $s_{k+1} \neq t_{k+1}$, then $\max_s(s, t) = s$ if $s_{k+1} \succ t_{k+1}$ else $\max_s(s, t) = t$; on the contrary, $\min_s(s, t) = s$ if $s_{k+1} \prec t_{k+1}$ else $\min_s(s, t) = t$.

It is easy to show that S_{SLO} is a c-semi-ring and that the ordering induced by $\max_s$ on A is lexicographic ordering [Fargier et al., 1993]. To model a CP-net as a soft constraint problem based on S_{SLO} , we set $\mathbf{MAX}$ equal to the cardinality of the largest domain - 1, and n equal to the number of soft constraints of the SC net. All the weights of the edges are set to 1. Considering the binary soft constraint on $Pa(X) =$

$\{U_1 \dots U_h\}$ and X , a tuple of assignments $(u_1, \dots, u_h, x)$ will be assigned, as preference, the sequence of n integers: $(MAX, MAX, \dots, MAX - i + 1, \dots, MAX)$. In this sequence, each element corresponds to a soft constraint. The element corresponding to the constraint on $Pa(X)$ and X is $MAX - i + 1$, where i is the distance from the top of the total order of the value x (i.e. we have a preference statement of the form $u : x_1 \succ x_2 \succ \dots x_i = x \succ x_{|D(X)|}$). In the example shown in Figure 2, all the preferences will be lists of four integers (0 and 1), where position i corresponds to constraint with weight w_i . For example, in constraint weighted w_3 , $p_1 = (1, 1, 1, 1)$ and $p_2 = (1, 1, 0, 1)$. Given the pair of outcomes $o_1 = abcd$ and $o_2 = \bar{a}bcd$, the global preference associated with o_1 is $(1, 1, 1, 1)$, since it does not violate any constraint, while the preference associated with o_2 is $\min_S\{(1, 1, 1, 1), (1, 0, 0, 1, 1), (1, 1, 0, 1), (1, 1, 1, 1)\} = (1, 0, 1, 1)$. We can conclude that o_1 is better than o_2 .

Similar to the comparison performed for min+ semi-ring, the following table compares the preference relation induced by the SLO semiring and that induced by the CP-net.

CP-nets $\Rightarrow$ SLO		SLO $\Rightarrow$ CP-nets	
$\prec$	$<$	$<$	$\prec, \sim$
$\succ$	$>$	$>$	$\succ, \sim$
$=$	$=$	$=$	$=$
$\sim$	$<, >$		

Note that the SLO model both preserves information and ensures the cp-condition. The proof of this is straightforward and is omitted due to lack of space. The SLO model, like the weighted model, is very useful to answer dominance queries as it inherits the linear complexity from its semi-ring structure. In addition, the sequences of integers show directly the “goodness” of an assignment, i.e. where it actually satisfies the preference and where it violates it.

4.3 Comparing the two approximations

Given an acyclic CP-net N , let N_c^{min+} and N_c^{SLO} stand for the corresponding min+ and SLO based SC-nets respectively. From the results in the previous section, we can see that pairs of outcomes ordered by N remain ordered the same way by both N_c^{min+} and N_c^{SLO} . On the other hand, pairs of outcomes incomparable in N are distributed among the three possibilities (equal or ordered in one the two ways) in N_c^{min+} , while being strictly ordered by N_c^{SLO} . Therefore, the (total) preference relation induced by N_c^{min+} is a less brutal linearization of the partial preference relation induced by N , compared to that induced by N_c^{SLO} . Mapping incomparability onto equality might seem more reasonable than mapping it onto an arbitrary strict ordering, since the choice is still left to the user. We might conclude that the min+ model is to be preferred to the SLO model, as far as approximation is concerned. However, maximizing the minimum reward, as in any fuzzy framework [Schiex, 1992], has proved its usefulness in problem representation. The user may therefore need to balance the linearization of the order and the suitability of the representation provided.

5 Future Work

We plan to use our approach in a preference elicitation system in which we guarantee the consistency of the user prefer-

ences, and guide the user to a consistent scenario. Moreover, we also plan to exploit the use of partially ordered preferences, as allowed in soft constraints, to better approximate CP nets. Finally, we intend to use machine learning techniques to learn conditional preferences from comparisons of complete assignments.

References

- [Bistarelli et al., 1996] S. Bistarelli, H. Fargier, U. Montanari, F. Rossi, T. Schiex, and G. Verfaillie. Semiring-based CSPs and valued CSPs: Basic properties and comparison. In *Over-Constrained Systems*. 1996.
- [Bistarelli et al., 1997] S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based Constraint Solving and Optimization. *Journal of the ACM*, 44(2):201–236, 1997.
- [Boutilier et al., 1999] C. Boutilier, R. Brafman, H. Hoos, and D. Poole. Reasoning with Conditional Ceteris Paribus Preference Statements. In *Proc. of UAI-99*, 1999.
- [Boutilier et al., 2001] C. Boutilier, F. Bacchus, and R. I. Brafman. UCP-Networks: A Directed Graphical Representation of Conditional Utilities. In *Proc. of UAI-01*, 2001.
- [Boutilier et al., 2002] C. Boutilier, R. Brafman, C. Domshlak, H. Hoos, and D. Poole. CP-nets: A Tool for Representing and Reasoning about Conditional Ceteris Paribus Preference Statements. *submitted for publication*, 2002.
- [Domshlak and Brafman, 2002] C. Domshlak and R. Brafman. CP-nets - Reasoning and Consistency Testing. In *Proc. of KR-02*, pages 121–132, 2002.
- [Doyle and Thomason, 1999] J. Doyle and R. H. Thomason. Background to Qualitative Decision Theory. *AI Magazine*, 20(2):55–68, 1999.
- [Doyle and Wellman, 1994] J. Doyle and M. Wellman. Representing Preferences as Ceteris Paribus Comparatives. In *Proc. AAAI Spring Symposium on Decision-Making Planning*, pages 69–75, 1994.
- [Fargier et al., 1993] H. Fargier, J. Lang, and T. Schiex. Selecting preferred solutions in fuzzy constraint satisfaction problems. In *Proc. 1st European Congress on Fuzzy and Intelligent Technologies (EUFIT)*, 1993.
- [Hansson, 2001] S. O. Hansson. Preference Logic. In D. M. Gabbay and F. Guenther, editors, *Handbook of Philosophical Logic*, volume 4, pages 319–394. Kluwer, 2001.
- [Lang, 2002] J. Lang. From Preference Representation to Combinatorial Vote. In *Proc. of KR-02*, 2002.
- [McGeachie and Doyle, 2002] M. McGeachie and J. Doyle. Efficient utility functions for ceteris paribus preferences. In *Proc. AAAI 2002*, pages 279–284, 2002.
- [Sabin and Weigel, 1998] D. Sabin and R. Weigel. Product Configuration Frameworks - A Survey. *IEEE Intelligent Systems and their Applications*, 13(4):42–49, 1998.
- [Schiex et al., 1995] T. Schiex, H. Fargier, and G. Verfaillie. Valued Constraint Satisfaction Problems: Hard and Easy Problems. In *Proc. of IJCAI*, pages 631–637, 1995.
- [Schiex, 1992] T. Schiex. Possibilistic constraint satisfaction problems, or “How to handle soft constraints?”. In *Proc. of UAI*, pages 269–275, 1992.

Multiset Ordering Constraints*

Alan Frisch and Ian Miguel

Dept of Computer Science
University of York
England
{frisch, ianm}@cs.york.ac.uk

Zeynep Kiziltan

Dept of Information Science
Uppsala University
Sweden
Zeynep.Kiziltan@dis.uu.se

Brahim Hnich and Toby Walsh

Cork Constraint Computation Centre
University College Cork
Ireland
{brahim,tw}@4c.ucc.ie

Abstract

We identify a new and important global (or non-binary) constraint which ensures that the values taken by two vectors of variables, when viewed as multisets, are ordered. This constraint is useful for a number of different applications including breaking symmetry and fuzzy constraint satisfaction. We propose and implement a linear time algorithm for enforcing generalised arc-consistency on such a multiset ordering constraint. Experimental results show considerable promise.

1 Introduction

Global (or non-binary) constraints are one of the factors central to the success of constraint programming [Régim, 1994; 1996; Beldiceanu, 2000]. Global constraints specify patterns that occur in many problems, and call efficient and effective constraint propagation algorithms. In this paper, we identify a new and important global constraint. This constraint ensures that the values taken by two vectors of variables, when viewed as multisets, are ordered. Such a constraint is useful in a number of domains. For example, in the progressive party problem (prob013 in csplib.org), we wish to assign a host for each guest and period. We can model this with a vector of variables for each period. Each variable is assigned the host for a particular guest. This model has unnecessary symmetry as the periods can be freely permuted. We can break this symmetry by considering the multiset of values associated with each vector and ordering these multisets. The aim of this paper is to study such multiset ordering constraints and to develop efficient and effective techniques for enforcing them.

2 Formal Background

A constraint satisfaction problem (CSP) consists of a set of variables, each with a finite domain of values, and a set of constraints that specify allowed values for subsets of variables. A solution is an assignment of values to the variables satisfying the constraints. To find such solutions, we explore partial assignments enforcing a local consistency like generalised arc-consistency (GAC). A constraint is GAC iff, when

a variable in the constraint is assigned a value v , compatible values exist for all the other variables in the constraint. Such compatible values are referred as the support for v . GAC reduces to node-consistency (NC) for unary constraints. Finally, a constraint is bounds consistent (BC) iff, when a variable in the constraint is assigned its maximum or minimum value, there exist compatible values for all the other variables in the constraint. If a constraint c is NC, BC, or GAC then we write $\text{NC}(c)$, $\text{BC}(c)$, or $\text{GAC}(c)$ respectively.

A multiset is an unordered list in which repetition is allowed. A multiset M is ordered smaller than another N , written $M \prec_m N$ iff either M is empty and N is not, or the largest value in M is smaller than the largest value in N , or the largest values are the same and, if we eliminate one occurrence of the largest value from both M and N , the resulting two multisets are ordered. Note that $M \preceq_m N$ iff $M = N$ or $M \prec_m N$. We will often view the values taken by a vector of variables as a multiset. Given two vectors of variables, $\vec{x}$ and $\vec{y}$, we write a multiset ordering constraint as $\vec{x} \preceq_m \vec{y}$. This constraint is satisfied iff the values for the variables in the vectors, when treated as multisets, satisfy the multiset ordering. Similarly, we write a strict multiset ordering constraint as $\vec{x} \prec_m \vec{y}$. Throughout the paper, we assume that vectors of variables have finite integer domains, and the variables are disjoint and not repeated.

Vectors of variables are indexed from 0. The minimum element in the domain of x_i is $\min(x_i)$, and the maximum is $\max(x_i)$. The function $\text{floor}(\vec{x})$ assigns all variables in $\vec{x}$ to their minimum values, whilst $\text{ceil}(\vec{x})$ assigns all to their maximums. The vector $\vec{x}_{v=d}$ is identical to $\vec{x}$ except v now has the domain $\{d\}$. An occurrence vector $\text{occ}(\vec{x})$ associated with $\vec{x}$ is indexed in decreasing order from the maximum $\max(\vec{x})$ to the minimum $\min(\vec{x})$ value from the domains in $\vec{x}$. The i th element of $\text{occ}(\vec{x})$ is the number of occurrences of $\max(\vec{x}) - i$ in $\vec{x}$. When comparing two occurrence vectors, we assume they start and end with the occurrence of the same value, adding leading/ trailing zeroes as necessary. Finally, $\vec{x} \leq_{\text{lex}} \vec{y}$ iff $\vec{x}$ is lexicographically less than or equal to $\vec{y}$.

3 Motivating applications

3.1 Matrix symmetry

Many constraints programs contain matrices of decision variables (so called “matrix models”), and the rows and/or

*Support received by EPSRC under GR/R30792 and by the Science Foundation Ireland. We thank Chris Beck and Chris Jefferson.

columns of these matrices are symmetric and can be permuted [Flener *et al.*, 2002]. Such symmetries are very difficult to deal with as there are a super-exponential number of permutations of the rows or columns to consider. There are several ways to break symmetry in a CSP, such as SBDS [Gent and Smith, 2000] or SBDD [Fahle *et al.*, 2001]. One of the most effective, and the one which we will concentrate on as a major application for a multiset ordering constraint, is adding extra symmetry-breaking constraints to an initial model. Existing techniques for dealing with such symmetries typically eliminate only some of the symmetry. Additional techniques, like those proposed here, are therefore of considerable value.

The progressive party problem mentioned earlier has a 2d matrix of decision variables with matrix symmetry. The rows of the matrix are the guests, the columns are the periods. Each variable gives the host assigned to a given guest in a given period. As periods are indistinguishable, the columns of the matrix are symmetric. One way to break such column symmetry is to lex order the columns [Frisch *et al.*, 2002]. Similarly, as guests can be indistinguishable, (some of) the rows may be symmetric and can be lex ordered. Alternatively, we can treat each row and/or column as a multiset and break such symmetry by multiset ordering the rows and/or columns.

Unlike lex ordering, multiset ordering the rows of a matrix may not eliminate all row symmetry. For example, consider the symmetric matrices:

$$\begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \quad \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}$$

Both satisfy the constraint that the first row is multiset less than the second. It is therefore a little surprising to discover that multiset ordering (which does not break all row symmetry) is not dominated by lex ordering (which does) but is incomparable. For example, $\langle 0, 2 \rangle \leq_{lex} \langle 1, 1 \rangle$ but $\langle 1, 1 \rangle \leq_m \langle 0, 2 \rangle$.

When we have both row and column symmetry, we can multiset order both rows and columns. Like lex ordering both rows and columns, this may not eliminate all row and column symmetry. Consider the symmetric matrices:

$$\begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix} \quad \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$

Both have multiset ordered rows and columns. Unsurprisingly, multiset ordering rows and columns is incomparable to lex ordering rows and columns. Consider the symmetric matrices:

$$\begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \end{pmatrix}$$

The first has lex ordered rows and columns, but the columns are not multiset ordered. The second has rows and columns that are multiset ordered but the columns are not lex ordered.

An alternative way to deal with row and column symmetry is to multiset order in one dimension and apply the symmetry breaking method of our choice in the other dimension. This is one of the best features of using multiset ordering to break symmetry. It is compatible with any other method in the other dimension. For instance, we can multiset order the rows and lex order the columns. Preliminary results in [Kiziltan and Smith, 2002] suggest that such a combined method is very promising. This combined method does not eliminate

all symmetry (but it is unlikely that any polynomial set of constraints does). Consider the symmetric matrices:

$$\begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 2 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 2 & 1 & 0 \end{pmatrix}$$

Both have rows that are multiset ordered, and rows and columns that are lex ordered. Multiset ordering the rows and lex ordering the columns is again incomparable to lex ordering rows and columns. Consider the symmetric matrices:

$$\begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 2 & 0 \\ 0 & 2 & 2 \end{pmatrix} \quad \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 2 & 2 \\ 1 & 0 & 0 \\ 1 & 0 & 2 \end{pmatrix}$$

The first matrix has rows that are multiset ordered and columns that are lex ordered. However, its rows are not lex ordered. The second matrix has rows and columns that are lex ordered but does not have rows that are multiset ordered. Whilst the two orderings are theoretically incomparable, our experimental results (see later) show that multiset ordering the rows and lex ordering the columns is often the most effective symmetry breaking constraint currently known.

3.2 Fuzzy constraints

Another application for multiset ordering is to fuzzy CSPs. A fuzzy constraint associates a degree of satisfaction to an assignment tuple for the variables it constrains. To combine degrees of satisfaction, we can use a combination operator like the minimum function. Unfortunately, the minimum function may cause a *drowning effect* when one poorly satisfied constraint ‘drowns’ many highly satisfied constraints. One solution is to collect a vector of degrees of satisfaction, sort these values in ascending order and compare them lexicographically. This *leximin* combination operator identifies the assignment that violates the fewest constraints [Fargier, 1994]. This induces an ordering identical to the multiset ordering except that the lower elements of the satisfaction scale are the more significant. It is simple to modify a multiset ordering constraint to consider the values in a reverse order. To solve such leximin fuzzy CSPs, we can then use branch and bound, adding an ordering constraint when we find a solution to ensure that future solutions are greater in the leximin ordering.

4 GAC algorithm for multiset ordering

The last section motivated why we want multiset ordering constraints. We need, however, to be able to propagate such constraints efficiently. We therefore developed an efficient GAC algorithm for such constraints.

4.1 Background

The algorithm exploits two theoretical results. The first reduces the problem to testing support for upper bounds of $\vec{x}$ and lower bounds of $\vec{y}$ on suitable ground vectors. The second reduces these tests to lex ordering suitable occurrence vectors. Identical results hold for the strict multiset ordering constraint but for reasons of space we omit them here.

Lemma 1 *Given two disjoint and non-repeating vectors of variables, $\vec{x}$ and $\vec{y}$, with non-empty domains, $GAC(\vec{x} \leq_m \vec{y})$ iff $\forall x_i \in \vec{x}, \forall y_j \in \vec{y}$:*

$$\begin{aligned} \text{floor}(\vec{x}_{x_i = \max(x_i)}) &\leq_m \text{ceil}(\vec{y}) \\ \text{floor}(\vec{x}) &\leq_m \text{ceil}(\vec{y}_{y_j = \min(y_j)}) \end{aligned}$$

Proof: ($\Rightarrow$) As the constraint is GAC, all values have support. In particular, $x_i = \max(x_i)$ has support. The best support comes if all the other variables in $\vec{x}$ take their minimums, and all the variables in $\vec{y}$ take their maximums. Hence, $\text{floor}(\vec{x}_{x_i=\max(x_i)}) \leq_m \text{ceil}(\vec{y})$. Similarly, for y_i .

($\Leftarrow$) The first constraint ensures that $\max(x_i)$ is supported. The values which support $\max(x_i)$ also support all smaller values. Hence, all the values in the domain of x_i are supported. By an analogous argument, all the values in the domain of y_i are supported. Hence the constraint is GAC. QED.

The next lemma reduces these tests for support to lex ordering suitable occurrence vectors.

Lemma 2 *Given two multisets of values, M and N , $M \leq_m N$ iff $\text{occ}(M) \leq_{\text{lex}} \text{occ}(N)$.*

Proof: See [Kiziltan and Walsh, 2002].

4.2 A worked example

Based on these lemmas, we have designed an efficient algorithm for enforcing GAC on a multiset ordering constraint. The algorithm goes through the x_i and y_j checking for support in the appropriate occurrence vectors. Incremental computation of the lex ordering test avoids repetition of work. Consider the multiset ordering constraint $\vec{x} \leq_m \vec{y}$ where:

$$\begin{aligned}\vec{x} &= \langle \{5\}, \{4, 5\}, \{3, 4, 5\}, \{2, 4\}, \{1\}, \{1\} \rangle \\ \vec{y} &= \langle \{4, 5\}, \{4\}, \{1, 2, 3, 4\}, \{2, 3\}, \{1\}, \{0\} \rangle\end{aligned}$$

We construct occurrence vectors for $\text{floor}(\vec{x})$ and $\text{ceil}(\vec{y})$, indexed from 5 to 0:

$$\begin{aligned}\text{occ}(\text{floor}(\vec{x})) &= \langle 1, 1, 1, 1, 2, 0 \rangle \\ \text{occ}(\text{ceil}(\vec{y})) &= \langle 1, 2, 1, 0, 1, 1 \rangle\end{aligned}$$

Recall that index i in $\text{occ}(\text{ceil}(\vec{y}))$ denotes the number of occurrences of the value i in $\text{ceil}(\vec{y})$. For example, index 4 is 2 as the value 4 occurs twice.

We first check if $\text{occ}(\text{floor}(\vec{x})) >_{\text{lex}} \text{occ}(\text{ceil}(\vec{y}))$. If so, we can fail immediately because no value for any variable can have support. Here, $\text{occ}(\text{floor}(\vec{x})) \leq_{\text{lex}} \text{occ}(\text{ceil}(\vec{y}))$. In fact, we record (in a pointer, α) that the two occurrence vectors are lex ordered by index 4 of $\text{occ}(\text{floor}(\vec{x}))$, which is strictly smaller than index 4 of $\text{occ}(\text{ceil}(\vec{y}))$. This means that we will fail to find support in the y_j if any of the x_i is assigned a new value greater than 4. We now go through the x_i checking for support for their maximum values, and then the y_i checking for support for their minimum values.

Consider x_0 . As it has a singleton domain, and $\text{occ}(\text{floor}(\vec{x})) \leq_{\text{lex}} \text{occ}(\text{ceil}(\vec{y}))$, its only value must have support so we skip it. Now consider x_1 . Do its values have support? Changing $\text{occ}(\text{floor}(\vec{x}))$ to $\text{occ}(\langle \min(x_0), \max(x_1), \dots, \min(x_{n-1}) \rangle)$ decreases the number of occurrences of $\min(x_1) = 4$ by 1, and increases the number of occurrences of $\max(x_1) = 5$ by 1. As $\min(x_1) \geq \alpha = 4$, this upsets the lex ordering of the two occurrence vectors. We therefore prune all values in the domain of x_1 larger than α . This leaves a single supported value, 4.

Now consider x_2 . Changing $\text{occ}(\text{floor}(\vec{x}))$ to $\text{occ}(\langle \min(x_0), \dots, \max(x_2), \dots, \min(x_{n-1}) \rangle)$ decreases the number of occurrences of $\min(x_2) = 3$ by 1, and increases the number of occurrences of $\max(x_2) = 5$ by 1. As with x_1 , any value of x_2 larger than $\alpha = 4$ upsets the lex ordering. We therefore prune 5 from the

domain of x_2 . Now consider x_3 . Changing $\text{occ}(\text{floor}(\vec{x}))$ to $\text{occ}(\langle \min(x_0), \dots, \max(x_3), \dots, \min(x_{n-1}) \rangle)$ decreases the number of occurrences of $\min(x_3) = 2$ by 1, and increases the number of occurrences of $\max(x_3) = 4$ by 1. The occurrence vectors beneath α would now be lex ordered the wrong way. We therefore also prune the value $\alpha = 4$, leaving a single supported value 2 in the domain of x_3 . As x_4 and x_5 have singleton domains, their values have support.

Similarly, we check the minimums of the y_j for support. However, rather than prune values above (and in some cases equal to) α , there is now a dual pointer β and we prune values in the domains of y_j up to (and in some cases equal to) β . The pointer β is the largest index such that the occurrence vectors beneath it are lex ordered the wrong way. Any value less than β cannot hope to change the lex ordering as the value at β will still order the vectors the wrong way. Such values can therefore be pruned. Once we have considered each of the y_j , we have the following generalized arc-consistent vectors:

$$\begin{aligned}\vec{x} &= \langle \{5\}, \{4\}, \{3, 4\}, \{2\}, \{1\}, \{1\} \rangle \\ \vec{y} &= \langle \{5\}, \{4\}, \{3, 4\}, \{2, 3\}, \{1\}, \{0\} \rangle\end{aligned}$$

4.3 Algorithm details

The algorithm uses two pointers α and β , and two flags γ and δ to avoid traversing the occurrence vectors each time we look for support. The pointer α is set to the most significant index above which all occurrences are pair-wise equal and at α they are strictly ordered. If the vectors are equal then α is set to $-\infty$. The pointer β is set to the most significant index below α such that the occurrence vectors are lex ordered the wrong way. If no such index exists, we set β to $-\infty$. The flag γ is set to true if all the indices between α and β are pair-wise equal and the flag δ is set to true if the sub-vectors below β are lex ordered the wrong way. For example, given the occurrence vectors in section 4.2, α is set to 4, β to 2, and the flags γ and δ are set to true.

We summarise the major steps the algorithm performs:

- A. Build $\text{occ}(\text{floor}(\vec{x}))$ and $\text{occ}(\text{ceil}(\vec{y}))$
- B. Set $\alpha, \beta, \gamma, \delta$ according to their definitions
- C. For each x_i : If its maximum disturbs the lex ordering on the occurrence vectors, tighten its upper-bound to α when the occurrence vectors are lex ordered below α , otherwise to $\alpha - 1$.
- D. For each y_j : If its minimum disturbs the lex ordering on the occurrence vectors, then tighten its lower-bound to β when the occurrence vectors are lex ordered below α , otherwise to $\beta + 1$.

When we prune a value, we do not need to check recursively for previous support. Pruning changes neither the lower bounds of $\vec{x}$ nor the upper bounds of $\vec{y}$. These values continue to provide support. The exception is when a domain is a singleton, and pruning causes a domain wipe-out.

We now give pseudo-code for an algorithm that maintains GAC on a multiset ordering constraint between vectors $\vec{x}$ and $\vec{y}$ which are of length n and m respectively. As the algorithm reasons about occurrences vectors, the original vectors need not be identical in length (though they often are).

The algorithm is called whenever lower bounds of x_i or upper bounds of y_j change. Lines A1 to A3 build the occurrence vectors $\vec{o}\vec{x}$ and $\vec{o}\vec{y}$. Line B1 calls the procedure to

set the pointers α and β , and the flags γ and δ . Lines **C1-13** check support for the maximums of the x_i 's while lines **D1-14** check support for the minimums of the y_i 's.

Procedure GACMSO()

```

A1.  $l := \min(\{\min(x_i) | i \in [0, n)\} \cup \{\min(y_j) | j \in [0, m)\})$ 
A2.  $u := \max(\{\max(x_i) | i \in [0, n)\} \cup \{\max(y_j) | j \in [0, m)\})$ 
A3.  $o\vec{x} = occ(floor(\vec{x})), o\vec{y} = occ(ceil(\vec{y}))$ 
B1. SetPointers-and-Flags( $l, u$ )
C1. FOR  $i = 0$  TO  $n - 1$  DO % check support for  $x$ 's
C2. IF ( $\min(x_i) \neq \max(x_i)$ ) THEN
C3.  $a := \min(x_i); b := \max(x_i);$ 
C4. IF ( $a \geq \alpha$ ) THEN  $NC(x_i = a);$ 
C5. IF ( $\alpha > a > \beta \wedge b \geq \alpha$ ) THEN  $NC(x_i \leq \alpha);$ 
C6. IF ( $a = \beta \wedge b \geq \alpha$ ) THEN
C7. IF ( $ox_\alpha + 1 = oy_\alpha$ ) THEN
C8. IF ( $\gamma \wedge ox_\beta - 1 = oy_\beta \wedge \delta$ ) THEN  $NC(x_i < \alpha);$ 
C9. ELSE  $NC(x_i \leq \alpha);$ 
C10. IF ( $a < \beta \wedge b \geq \alpha$ ) THEN
C11. IF ( $ox_\alpha + 1 = oy_\alpha$ ) THEN
C12. IF ( $\gamma$ ) THEN  $NC(x_i < \alpha);$ 
C13. ELSE  $NC(x_i \leq \alpha);$ 
D1. FOR  $j = 0$  TO  $m - 1$  DO % check support for  $y$ 's
D2. IF ( $\min(y_j) \neq \max(y_j)$ ) THEN
D3.  $a := \min(y_j); b := \max(y_j)$ 
D4. IF ( $b > \alpha$ ) THEN  $NC(y_j = b);$ 
D5. IF ( $b = \alpha \wedge ox_\alpha + 1 = oy_\alpha$ ) THEN
D6. IF ( $a \leq \beta \wedge \gamma$ ) THEN
D7. IF ( $ox_\beta = oy_\beta + 1$ ) THEN
D8. IF ( $\delta$ ) THEN  $NC(y_j > \beta);$ 
D9. ELSE  $NC(y_j \geq \beta);$ 
D10. ELSE  $NC(y_j > \beta);$ 

```

Procedure SetPointers-and-Flags(l, u)

```

B1.  $\gamma := false; \delta := false; \alpha := u;$ 
B2. WHILE ( $\alpha \geq l \wedge ox_\alpha = oy_\alpha$ ) DO  $\alpha := \alpha - 1;$ 
B3. IF ( $\alpha \geq l \wedge ox_\alpha > oy_\alpha$ ) THEN FAIL;
B4. IF ( $\alpha = l - 1$ ) THEN  $\alpha := -\infty; \beta := -\infty;$ 
B5. ELSE
B6.  $\beta := \alpha - 1; \gamma := true;$ 
B7. WHILE ( $\beta \geq l \wedge ox_\beta \leq oy_\beta$ ) DO
B8. IF ( $ox_\beta < oy_\beta$ ) THEN  $\gamma := false;$ 
B9.  $\beta := \beta - 1;$ 
B10. IF ( $\beta = l - 1$ ) THEN  $\beta := -\infty; \gamma := false;$ 
B11. IF ( $\beta \neq -\infty \wedge \beta > l$ ) THEN
B12.  $i := \beta - 1;$ 
B13. WHILE ( $i \geq l \wedge ox_i = oy_i$ ) DO  $i := i - 1;$ 
B14. IF ( $i \geq l \wedge ox_i > oy_i$ ) THEN  $\delta := true;$ 

```

For each x_i , we only check for support if x_i is not singleton (line **C2**). There are six cases where we prune the domain of x_i : (1) at line **C4**, as $\min(x_i) \geq \alpha$, any value in the domain of x_i greater than $\min(x_i)$ lacks support because it will disturb the lex ordering; (2) at line **C5**, as $\alpha > \min(x_i) > \beta$, and $\max(x_i) \geq \alpha$, all the values in the domain of x_i greater than α disturb the lex ordering, and lack support; (3) at line **C6**, **C7**, and **C8**, all values greater than or equal to α lack support. If we assign α to x_i , then the vectors will be equal at index α and β , the values between α and β are also all pair-wise equal (since γ is true), and the vectors below β are ordered the wrong way (since δ is true). Thus, the value α also lacks support and is pruned; (4) at line **C9**, all values greater than α lack support; (5) at line **C10**, **C11**, and **C12**, all values greater than or equal to α lack support. If we assign α to x_i , the vectors will be lex ordered the wrong way as the

difference between the number of occurrences at α is exactly one, and γ is true. Thus, the value α also lacks support and is pruned; (6) at line **C13** all values greater than α lack support.

For each y_j , we only check for support if y_j is not singleton (line **D2**). There are four cases where we prune the domain of y_j : (1) at line **D4**, as $\max(y_j) > \alpha$, any value smaller than $\max(y_j)$ disturbs the lex ordering and lacks support; (2) at lines **D5** to **D8**, the situation is dual to the third case for the x_i , and any value less than or equal to β lacks support; (3) at line **D9**, any value less than β lacks support; (4) at line **D10**, again, any value less than or equal to β lacks support.

4.4 Theoretical properties

In a longer technical report, we prove the following results about the algorithm's complexity and correctness.

Theorem 1 *GACMSO runs in time $O(n + m + d)$ where d is $u - l$.*

If $d \ll n, m$ (and for multisets, we expect this as the number of values is typically less than the cardinalities to permit repetition), then the algorithm is $O(n + m)$.

Theorem 2 *For disjoint and non-repeating vectors, GACMSO either establishes failure if $\vec{x} \leq_m \vec{y}$ is not satisfiable, or prunes values from $\vec{x}$ and $\vec{y}$ to ensure $GAC(\vec{x} \leq_m \vec{y})$.*

The algorithm can easily be modified to support strict multiset ordering. The only differences are that we fail if $o\vec{x} = o\vec{y}$ and that β is set to $l - 1$ not $-\infty$. The algorithm then enforces a strict inequality on the occurrence sub-vectors above β . Another variant of the algorithm is when $d \gg n, m$. In such a situation, it could be costly to construct the occurrence vectors. We can instead sort the minimums of the x_i and the maximums of the y_j , and compute α, β, γ and δ as if we had the occurrences by scanning these sorted lists. This information is all we need to compute support for each x_i and y_j in turn. The complexity of this modified algorithm is $O(n \log n + m \log m)$ as the cost of sorting dominates.

If we have multiple rows of a matrix that are multiset ordered, we can decompose this into multiset ordering constraint on all pairs of rows, or (further still) onto ordering constraints just on neighbouring pairs of rows. The following result shows that such decompositions hinder constraint propagation. Nevertheless, it will usually be most cost effective to post just the $O(n)$ ordering constraints between neighbouring pairs rather than the $O(n^2)$ constraints between all pairs.

Theorem 3 *$GAC(\forall i < j. \vec{x}_i \leq_m \vec{x}_j)$ is strictly stronger than $GAC(\vec{x}_i \leq_m \vec{x}_j)$ for all $i < j$, and this itself is strictly stronger than $GAC(\vec{x}_i \leq_m \vec{x}_{i+1})$ for all i .*

5 Alternative approaches

5.1 Arithmetic constraint

Barbara Smith (personal communication) has proposed enforcing $\vec{x} <_m \vec{y}$ on vectors of length n via the arithmetic constraint $n^{x_0} + \dots n^{x_{n-1}} < n^{y_0} + \dots n^{y_{n-1}}$. This is similar to the transformation of a leximin fuzzy CSP into an equivalent MAX CSP [Schiex *et al.*, 1995]. BC on such a constraint

is equivalent to GAC on the original multiset ordering constraint. However, such an arithmetic constraint is only feasible for small n . Further, most existing solvers will not enforce BC on such an arithmetic constraint, but will delay it until all but one of the variables are instantiated.

5.2 Decomposition

Multiset ordering is equivalent to the lex ordering the associated occurrence vectors. As we have efficient algorithms for constructing occurrence vectors (via the global cardinality constraint [Régim, 1996]) and for lex ordering [Frisch *et al.*, 2002], this might be an alternative approach. However, as the following theorem shows, such a decomposition hinders constraint propagation. Also, the two global cardinality constraints in such a decomposition are more expensive to enforce than the algorithm presented here. We write $gcc(\vec{x}, \vec{o}\vec{x})$ for the global cardinality constraint that channels from a vector of variables $\vec{x}$ to the associated occurrence vector $\vec{o}\vec{x}$.

Theorem 4 $GAC(\vec{x} <_m \vec{y})$ is strictly stronger than simultaneously enforcing $GAC(gcc(\vec{x}, \vec{o}\vec{x}))$, $GAC(gcc(\vec{y}, \vec{o}\vec{y}))$, and $GAC(\vec{o}\vec{x} <_{lex} \vec{o}\vec{y})$.

Proof: Clearly it is as strong. To show strictness, consider $\vec{x} = \{\{1, 2\}, \{1, 2\}, \{2\}, \{2\}\}$ and $\vec{y} = \{\{1, 2\}, \{1, 2\}, \{0, 1, 2\}, \{0, 1\}\}$. The multiset ordering constraint is not GAC since 0 in y_2 has no support but the decomposition is unable to determine this. QED.

Another approach is to use the sorted constraint in the Eclipse solver. This ensures that the values taken by one vector of variables are identical but in sorted order to the values taken by a second vector of variables. To post a multiset ordering constraint on two vectors, we can channel each into a sorted vector and lex order these. The above example demonstrates that such a decomposition again hinders propagation. The sorting constraint is also more expensive to enforce.

6 Experimental results

We designed some experiments to test three goals. First, is multiset ordering an effective method for dealing with row and/or column symmetry? Second, how does multiset ordering compare to lex ordering? Which one breaks more symmetry? Is a combined method, which multiset orders one dimension and lex orders the other one of the matrix, superior? Third, does our GAC algorithm do more inference in practice than the decomposition? Similarly, is the algorithm more efficient in practice than its decomposition?

The symmetry breaking constraints we used are strict lex ordering on the columns ($<_{lex}C$), on the rows ($<_{lex}R$); multiset ordering on the rows ($\leq_m R$), (strict) multiset ordering on the columns ($\leq_m C$ and $<_m C$); and combinations of these constraints. Such constraints are posted between adjacent rows/columns. The results of the experiments are shown in tables where a “-” means no result is obtained in 1 hour (3600 secs). The experiments are done using ILOG Solver 5.2 on a 1000MHz pentium III with 256 Mb RAM using Windows XP.

6.1 Progressive Party Problem

There are a set of host boats, each with a capacity, and a set of guest boats, each with a crew size. We wish to assign a host

	Fails	Choice points	Time (secs.)
No Symmetry Breaking	180,738	180,860	75.9
$<_{lex}C$	180,738	180,860	81.5
$<_{lex}R$	2,720	2,842	2.7
$<_{lex}RC$	2,720	2,842	2.7
$\leq_m C$	137,185	137,306	71.2
$\leq_m R$	10,853	10,977	8.6
$\leq_m RC$	-	-	-
$<_{lex}C \leq_m R$	10,853	10,977	8.6
$\leq_m C <_{lex}R$	2,016	2,137	2.6

Table 1: 5-13-29 progressive party problem using row-by-row labelling.

for each guest and period, such that a guest crew never visits the same host twice, no two guest crews meet more than once, and the spare capacity of each host boat, after accommodating its own crew, is not exceeded (prob013 in csplib.org).

A matrix model of this problem [Smith *et al.*, 1995] is a 2-d matrix of *guests* $\times$ *periods* where each variable is assigned a host representing that a host is accommodating a particular guest in a given time period. The rows are the guests, the columns are the periods. This model has column and partial row symmetry: any two periods, and any two guests with the same crew size are indistinguishable. We consider the 13-hosts and 29 guests problem with 5 and 6 time periods, referred as 5-13-29 and 6-13-29. These problems have $p!14!2!4!5!7!$ row and column symmetries where p is the number of time periods. The actual data can be found in csplib.org. Due to the problem constraints, no pair of rows/columns can be equal, hence we can safely pose strict lex ordering. However, any two distinct rows/columns might be equal when viewed as multisets.

As in [Smith *et al.*, 1995], the guest boats are ordered in descending order of their size. We order the host boats in descending order of spare capacity to choose a value in a succeed-first manner. Results obtained by row-by-row, and column-by-column labelling strategies are given in Tables 1 and 2. With row-by-row labelling, we cannot solve 6-13-29 with or without symmetry breaking. For the 5-13-29 problem, $<_{lex}R$ breaks a lot more row symmetry than $\leq_m R$. However, the reverse is true for the columns. Here, $<_{lex}C$ does not break any symmetry but $\leq_m C$ does. Multiset ordering one dimension of a matrix therefore does not necessarily break less symmetry than lex ordering the same dimension. Such phenomena occur through interactions with the search strategy: a search strategy might already lex order, so multiset ordering constraints break additional symmetry. The smallest search tree and also the least solving time is obtained by $\leq_m C$ $<_{lex}R$. This supports our conjecture that lex ordering one dimension combined with multiset ordering the other can break more symmetry than lex/multiset ordering both dimensions.

With column-by-column labelling, we are able to solve the 6-13-29 problem. Neither of $<_{lex}R$, $<_{lex}C$, $<_{lex}RC$ break any symmetry. The smallest search tree is obtained by $\leq_m C$. This supports our conjecture that multiset ordering one dimension can break more symmetry than lex ordering the same or both dimensions. If the search strategy already orders both dimensions lexicographically, imposing a constraint like multiset ordering in one dimension breaks additional symmetry.

	Fails	Choice points	Time (secs.)
No Symmetry Breaking	20,722	20,871	12.3
$<_{lex}C$	20,722	20,871	12.4
$<_{lex}R$	20,722	20,871	12.5
$<_{lex}RC$	20,722	20,871	12.4
$\leq_m C$	7,053	7,202	4.6
$\leq_m R$	-	-	-
$\leq_m RC$	-	-	-
$<_{lex}C \leq_m R$	-	-	-
$\leq_m C <_{lex}R$	7,053	7,202	4.6

Table 2: 6-13-29 progressive party problem using column-by-column labelling.

6.2 Sports Scheduling with Odd Teams

This is a modified version of prob026 in csplib.org. We have n teams (n is odd), playing over n weeks. Each week is divided into $(n - 1)/2$ periods, and each period is divided into 2 slots, home and away. We wish to find a schedule so that every team plays at most once a week, every team plays twice in the same period over the tournament and every team plays every other team. We slightly modify the model in [Van Hentenryck *et al.*, 1999], where *teams* is a 3-d matrix of *periods* $\times$ *weeks* $\times$ *slots*. Each element of *teams* is the team playing in a given period, week and slot. We treat this matrix as 2-d where the rows are the periods and columns are the weeks, and each entry is a list of variables giving the slots.

As the periods and the weeks are indistinguishable, this problem has $n!(n - 1/2)!$ row and column symmetries. We pose strict ordering constraints on the rows and columns of *teams* as the periods and weeks cannot be equal. Due to the constraints on the periods, posing multiset ordering on the rows is not effective.

Results obtained by column-by-column labelling of the *teams* are given in Table 3. For one column, we first label the first slots; for the other, we first label the second slots. With this strategy, $<_{lex}R$ does not break any symmetry, so we omit it in the table. Posing multiset ordering by our algorithm is much more effective and efficient than by *gcc* and lex ordering constraints. This holds for many other search strategies. In Table 3, we note that $\leq_m C$ gives a smaller search tree than $<_{lex}C$. However, for other search strategies the reverse is true. This supports the theoretical result that lex ordering and multiset ordering are incomparable.

7 Conclusions

We have identified a new and important global (non-binary) constraint. This constraint ensures that the values taken by two vectors of variables, when viewed as multisets, are ordered. We have developed an efficient linear time algorithm for enforcing generalised arc-consistency on such a multiset ordering constraint. We have proposed a number of applications for this new constraint including breaking symmetry in matrix models, and fuzzy constraint satisfaction. We have shown that alternative methods for posting a multiset ordering constraint like an arithmetic constraint or decomposition are inferior. Finally, we have implemented this generalized arc-consistency algorithm in ILOG Solver. Experimental results on a number of problem domains show considerable promise.

n	Model	Failures	Choice points	Time (sec.)
5	No symmetry breaking	3	12	1.1
	$gcc+ <_{lex}C$	2	11	1.4
	$<_m C$	1	10	1.4
	$<_{lex}C$	3	12	1.5
7	No symmetry breaking	6,871	6,890	1.9
	$gcc+ <_{lex}C$	74	92	1.3
	$<_m C$	69	87	1.1
	$<_{lex}C$	771	788	1.3
9	No symmetry breaking	-	-	-
	$gcc+ <_{lex}C$	2,616,149	2,616,177	857.2
	$<_m C$	760,973	761,003	130.5
	$<_{lex}C$	-	-	-

Table 3: Sports scheduling problem.

References

- [Beldiceanu, 2000] N. Beldiceanu. Global constraints as graph properties on a structured network of elementary constraints of the same type. In *Proc. of CP-2000*, LNCS 1894, pp 52–66, 2000.
- [Fahle *et al.*, 2001] T. Fahle, S. Schamberger and M. Sellman. Symmetry Breaking. In *Proc. of CP-2001*, LNCS 2239, pp 93–107, 2001.
- [Fargier, 1994] H. Fargier. *Problèmes de satisfaction de contraintes flexibles: application à l’ordonnancement de production*. PhD thesis, Uni. of Paul Sabatier, Toulouse, 1994.
- [Flener *et al.*, 2002] P. Flener, A. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, J. Pearson, and T. Walsh. Breaking row and column symmetry in matrix models. In *Proc. of CP-2002*, LNCS 2470, pp 462–476, 2002.
- [Frisch *et al.*, 2002] A. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, and T. Walsh. Global constraints for lexicographic orderings. In *Proc. of CP-2002*, LNCS 2470, pp 93–108, 2002.
- [Gent and Smith, 2000] I. Gent and B. Smith. Symmetry Breaking in Constraint Programming. In *Proc. of ECAI ’2000*, pp 599–603, 2000.
- [Kiziltan and Smith, 2002] Z. Kiziltan and B. Smith. Symmetry breaking constraints for matrix models. In *Proc. of 2nd Int. Workshop on Symmetry in Constraint Satisfaction Problems (SymCon-02)*, held alongside CP-2002, 2002.
- [Kiziltan and Walsh, 2002] Z. Kiziltan and T. Walsh. Constraint Programming with Multisets. In *Proc. of 2nd Int. Workshop on Symmetry in Constraint Satisfaction Problems (SymCon-02)*, held alongside CP-2002, 2002.
- [Régis, 1994] J.C. Régis. A filtering algorithm for constraints of difference in CSPs. In *Proc. AAAI’94*, pp 362–367. 1994.
- [Régis, 1996] J.C. Régis. Generalized arc consistency for global cardinality constraints. In *Proc. of AAAI’96*, pp 209–215. 1996.
- [Schiex *et al.*, 1995] T. Schiex, H. Fargier, and G. Verfaillie. Valued constraint satisfaction problems: hard and easy problems. In *Proc. of IJCAI’95*, pp 631–637. 1995.
- [Smith *et al.*, 1995] B.M. Smith, S.C. Brailsford, P.M. Hubbard, and H.P. Williams. The progressive party problem. Research Report 95.8, Uni. of Leeds. 1995.
- [Van Hentenryck *et al.*, 1999] P. van Hentenryck, L. Michel, L. Perron, and J.C. Régis. Constraint programming in OPL. In *Proc. of PPDP’99*, pp 98–116. Springer, 1999.

Non-Binary Constraints and Optimal Dual-Graph Representations

Gianluigi Greco and Francesco Scarcello

DEIS - University of Calabria

I-87036, Rende, Italy

{ggreco,scarcello}@si.deis.unical.it

Abstract

We study the relationships among structural methods for identifying and solving tractable classes of Constraint Satisfaction Problems (CSPs). In particular, we first answer a long-standing question about the notion of biconnected components applied to an “optimal” reduct of the dual constraint-graph, by showing that this notion is in fact equivalent to the hinge decomposition method. Then, we give a precise characterization of the relationship between the treewidth notion applied to the hidden-variable encoding of a CSP and the same notion applied to some optimal reduct of the dual constraint-graph. Finally, we face the open problem of computing such an optimal reduct. We provide an algorithm that outputs an approximation of an optimal tree decomposition, and give a qualitative explanation of the difference between this graph-based method and more general hypergraph-based methods.

1 Introduction and summary of results

Constraint satisfaction is a central issue of AI research and has an impressive spectrum of applications (see, e.g., [Pearson and Jeavons, 1997]). A constraint (S_i, R_i) consists of a *constraint scope* S_i , i.e., a list of variables and of an associated *constraint relation* r_i containing the legal combinations of values. A CSP consists of a set $\{(S_1, r_1), (S_2, r_2), \dots, (S_q, r_q)\}$ of constraints whose variables may overlap. A solution to a CSP is an assignment of values to all variables such that all constraints are simultaneously satisfied. By *solving* a CSP we mean determining whether the problem has a solution at all (i.e., checking for *constraint satisfiability*), and, if so, compute one solution.

Constraint satisfiability in its general form is well-known to be NP-hard. Much effort has been spent to identify *tractable classes* of CSPs, and deep and useful results have been achieved. The various successful approaches to obtain tractable CSP classes can be divided into two main groups [Pearson and Jeavons, 1997]: the techniques that identify tractable classes of CSPs only on the base of the structure of the constraint scopes $\{S_1, \dots, S_q\}$ independently of the actual constraint relations $r_1, \dots, r_q$ and the techniques that identify tractable classes by exploiting particular properties of the constraint relations $r_1, \dots, r_q$. In this paper, we

will deal with this latter group of techniques, usually called *structural decomposition methods*. There are several papers proposing polynomially tractable classes of constraints based on different structural properties of constraint-scopes interactions (see, e.g., [Dechter, 1992; Dechter and Pearl, 1989; Gyssens *et al.*, 1994; Gottlob *et al.*, 2000]). Such a structure is best represented by the hypergraph $\mathcal{H}(I) = (V, H)$ associated to any CSP instance $I = (Var, U, \mathcal{C})$, 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 the constraint C . We often denote the set of vertices by $\mathcal{N}(\mathcal{H})$ and the set of hyperedges by $\mathcal{E}(\mathcal{H})$.

However, many interesting techniques for solving CSPs or for identifying tractable classes of CSPs have been designed for binary CSPs, i.e., CSP instances where each scope contains two variables at most. Therefore, historically, the first attempts to deal with general (i.e., non-binary) constraint problems try to exploit the existent methods, by representing any CSP instance I by some graph, rather than by the hypergraph $\mathcal{H}(I)$. A first idea is to use the primal graph of $\mathcal{H}(I)$, whose edges connect each pair of variables occurring together in some constraint of I . Clearly, there is an evident loss of information in using the primal graph instead of the hypergraph. For instance, each constraint-scope of I induces a clique in the primal graph, but if one looks at the graph only, there is no way to understand whether such a clique comes from a hyperedge of the hypergraph, or by some intricate interactions among scopes. In fact, in [Gottlob *et al.*, 2000], a deep comparison among various structural decomposition methods showed that some technique designed for hypergraphs is more powerful than all the (known) techniques working on the primal graphs. In this paper, we focus on the other two important graph-based representations of non-binary constraints, described in the literature, only marginally considered in that work:

Dual-graph Representation [Dechter, 1992]. Given a hypergraph $\mathcal{H}$, its *dual graph*, denoted by $dual(\mathcal{H}) = (N, E)$, is the graph whose set of vertices N is the set of hyperedges $\mathcal{E}(\mathcal{H})$, and whose edges connect each pair of vertices (i.e., hyperedges) having some variable in common, that is $E = \{\{h_1, h_2\} \mid h_1, h_2 \in \mathcal{E}(\mathcal{H}) \text{ and } h_1 \cap h_2 \neq \emptyset\}$

Hidden-variable Representation [Seidel, 1981; Chekuri and Rajaraman, 2000]. Given a hypergraph $\mathcal{H}$, we define its *incidence graph* as the bipartite graph $inc(\mathcal{H}) = (N, E)$,

where $N = \mathcal{E}(\mathcal{H}) \cup \mathcal{N}(\mathcal{H})$, and $E = \{ \{h, a\} \mid h \in \mathcal{E}(\mathcal{H}) \text{ and } a \in h \}$, i.e. it contains an edge from h to a if and only if a is a node of h . (E.g., Figure 2 shows a constraint hypergraph (a) and its incidence graph (b)).

There is a great interest in comparing CSP solving techniques based on these encodings of the structure [Bacchus *et al.*, 2002] and long standing questions about their relationships with hypergraph-based techniques, described below.

One of the major difficulties in doing precise and formal analysis of dual-graph based methods is due to an important feature of this encoding: some edges of the dual graph can be safely removed from this graph, making the evaluation of CSPs easier. Indeed, even if $dual(\mathcal{H})$ appears very intricate, sometimes is possible to find suitable simplifications that make it much more useful. Such simplified graphs are called *reducts* of $dual(\mathcal{H})$. For instance, if $\mathcal{H}$ is acyclic, there is a polynomial time algorithm for making its dual graph acyclic, and in fact a join tree of $\mathcal{H}$. An example is shown in Figure 2.c, where the acyclic graph obtained by removing the dashed edges is a join tree of the given hypergraph. However, different removal choices may lead to different performances of evaluation algorithms. Thus, the efficiency of any technique based on the dual graph depends crucially on the availability of a good algorithm for simplifying the dual graph. Note that finding the “best reduct” is a difficult task and is currently not known whether it is feasible in polynomial time, in general.

On the other hand, the fact that effective dual graph representations are not unique made comparisons among different methods quite difficult. For instance, Gyssens *et al.* [1994] compared the notion of Hinge decompositions (short: HINGE) and the notion of Biconnected components of the dual graph (short: BICOMP^d). It turned out that HINGE is a generalization of BICOMP^d, and thus the hinge decomposition technique is not worse than the biconnected components technique. However, the precise relationship between these methods remained an open question, because biconnected components can perform very bad unless “clever simplifications” of the dual graph are chosen.

The first contribution of this paper is solving this question. First we formally define, for each method D , the method D^{optd} , that is, the method D applied to the best possible simplification of the dual graph with respect to D . Indeed, in general, the notion of best simplification depends on the method D used for decomposing the graph. This way, methods applied to dual graph encodings are well-defined and can be compared with other methods.

We formally prove that BICOMP^{optd} is equivalent to HINGE. In fact, we show that any hinge decomposition corresponds to the biconnected-components tree of some reduct of the dual graph. It is worthwhile noting that, as a corollary of this result, we obtain that, for the BICOMP^{optd} method, an optimal reduction of the dual graph can be computed in polynomial time, since any hinge decomposition can be computed in polynomial time.

Then, we consider the powerful decomposition method for dealing with graphs: the tree decomposition method [Robertson and Seymour, 1986], which is equivalent to the tree-clustering method [Dechter and Pearl, 1989]. It is known that

any class of CSP instances such that the treewidth of their incidence graph (respectively, of some reduct of their dual graphs) is bounded by some constant k is tractable. That is, all such instances may be evaluated in time $O(n^c)$, where n is the size of a CSP instance and c is a constant that depends crucially on the bound k on the decomposition width.

We perform a detailed comparison of the tree decomposition method applied to the incidence graph of the hypergraph (i.e., on the hidden-variable encoding), denoted by TREEWIDTHⁱⁿ, and the tree decomposition method applied to some optimal reduct of the dual graph, denoted by TREEWIDTH^{optd}. It turns out that every CSP class that is tractable according to TREEWIDTHⁱⁿ is tractable according to TREEWIDTH^{optd}, as well. Moreover, there are CSP classes that are tractable according to TREEWIDTH^{optd} but are not tractable according to TREEWIDTHⁱⁿ, i.e., their largest width is not bounded by any fixed number. However, we show that TREEWIDTH^{optd} does not strongly generalize TREEWIDTHⁱⁿ. Indeed, there are classes of CSPs whose incidence-graph treewidth is bounded by a constant k , but the largest width of some optimal reduct of the dual graph is much greater. Thus, even if such classes are tractable, their evaluation can be much more efficient by using the TREEWIDTHⁱⁿ method. It follows that either of these methods may be useful for some kind of CSP instances, and hence there is no definitely better choice between them.

Finally, we focus on further interesting open questions about TREEWIDTH^{optd}. Define the *optimal treewidth* $tw^{optd}(\mathcal{H})$ of the dual graph of some hypergraph $\mathcal{H}$ as the minimum treewidth over all reducts of $dual(\mathcal{H})$. Kolaitis and Vardi [2000] observed that is not trivial to find a “good” reduct of the dual graph and defined the following problem k -OPT, for any fixed constant $k > 0$: Given a hypergraph $\mathcal{H}$, decide whether the optimal treewidth of $dual(\mathcal{H})$ is at most k . The question is whether k -OPT is decidable in polynomial time or not, that is whether there is an efficient way for computing a reduct of the dual graph that have the minimum treewidth. Moreover, even if it is known that TREEWIDTH^{optd} is strongly generalized by the hypertree decomposition method [Gottlob *et al.*, 2000], it is not clear why there is such a big difference between these methods. Indeed, at a first glance the kind of tree labelling in these methods seems rather similar.

We face both the above questions. Let $k > 0$ be a fixed constant and $\mathcal{H}$ be a constraint hypergraph. We present a polynomial time algorithm k -TREE-APPROX that, if the optimal treewidth of $dual(\mathcal{H})$ is at most k , outputs a tree decomposition of width at most $2k$ of some reduct of $dual(\mathcal{H})$. Thus, k -TREE-APPROX provides a 2-approximation of TREEWIDTH^{optd}. Note that the question whether k -OPT is decidable in polynomial time remains open, because k -TREE-APPROX can compute a tree decomposition of width at most $2k$ even if the optimal treewidth k' of $dual(\mathcal{H})$ is greater than k , i.e., if $k < k' \leq 2k$.

Moreover, our algorithm is also able to compute a new kind of structural decomposition that allows us to shed some light on the striking difference between TREEWIDTH^{optd} and the more general hypergraph-based notions of query decomposition and hypertree decomposition.

2 Preliminaries

It is well known that CSPs with *acyclic* constraint hypergraphs are polynomially solvable [Dechter, 1992]. The known structural properties that lead to tractable CSP classes are all (explicitly or implicitly) based on some generalization of acyclicity. In particular, each method D defines some concept of *width* which can be interpreted as a measure of cyclicity of the underlying constraint (hyper)graph such that, for each fixed width k , all CSPs of width bounded by k are solvable in polynomial time. This (possibly infinite) set of CSPs is called the *tractability class* of D w.r.t. k , and is denoted by $C(D, k)$. Any pair of decomposition methods D_1 and D_2 can be compared according to their ability to identify tractable classes of CSPs. Formally, Gottlob et al. [2000] defined the following criteria:

Generalization. D_2 *generalizes* D_1 ($D_1 \preceq D_2$) if there exists a constant δ such that, for each $k > 0$, $C(D_1, k) \subseteq C(D_2, k + \delta)$. In practical terms, this means that whenever a class $\mathcal{C}$ of constraints is tractable according to method D_1 , it is also tractable according to D_2 .

Beating. D_2 *beats* D_1 , denoted by $D_2 \triangleright D_1$, if there exists an integer $k > 0$ such that $C(D_2, k)$ is not contained in class $C(D_1, m)$ for any $m > 0$. Intuitively, this means that some classes of problems are tractable according to D_2 but not according to D_1 . For such classes, using D_2 is thus better than using D_1 .

Strong generalization. D_2 *strongly generalizes* D_1 , denoted by $D_1 \ll D_2$, if D_2 generalizes D_1 and D_2 beats D_1 . This means that D_2 is really the more powerful method, given that, whenever D_1 guarantees polynomial runtime for constraint solving, then also D_2 guarantees tractable constraint solving, but there are classes of constraints that can be solved in polynomial time by using D_2 but are not tractable according to D_1 .

Equivalence. D_1 and D_2 are *equivalent*, denoted by $D_1 \equiv D_2$, if D_1 generalizes D_2 and D_2 generalizes D_1 . Intuitively, this means that these two methods do not differ significantly from each other.

We assume the reader is familiar with the main structural decomposition methods proposed in the literature (see, e.g., [Pearson and Jeavons, 1997]). Anyway, for the sake of presentation, we recall the definition of graph *treewidth*.

A *tree decomposition* [Robertson and Seymour, 1986] of a graph $G = (V, E)$ is a pair $\langle T, \chi \rangle$, where $T = (N, F)$ is a tree, and χ is a labelling function associating to each vertex $p \in N$ a set of vertices $\chi(p) \subseteq V$, such that the following conditions are satisfied: (1) for each vertex b of G , there exists $p \in N$ such that $b \in \chi(p)$; (2) for each edge $\{b, d\} \in E$, there exists $p \in N$ such that $\{b, d\} \subseteq \chi(p)$; (3) for each vertex b of G , the set $\{p \in N \mid b \in \chi(p)\}$ induces a (connected) subtree of T . (Connectedness condition.)

The *width* of the tree decomposition $\langle T, \chi \rangle$ is $\max_{p \in N} |\chi(p) - 1|$. The *treewidth* of G , $tw(G)$, is the minimum width over all its tree decompositions.

We next formally define how the dual graph $dual(\mathcal{H})$ of a hypergraph $\mathcal{H}$ can be simplified, by removing edges in a suitable way. For any edge $\{h, h'\}$ of a dual graph, let $\ell(\{h, h'\})$ denote the set of variables $h \cap h'$ that the hyperedges h and h' have in common.

Definition 2.1 Let $G = (V, E)$ be the dual graph of some hypergraph $\mathcal{H}$. A *reduct* G' of G is a graph (V', E') such that $V' = V$, $E' \subseteq E$, and the following condition holds: for each edge $q = \{h, h'\}$ belonging to $(E - E')$, there exists in G' a path P from h to h' , such that the variables in $\ell(q)$ are included in $\ell(q')$ for each edge q' occurring in the path P . That is, if all the variables shared by h and h' occur in some other path between these vertices, the edge connecting them can be deleted from the dual graph.

Let h be a hyperedge of $\mathcal{H}$, and G be a reduct of $dual(\mathcal{H})$. Then, we denote with $v_G(h)$ the vertex of G' corresponding to the hyperedge h . Moreover, given a set of hyperedges H , we denote by $v_G(H)$ the set of vertices of G' associated with the elements in H . Whenever no confusion arises, $v_G(h)$ (resp. $v_G(H)$) will be denoted simply by $v(h)$ (resp. $v(H)$). Finally, we denote by $dual_G(H)$ the subgraph of G induced by the nodes in $v_G(H)$.

A non-binary CSP can be solved by applying any binary decomposition method, say D , to any reduct of the dual graph. However, the performance of this method depends in general on the particular reduct selected for solving the problem. We are interested in the reduct that guarantees the best possible performances, when the method D is applied to it. Formally, given a hypergraph $\mathcal{H}$, define D^{optd} as the method D applied to the reduct of $dual(\mathcal{H})$ having the minimum D -width over all the possible reducts of $dual(\mathcal{H})$.

It can be seen that the strongly generalization results between graph-based decomposition methods proved by Gottlob et al. [2000] can be extended to their *optd* versions.

Theorem 2.2 The following relationships hold:

- $BICOMP^{optd} \ll TREewidth^{optd}$.
- $CUTSET^{optd} \ll TREewidth^{optd}$.

Thus, $TREewidth^{optd}$ seems a good candidate for solving non-binary CSP, as it strongly generalizes all the other graph-based methods. However, we recall that all the known algorithms for computing a k -bounded tree decomposition of a graph are exponential in k (even if they are polynomial for any fixed constant k), while computing the biconnected components of a graph is a linear task, independently of their width. This latter technique can be therefore very useful if the size of the structure and the bound k are large and the powerful methods like $TREewidth$ are too expensive. In the next section, we face the problem of computing optimal reducts of the dual graph w.r.t. the $BICOMP$ method.

3 Hinges VS Biconnected Components

In [Gyssens et al., 1994], it has been shown that the *HINGE* method generalizes $BICOMP$ applied to any reduct of the dual graph. Anyhow, in the same paper, Gyssens et al. observed that a fine comparison between the two methods is quite difficult, as there is no obvious way to find a suitable reduct of the dual graph to keep the biconnected width small. Here we complete the picture by showing that, in fact, hinge decompositions correspond to such clever simplifications of the dual graph.

Lemma 3.1 $HINGE \preceq BICOMP^{optd}$.

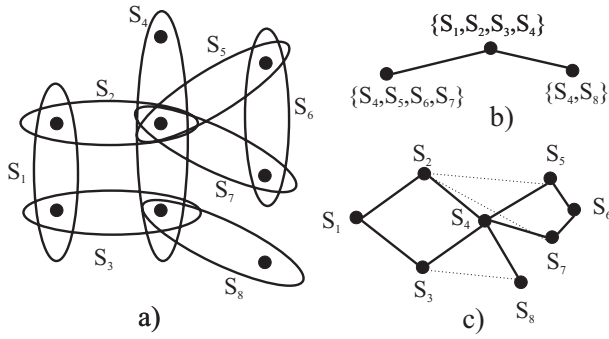


Figure 1: (a) a hypergraph $\mathcal{H}'$, (b) a hinge decomposition of $\mathcal{H}'$, and (c) a reduct of $\text{dual}(\mathcal{H}')$

Proof. (Sketch.) Let $\mathcal{H}$ be a hypergraph and T a hinge decomposition. Let G be a graph constructed by removing from $\text{dual}(\mathcal{H})$ each edge $\{v(h), v(h')\}$ such that there exists two distinct nodes of T , say H and H' , such that $h \in H$ and $h' \in H'$.

We claim that i) G is a reduct of $\text{dual}(\mathcal{H})$, and ii) for each hinge H in T , there exists a biconnected component C of G , such that C is the subgraph of G induced by the nodes $v_G(H)$, i.e. $C = \text{dual}_G(H)$. Moreover, every biconnected component of G corresponds to some hinge, because each node of G is an edge of the original hypergraph, and thus must be contained in some hinge of T , by definition of hinge decomposition. Finally, note that the size (number of edges) of any hinge H , is the same as the size (number of nodes) of the corresponding biconnected component $C = \text{dual}_G(H)$. $\square$

Combining the above lemma and the results in [Gyssens *et al.*, 1994], it follows that these two methods identifies the same classes of tractable CSPs.

Theorem 3.2 $\text{BICOMP}^{\text{optd}} \equiv \text{HINGE}$.

It is worthwhile noting that, given any hinge decomposition of a hypergraph $\mathcal{H}$, the proof of Lemma 3.1 provides in fact an algorithm for computing an optimal reduct of $\text{dual}(\mathcal{H})$ with respect to the BICOMP method. For instance, Figure 1 shows a hypergraph $\mathcal{H}'$, a hinge decomposition of $\mathcal{H}'$, and the optimal reduct of the dual graph obtained by applying the above construction, where dotted edges represent the edges of the dual graph removed in this simplification. Note that the biconnected components of this graph correspond to the hinges of the given decomposition. Since biconnected components can be computed in linear time, it follows that a $\text{BICOMP}^{\text{optd}}$ decomposition of a hypergraph $\mathcal{H}$ can be computed in time $O(|\mathcal{N}(\mathcal{H})||\mathcal{E}(\mathcal{H})|^2)$, which is the best known upper bound for computing a hinge decomposition.

Observe that the above result, together with Theorem 2.2, gives us a new insight of the power of $\text{TREEWIDTH}^{\text{optd}}$. Indeed, while the tree decomposition method applied to the primal graph is incomparable with HINGE [Gottlob *et al.*, 2000], its application to an optimal reduct of the dual graph strongly generalizes HINGE.

Corollary 3.3 $\text{HINGE} \prec \text{TREEWIDTH}^{\text{optd}}$.

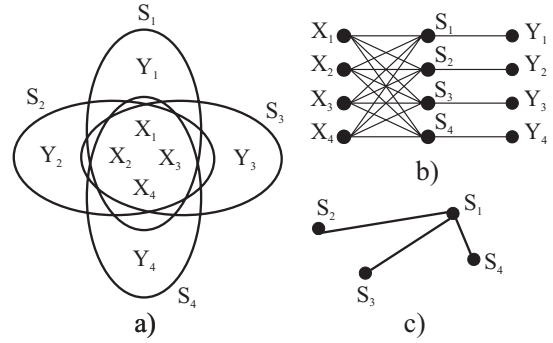


Figure 2: (a) The hypergraph $\text{Rose}(4)$, (b) its incidence graph, and (c) a reduct of $\text{dual}(\text{Rose}(4))$

4 Hidden-Variables VS Dual Graph

It is well known that both the dual graph and the hidden-variable (incidence-graph) representations may be used for identifying tractable classes of non-binary CSPs according to the tree decomposition method (see, e.g., [Kolaitis and Vardi, 2000]). However, it was not clear whether either of these methods generalizes the other one or beats the other one on some classes of CSPs. In this section, we precisely characterize the relationship between $\text{TREEWIDTH}^{\text{optd}}$ and $\text{TREEWIDTH}^{\text{in}}$. First, we observe that there is a CSP class where $\text{TREEWIDTH}^{\text{optd}}$ is definitely better than $\text{TREEWIDTH}^{\text{in}}$.

Theorem 4.1 $\text{TREEWIDTH}^{\text{optd}} \triangleright \text{TREEWIDTH}^{\text{in}}$.

Proof. (Sketch.) We show that there is a class of hypergraphs that is tractable w.r.t. $\text{TREEWIDTH}^{\text{optd}}$, but not w.r.t. $\text{TREEWIDTH}^{\text{in}}$. For any $n > 0$, let $\text{Rose}(n)$ be the hypergraph having n edges $\{h_1, \dots, h_n\}$ over the nodes $\{X_1, \dots, X_n, Y_1, \dots, Y_n\}$, such that $h_i = \{X_1, \dots, X_n, Y_i\}$, for each $0 < i \leq n$. Figure 2 shows the graph $\text{Rose}(4)$, its incidence graph and a reduct of its dual graph. It can be seen that any optimal reduct of $\text{dual}(\text{Rose}(n))$ is a tree and hence its width according to $\text{TREEWIDTH}^{\text{optd}}$ is 1. On the other hand, the bipartite graph $\text{inc}(\text{Rose}(n))$ contains a clique of size n and hence its treewidth is $n - 1$. It follows that the class $\{\text{Rose}(n) \mid n > 0\}$ has unbounded treewidth. $\square$

Note that any hypergraph $\text{Rose}(n)$ in the above proof is an acyclic hypergraph. It follows that the class of all acyclic CSP instances is not tractable according to $\text{TREEWIDTH}^{\text{in}}$.

Even though $\text{TREEWIDTH}^{\text{optd}}$ beats $\text{TREEWIDTH}^{\text{in}}$, we next shows that $\text{TREEWIDTH}^{\text{optd}}$ does not generalize $\text{TREEWIDTH}^{\text{in}}$.

Theorem 4.2 $\text{TREEWIDTH}^{\text{in}} \not\prec \text{TREEWIDTH}^{\text{optd}}$.

Proof. (Sketch.) We show that there is a class of hypergraphs $\text{Subset}(k)$ such that the treewidth of the incidence graph is k , but the treewidth of any optimal reduct of $\text{dual}(\text{Subset}(k))$ is $(k/3)^2$. For any k , the hypergraph $\text{Subset}(k)$ is such that $\mathcal{N}(\text{Subset}(k)) = X \cup Y \cup Z$, where $X = \{X_1, \dots, X_m\}$, $Y = \{Y_1, \dots, Y_m\}$, and $Z = \{Z_1, \dots, Z_m\}$, with $m = k/3$.

There are m^2 distinct edges, each containing $m - 1$ nodes from X , $m - 1$ nodes from Y , and $m - 1$ nodes from Z . In particular, each edge e contains one of the m subsets of $m - 1$

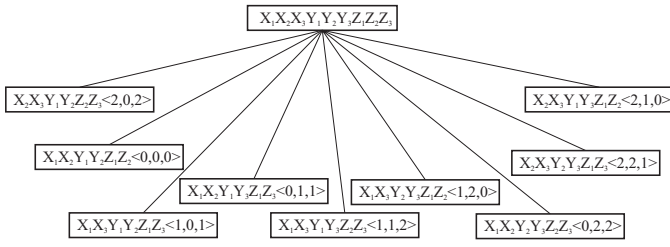


Figure 3: A tree decomposition of $\text{inc}(\text{Subset}(9))$.

elements from X and one the m subsets of $m - 1$ elements from Y . These subsets may be identified by means of two integers, say a and b , ranging from 0 to $m - 1$. Similarly, the edge contains a subset of variables from Z identified by an integer c and functionally determined by a and b as follows: $c = (a + b) \bmod m$. Thus, each edge e is simply denoted as a triple $\langle a, b, c \rangle$.

Note that the incidence treewidth of $\text{Subset}(k)$ is k . For instance, Figure 3 shows a tree decomposition of the incidence graph of $\text{Subset}(9)$, where $m = 3$. Moreover, it can be proved that the dual graph $\text{dual}(\text{Subset}(k))$ cannot be reduced and contains a clique of size $m^2 = (k/3)^2$. It follows that its treewidth is $(k/3)^2$. $\square$

To give a complete picture of the relationship between $\text{TREEWIDTH}^{\text{optd}}$ and $\text{TREEWIDTH}^{\text{in}}$, we next show that any decomposition of the incidence graph with width k can be modified to be a decomposition of a reduct of the dual graph having width at most 2^k . Thus, $\text{TREEWIDTH}^{\text{in}}$ does not beat $\text{TREEWIDTH}^{\text{optd}}$.

The algorithm is shown in Figure 4. It takes a tree decomposition $TD = \langle T, \chi \rangle$ of $\text{inc}(\mathcal{H})$, and outputs a tree decomposition $TD' = \langle T, \chi' \rangle$ of a suitable reduct of $\text{dual}(\mathcal{H})$.

It performs a breadth-first visit of T . For each visited node p , we compute the labelling $\chi'(p)$ of this node in the new tree decomposition TD' from its label $\chi(p)$ in the given tree decomposition TD : Let $\chi(p) = V_p \cup C_p$, where $V_p \subseteq \mathcal{N}(\mathcal{H})$ and $C_p \subseteq \mathcal{E}(\mathcal{H})$. Then, $\chi'(p)$ contains all edges in C_p plus one arbitrarily chosen edge $h_{V'}$ that covers V' (i.e., such that $V' \subseteq h_{V'}$), for each subset V' of V_p . This way, p is now labelled only by edges of $\mathcal{H}$ (i.e., nodes of the dual graph). Moreover, the algorithm forces all such edges $h_{V'}$ to belong to the χ' labellings of any vertex q in the subtree rooted at p such that $V' \subseteq \chi(q)$.

Note that, if $\langle T, \chi \rangle$ has width k , then $\langle T, \chi' \rangle$ has width at most 2^k . Then, from this algorithm (whose proof of correctness is omitted), we get the following relationship.

Theorem 4.3 $\text{TREEWIDTH}^{\text{in}} \not\preceq \text{TREEWIDTH}^{\text{optd}}$.

5 A 2-approximation of $\text{TREEWIDTH}^{\text{optd}}$

In this section, we face the problem of computing an optimal reduct of a dual graph in order to get the minimum possible treewidth. We recall that it is not known whether this problem is feasible in polynomial time or not [Kolaitis and Vardi, 2000]. Moreover, we provide a qualitative explanation of the remarkable difference between $\text{TREEWIDTH}^{\text{optd}}$

Input: A tree decomposition $\langle T, \chi \rangle$ of $\text{inc}(\mathcal{H})$;
Output: A tree decomposition $\langle T, \chi' \rangle$ of a reduct of $\text{dual}(\mathcal{H})$;

```

for all  $p$  in  $T$  and all  $V' \subseteq \chi(p)$  do
   $\text{cover}(V') := \emptyset$ ;
 $\text{queue} := \text{root}(T)$ ;
while  $\text{queue} \neq \emptyset$  do
  extract  $p$  from  $\text{queue}$ ;
   $V_p := \chi(p) \cap \mathcal{N}(\mathcal{H})$ ;
   $C_p := \chi(p) \cap \mathcal{E}(\mathcal{H})$ ;
   $\chi'(p) := C_p$ ;
  for all  $V' \subseteq V_p$  s.t. there is  $h \in \mathcal{E}(\mathcal{H})$  s.t.  $V' \subseteq h$  do
    if  $\text{cover}(V') = \emptyset$  then
      choose any  $h \in \mathcal{E}(\mathcal{H})$  s.t.  $V' \subseteq h$ ;
       $\chi'(p) := \chi'(p) \cup \{h\}$ ;
       $\text{cover}(V') := \{h\}$ ;
    else
       $\chi'(p) := \chi'(p) \cup \text{cover}(V')$ ;
    end if
   $\text{insert}(\text{queue}, \text{children}(p))$ ;
end while
return  $\langle T, \chi' \rangle$ 

```

Figure 4: **Algorithm** incidenceToDualTreeDecomposition

and hypergraph-based notions such as query width and hypertree width.

Let $k > 0$ be a fixed constant and $\mathcal{H}$ be a constraint hypergraph. Figure 5 shows the Algorithm k-TREE-APPROX that, if the optimal treewidth of $\text{dual}(\mathcal{H})$ is at most k , outputs in polynomial time a tree decomposition of width at most $2k$ of some reduct of $\text{dual}(\mathcal{H})$.

Let S be a subset of nodes of $\text{dual}(\mathcal{H})$. A non-empty set C of nodes of $\text{dual}(\mathcal{H})$ is [S]-connected if, for each pair $h, h' \in C$, there exists a path $h = h_1, \dots, h_n = h'$ in $\text{dual}(\mathcal{H})$ such that $h_i \cap h_{i+1} \not\subseteq z$, for every $z \in S$. The [S]-components of $\text{dual}(\mathcal{H})$ are the maximal [S]-connected sets of nodes of $\text{dual}(\mathcal{H})$.

Roughly, k-TREE-APPROX is based on a recursive procedure $k\text{Decomp}$ that, given a vertex p and a $[\chi(p)]$ -component C to be “decomposed,” computes the labelling $\chi(q)$ of a child q of p chosen to deal with C . Then, it recursively call itself for computing the subtree rooted at q that should cover all the $\chi(q)$ -components included in C .

Finally, if the output $\langle T, \chi \rangle$ of $k\text{Decomp}$ is not a tree decomposition of some reduct of $\text{dual}(\mathcal{H})$, then the MAIN module of the algorithm computes a new labelling χ' that is guaranteed to be a legal tree decomposition.

The proof of the following theorem is quite involved and will be omitted for space limitation.

Theorem 5.1 Let k be a fixed integer. Given any hypergraph $\mathcal{H}$, if there exists a reduct of $\text{dual}(\mathcal{H})$ having treewidth bounded by k , then k-TREE-APPROX computes in L^{LOGCFL} a tree decomposition of some reduct of $\text{dual}(\mathcal{H})$ having width at most $2k$.

It is worthwhile noting that the output $\langle T, \chi \rangle$ of $k\text{Decomp}$, even if it is not a tree decomposition, is in fact a new kind of structural decomposition, defined below, that we call *weak query decomposition*.

Definition 5.2 A weak query decomposition of a hypergraph $\mathcal{H}$ is a pair $\langle T, \lambda \rangle$, where $T = (N, E)$ is a tree, and λ is a

```

Input: A hypergraph  $\mathcal{H}$ ;
Output: A tree decomposition of a reduct of  $dual(\mathcal{H})$ , having size  $\leq 2 \times k$ ;
var  $T$ : tree,  $\chi$ : vertices of  $T \mapsto 2^{\mathcal{N}(dual(\mathcal{H}))}$ ;
Procedure  $kDecomp(C \subseteq \mathcal{N}(dual(\mathcal{H})), p \in T)$  : boolean;
begin
  guess  $S \subseteq \chi(p) \cup C$ , with  $|S| \leq k$ ;
  check that the following conditions hold
    C1:  $S \cap C \neq \emptyset$ , and
    C2:  $\forall \{h, h'\} \in dual(\mathcal{H})$ , s.t.  $h \in C$  and  $h' \in \chi(p)$ ,
        there exists  $h'' \in S$  with  $\ell(\{h, h'\}) \subseteq h''$ ;
  if the check fails then return false;
  for each  $[S]$ -component  $C' \subseteq C$  do
     $T.insertChild(p, q)$ , with  $\chi(q) = S$ ;
    if not  $kDecomp(C', q)$  return false;
  return true;
end;
begin (MAIN)
   $kDecomposable := kDecomp(\mathcal{N}(dual(\mathcal{H})), \perp)$ ;
  if not  $kDecomposable$  then return false;
  if  $\langle T, \chi \rangle$  is a tree decomposition of some reduct of  $dual(\mathcal{H})$  then
    return  $\langle T, \chi \rangle$ 
  else for each vertex  $p \in T$  do
    for each child  $q$  of  $p$ , let  $\chi'(q) := \chi(q) \cup \chi(p)$ ;
    return  $\langle T, \chi' \rangle$ ;
end.

```

Figure 5: Algorithm k-TREE-APPROX

labelling function which associates to each vertex $p \in N$ a set $\lambda(p) \subseteq \mathcal{E}(\mathcal{H})$, such that the following conditions hold:

1. for each edges h of $\mathcal{H}$, there exists $p \in N$ such that $h \in \lambda(p)$;
2. for each edge h of $\mathcal{H}$, the set $\{p \in N \mid h \in \lambda(p)\}$ induces a (connected) subtree of T ;
3. for each pair of vertices v, v' of T and each pair of edges $h \in \lambda(v)$, $h' \in \lambda(v')$ there is a path P connecting v and v' in T such that, for each v in P , there exists $h'' \in \lambda(v)$ such that $\ell(\{h, h'\}) \subseteq h''$.

The width of the weak query decomposition $\langle T, \lambda \rangle$ is $\max_{p \in N} |\lambda(p)|$. The weak query-width $wqw(\mathcal{H})$ of $\mathcal{H}$ is the minimum width over all its weak query decompositions.

Note that Condition 3 above entails the usual *connectedness condition 3'*: for each variable $Y \in \mathcal{N}(\mathcal{H})$, the set $\{p \in N \mid Y \text{ occurs in some edge } h \in \lambda(p)\}$ induces a (connected) subtree of T . Thus, any weak query decomposition of a hypergraph is also a query decomposition of the hypergraph, and thus any class of CSPs having bounded weak query-width is tractable, i.e., can be solved in polynomial time. However, unlike query decompositions [Gottlob *et al.*, 2002], weak query decompositions are efficiently computable. Indeed, LOGCFL is not only included in polynomial time, but also in NC^2 , and thus it contains highly parallelizable problems. Moreover, this new notion has essentially the same “decomposition power” as $TREEWIDTH^{optd}$.

Theorem 5.3 For any hypergraph $\mathcal{H}$,

$$wqw(\mathcal{H}) \leq tw^{optd}(\mathcal{H}) \leq 2wqw(\mathcal{H}).$$

Thus, $TREEWIDTH^{optd}$ and the notion of weak query decomposition are almost equivalent, up to a bounded difference in favor of the latter method.

Remark. The results in this section shed some light on the difference between the “decomposition power” of $TREEWIDTH^{optd}$ and the strictly more general methods of query and hypertree decompositions. Note that the crucial condition to be maintained in all these tree-structured decompositions is the connectedness condition for the constraint variables, and looking at Condition 3 and Condition 3' above we can make the following observation: while in the weak query decompositions (and similarly in $TREEWIDTH^{optd}$) each hyperedge in the labelling of a tree node plays an independent role, in query decompositions (and hypertree decompositions) all such edges contribute together to maintain the connectedness condition. That is, these hypergraph based notions exploit the union of the hyperedges labelling any tree-node, while the $TREEWIDTH^{optd}$ and the weak query width methods exploit the power of each hyperedge separately.

References

- [Bacchus *et al.*, 2002] F. Bacchus, X. Chen, P. van Beek, and T. Walsh. Binary vs Non-Binary Constraints. *Artificial Intelligence*, 140: (1-2), 1-37, 2002.
- [Chekuri and Rajaraman, 2000] Ch. Chekuri and A. Rajaraman. Conjunctive Query Containment Revisited. *Theoretical Computer Science*, 239(2):211–229, 2000.
- [Dechter, 1992] R. Dechter. Constraint Networks. In *Encyclopedia of Artificial Intelligence*, second edition, Wiley and Sons, pp. 276–285, 1992.
- [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.
- [Freuder, 1985] E.C. Freuder. A sufficient condition for backtrack-bounded search. *Journal of the ACM*, 32(4):755–761, 1985.
- [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.*, 2002] G. Gottlob, N. Leone, and F. Scarcello. Hypertree decompositions and tractable queries, *Journal of Computer and System Sciences*, 64(3): 579–627, 2002.
- [Gyssens *et al.*, 1994] M. Gyssens, P.G. Jeavons, and D.A. Cohen. Decomposing constraint satisfaction problems using database techniques. *Artificial Intelligence*, 66:57–89, 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(2): 302–332, 2000.
- [Pearson and Jeavons, 1997] J. Pearson and P.G. Jeavons. A Survey of Tractable Constraint Satisfaction Problems, CSD-TR-97-15, Royal Holloway, Univ. of London, 1997.
- [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.
- [Seidel, 1981] R. Seidel. A new method for solving constraint satisfaction problems. In *Proc. of IJCAI'81*, 1981.

Algorithms for Identifying Rigid Subsystems in Geometric Constraint Systems

Christophe Jermann

AI Lab.

EPFL, 1015 Lausanne

Switzerland

e-mail: Christophe.Jermann@epfl.ch

Bertrand Neveu and Gilles Trombettoni

COPRIN Team, INRIA-I3S/CNRS-CERMICS,

2004 route des lucioles, BP 93,

06902 Sophia Antipolis, France

e-mail: Bertrand.Neveu@sophia.inria.fr,

Gilles.Trombettoni@sophia.inria.fr

Abstract

The structural rigidity property, a generalization of Laman's theorem which characterizes rigid bar frameworks in 2D, is generally considered a good approximation of rigidity in geometric constraint satisfaction problems (GCSPs). However, it may fail even on simple GCSPs because it does not take geometric properties into account.

In this paper, we question the flow-based algorithm used by Hoffmann *et al.* to identify rigid subGCSPs. We show that this algorithm may fail because of the structural rigidity, but also by design. We introduce a new flow-based algorithm which uses Jermann *et al.*'s characterization of rigidity. We show that this algorithm is correct in 2D and 3D, and can be used to tackle the major issues related to rigidity: deciding whether a GCSP is rigid or not and identifying rigid (or over-rigid) subGCSPs.

Keywords: Geometric Constraints, Rigidity characterization, Flow algorithms

1 Introduction

Geometric constraint satisfaction problems (GCSPs) arise naturally in several areas, such as CAD, robotics and molecular biology. The rigidity concept is in the heart of many of these problems: deciding whether a GCSP is rigid or not, detecting rigid or over-rigid sub-parts, and so on.

Several methods [Kramer, 1992; Bouma *et al.*, 1995; Dufourd *et al.*, 1998; Lamure and Michelucci, 1998; Hoffmann *et al.*, 2000; Jermann *et al.*, 2000; Jermann, 2002] for solving GCSPs have to deal with rigidity; e.g., geometric decompositions produce sequences of rigid subGCSPs to be solved separately and then assembled.

The techniques used for rigidity detection can be classified in two categories: pattern-based approaches [Bouma *et al.*, 1995; Kramer, 1992] depend on a repertoire of rigid bodies of known *shape* which cannot cover all practical instances; flow-based approaches [Hoffmann *et al.*, 1997; Lamure and Michelucci, 1998] use flow (or maximum matching) machinery to identify subGCSPs verifying the *structural rigidity*, a property based on a degree of freedom count.

The latter approaches are more general even though structural rigidity is only an approximation of rigidity. Heuristics,

like *ad-hoc* geometric rules, have been proposed to enhance structural rigidity capabilities, none of which succeeded to cover the gap between structural rigidity and rigidity. In [Jermann *et al.*, 2002], we have defined the *extended structural rigidity*, a new approximation of rigidity which supersedes even the heuristically enhanced characterizations.

In this paper, we focus on the algorithmic aspects of the structural characterization of rigidity. [Hoffmann *et al.*, 1997] have proposed a flow-based algorithm called **Dense** for this purpose. After providing the necessary background (Section 2), we exemplify the limits of this algorithm and the capabilities of our new algorithm (Section 3). Section 4 presents the specificities of our new algorithm and explains its advantages: it uses the extended structural rigidity instead of the structural rigidity and it is designed in a geometrically correct way. To conclude, we explain how this algorithm can be used to tackle the major issues related to rigidity.

2 Background

This section provides the necessary background for the paper. It formally defines GCSPs, the rigidity concept and the structural characterizations of rigidity.

2.1 Geometric Constraint Satisfaction Problems

Definition 1 GCSP

A **geometric constraint satisfaction problem** (GCSP) $S = (O, C)$ is defined by a set O of geometric objects and a set C of geometric constraints binding its objects.

$S' = (O', C')$ is a **subGCSP** of $S = (O, C)$ (noted $S' \subset S$) iff $O' \subset O$ and $C' = \{c \in C | c \text{ binds only objects in } O'\}$ (i.e., S' is induced by O').

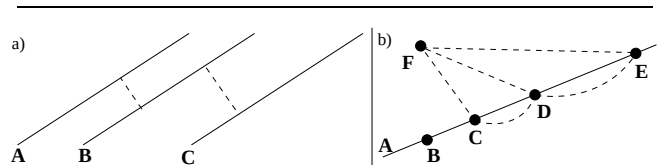


Figure 1: Two examples of GCSPs

Fig. 1-a presents a GCSP in 2D composed of 3 lines constrained by 2 parallelisms and 2 line-line distances; Fig. 1-b

depicts a GCSP in 3D composed of 1 line and 5 points bound by 4 point-line incidences and 5 point-point distances.

We assume that geometric objects are indeformable (e.g., no circle with variable radius). Also geometric constraints must involve only positions and orientations of the objects and they must be independent from the global reference system (i.e., constraints only fix objects relatively one to another). These limitations make the structural characterizations of rigidity easier and are mandatory for geometric solving methods based on rigidity.

According to these restrictions, a solution to a GCSP $S = (O, C)$ is composed of one position and orientation for each object in O and satisfies all the constraints in C . For the solving purpose, a GCSP is translated into a system of equations: each object is represented by a set of unknowns (over the reals) which determine its position and orientation; each constraint becomes a system of equations on the unknowns of the objects it constrains.

2.2 Rigidity

Rigidity is defined w.r.t. *movements*. A movement in a GCSP is either a *deformation* (if it does not preserve the relative positions of the objects) or a *displacement* (rotation+translation). Intuitively, a GCSP is *rigid* if it admits no deformation, and all the displacements of the geometric space. It is *under-rigid* if it admits some deformations, and *over-rigid* if it does not admit some displacements or has no solution. More formal definitions of rigidity can be found in [Whiteley, 1987].

In Fig. 1-b, the subGCSP CDF is rigid since a triangle is indeformable and admits all translations and rotations in 3D. The subGCSP AF is under-rigid: point F can move independently of line A since there is no constraint between them. The subGCSP $ACDEF$ is over-rigid since it has no solution: generically, it is impossible to place a point at the intersection of the 3 spheres (= 3 distance constraints) which centers are aligned.

2.3 Structural Rigidity

The *structural rigidity* corresponds to an analysis of *degrees of freedom* (DOF) in a GCSP. Intuitively, one DOF represents one independent movement in a GCSP. More formally:

Definition 2 Degree of freedom (DOF)

- Object o : $DOF(o)$ is the number of independent parameters used to determine the position and the orientation of o .
- Constraint c : $DOF(c)$ is the number of independent equations in the subsystem of equations representing c .
- GCSP $S = (O, C)$: $DOF(S) = \sum_O DOF(o) - \sum_C DOF(c)$.

In 3D, points have 3 DOFs, lines have 4 DOFs; point-line incidences remove 2 DOFs, and point-point distances remove 1 DOF. Thus, subGCSPs ACD , CDF and AF from the GCSP in Fig. 1-b have respectively 5, 6 and 7 DOFs.

Structural rigidity is a generalization of Laman's theorem [Laman, 1970], which characterizes generic rigidity of 2D bar frameworks. It is based on the following intuition: if a GCSP admits less (resp. more) movements than the number of independent displacements in the considered geometric

space (which is $\frac{d(d+1)}{2}$ in dimension d), then it is over-(resp. under-)rigid.

Definition 3 Structural rigidity ($s_rigidity$)

A GCSP $S = (O, C)$ in dimension d is **s_rigid** iff $DOF(S) = \frac{d(d+1)}{2}$ and $\forall S' \subset S, DOF(S') \geq \frac{d(d+1)}{2}$.

S is under- s_rigid iff $DOF(S) > \frac{d(d+1)}{2}$ and contains no over- s_rigid subGCSP.

S is over- s_rigid iff $\exists S' \subseteq S, DOF(S') < \frac{d(d+1)}{2}$.

In practice, structural rigidity is considered a *good approximation* of rigidity [Lamure and Michelucci, 1998; Hoffmann *et al.*, 1997]. However, the gap between rigidity and $s_rigidity$ is in fact significant (see [Jermann *et al.*, 2002]). We illustrate the difference on two subGCSPs from Fig. 1-b: $ABCD$ is s_rigid in 3D since $DOF(ABCD)=6$; it is in fact under-rigid since point B can move independently of segment CD along line A . $ACDE$ is over- s_rigid since $DOF(ACDE)=5$, but it is in fact well-rigid.

2.4 Extended Structural Rigidity

The extended structural rigidity ($es_rigidity$ in short) is based on the *degree of rigidity* (DOR) concept. The DOR of a subGCSP is the number of independent displacements it admits; it depends on the geometric properties it verifies. For example, the DOR of two lines in 2D is 3 if they are not parallel, 2 if they are parallel; the parallelism property can be an explicit constraint, but it can also be induced by the constraints of the GCSP embedding these lines. In this second case, computing the DOR may be equivalent to geometric theorem proving.

The principle behind the extended structural rigidity is the following: a GCSP is rigid if all its movements are displacements. Hence, comparing its DOF to its DOR allows us to determine if it admits movements (DOF) which are not displacements (DOR), i.e., deformations.

Definition 4 Extended Structural Rigidity ($es_rigidity$)

A GCSP $S = (O, C)$ in dimension d is **es_rigid** iff $DOF(S)=DOR(S)$ and $\forall S' \subset S, DOF(S') \geq DOR(S')$.

S is under- es_rigid iff $DOF(S) > DOR(S)$ and contains no over- es_rigid subGCSP.

S is over- es_rigid iff $\exists S' \subseteq S, DOF(S') < DOR(S')$.

The $es_rigidity$ is superior to the $s_rigidity$ (e.g., $es_rigidity$ exactly corresponds to rigidity on every subGCSPs in Fig. 1). See [Jermann *et al.*, 2002] for a comparison between $s_rigidity$ and $es_rigidity$ and details about the DOR concept.

2.5 Object-Constraint Network

A GCSP $S = (O, C)$ can be transformed into an *object-constraint network* $G = (s, V, t, E, w)$. Fig. 2-a depicts the object-constraint network of the GCSP presented in Fig. 1-a. A flow in this network (introduced in [Hoffmann *et al.*, 1997]) represents a distribution of the DOFs of the constraints onto the DOFs of the objects. This is why flow computation is used to perform the DOF analysis which is at the basis of structural characterization of rigidity: in Section 4, we will explain how this network is used for $s_rigidity$ ([Hoffmann *et al.*, 1997]'s Dense algorithm) and $es_rigidity$ (our new algorithm).

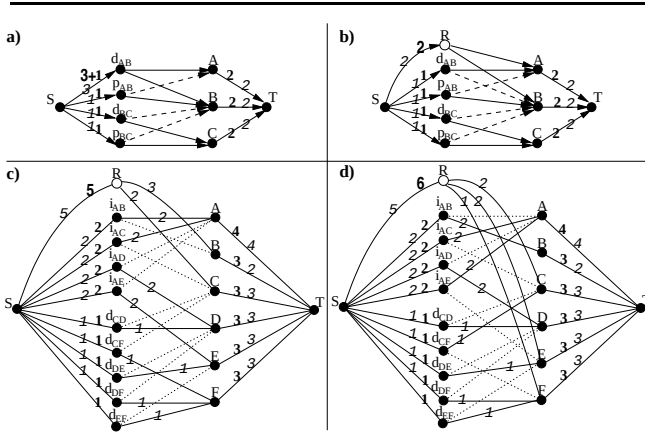


Figure 2: Object-constraint networks and flow distributions

Definition 5 Object-Constraint Network (s, V, t, E, w)

- s is the source and t is the sink.
- Each object $o \in O$ becomes an object-node $v_o \in V$.
- Each constraint $c \in C$ becomes a constraint-node $v_c \in V$.
- For each object $o \in O$ there is an arc $v_o \rightarrow T$ of capacity $w(v_o \rightarrow T) = \text{DOF}(o)$ in E .
- For each constraint $c \in C$, there is an arc $S \rightarrow v_c$ of capacity $w(S \rightarrow v_c) = \text{DOF}(c)$ in E .
- For each object $o \in O$ constrained by $c \in C$ there is an arc $v_c \rightarrow v_o$ of capacity $w(v_c \rightarrow v_o) = \infty$ in E .

3 Overview

In this section, we exemplify the contribution presented in this paper on the GCSPs in Fig. 1. We illustrate the two main differences between algorithm *Dense* and our algorithm:

1. In our algorithm, the overflow depends on the geometric properties of the objects it is applied to, while it depends only on the dimension of the geometric space in *Dense*.
2. In our algorithm, the overflow is applied via a dedicated node R which can be attached to any subset of objects, while it is applied directly via one constraint-node in *Dense*.

Example 1

The first example (Fig. 1-a; in 2D) highlights the first difference. Fig. 2-a presents the object-constraint network associated to this GCSP. In this picture, one can see the overload $K = 3$ applied on the first constraint by algorithm *Dense*. This constraint is linked to two lines, A and B , which are parallel and lie at prescribed distance in the plane; AB is a rigid subGCSP. However, one can easily see that the overload cannot be distributed completely since a capacity 5 (two constraints plus the overload) is applied to two lines having only 4 DOFs. Hence, the GCSP is identified as over-rigid since it contains a sub-GCSP with less than 3 DOFs.

Fig. 2-b displays our algorithm behavior when the virtual constraint R is linked to the same subGCSP, AB . The value of the overflow K is computed according to the geometric

properties of these lines: since they are parallel, $K = 2$ (instead of 3 in algorithm *Dense*). Thus, the flow can be saturated: a capacity 4 (two constraints plus the overflow) exactly matches the 4 DOFs of AB ; the GCSP is not identified over-rigid by our algorithm. Further overflow applications would allow to identify the GCSP as well-rigid.

Example 2

The second example (Fig. 1-b; in 3D) illustrates the second difference. Its object-constraint network is depicted in Fig. 2-d. This figure shows the application of an overflow 6 via the virtual constraint R onto the 3 points C , E and F by our algorithm; the overflow cannot be distributed completely, which signals an over-rigid subGCSP: $ACDEF$, found by adding reachable objects from R in the residual graph.

Algorithm *Dense* applies the overflow directly through a constraint-node. Since all constraints are binary in this example, *Dense* cannot apply an overflow to the same set of objects as our algorithm. More generally, *Dense* cannot apply the overflow to all subGCSPs and can miss rigid or over-rigid ones. Moreover, applying the overflow 6 to a pair of objects in this GCSP leads to an incorrect answer, as it was the case in the previous example; e.g., segments which are rigid would be identified over-rigid.

These examples show that some simple and very common subGCSPs in 2D and 3D, like parallel lines, triangles or segments, cannot be treated correctly by algorithm *Dense*¹.

In the following section, we detail the differences between algorithm *Dense* and our new algorithm and we present their consequences.

4 Algorithms

In this section, we present Hoffmann *et al.*'s *Dense* algorithm in comparison to our new algorithm. Both use flow machinery on the object-constraint network representing the GCSP. Our algorithm has two main differences with algorithm *Dense*:

- It uses *es_rigidity* instead of *s_rigidity*.
- It distributes flow in a *geometrically correct* way in the network.

These new features are achieved thanks to two major modifications in the *Distribute* function used by *Dense* (see beginning of Section 3).

We introduce first the principle of flow-based characterization of rigidity; then we present and discuss function *Distribute* which is the key to our contribution. Finally we explain how this function is used to design algorithms for the main problems related to rigidity.

4.1 Flow-based Rigidity Detection

From the geometric point of view, the principle of structural characterization of rigidity is to check if a GCSP admits only displacements. Hence, flow-based rigidity identification can be understood as follows:

¹In practice [Sitharam, 2000], *Dense* embeds heuristic rules to prevent this kind of simple failures, but more complicated examples can still mistake the algorithm since no rule-based approach can handle all the singular cases.

1. remove K displacements from the GCSP by introducing K DOFs on the constraint side;
2. check if an over-constrained subGCSP S' exists by computing a maximum flow in the overloaded object-constraint network;
3. if so S' verifies $\text{DOF}(S') < K$.

Indeed, a maximum flow in the object-constraint network represents an *optimal* distribution of the DOFs of the constraints among the DOFs of the objects. If it does not saturate all the arcs outgoing from the source, some constraints' DOFs cannot be absorbed by the objects, i.e., the GCSP is over-constrained. In this case, there exists a subGCSP S' such that $\text{DOF}(S') < 0$. When an overflow K is applied in the network on the constraint side, the identified subGCSP S' verifies $\text{DOF}(S') < K$. [Hoffmann *et al.*, 1997] have proven that S' is then induced by the objects traversed during the last search for an augmenting path, i.e., by the objects reachable from the overloaded constraint-node in the residual graph.

Depending on the value of K , this principle can be applied to identify s_rigid ($K = \frac{d(d+1)}{2} + 1$), over-s_rigid ($K = \frac{d(d+1)}{2}$), es_rigid ($K = \text{DOR} + 1$) or over-es_rigid ($K = \text{DOR}$) subGCSPs.

4.2 Function Distribute

Function **Distribute** [Hoffmann *et al.*, 1997] implements the principle presented above. We present our version of this function and explain why and how it differs from Hoffmann *et al.*'s one.

As already said, applying an overflow K corresponds, from the geometric point of view, to removing K displacements from the objects linked to this constraint. But nothing ensures that the subGCSP linked to a single constraint allows K independent displacements: removing K DOFs from a subGCSP S' with $\text{DOR}(S') < K$ is *geometrically incorrect*².

For instance, consider a subGCSP composed of 2 points linked by a point-point distance in 3D. This GCSP allows only 5 of the 6 independent displacements (3 rotations + 3 translations) of the 3D space since they lack the rotation around the line going through them. Therefore, removing 6 displacements from a couple of points is geometrically incorrect. However, Hoffmann *et al.*'s function **Distribute** does so when the distance constraint binding the two points in 3D is overloaded with $K = 6$.

In order to distribute the flow in a geometrically correct way, we propose to introduce a fictive constraint R , having $\text{DOF}(R) = K$. This constraint can be linked only to subset of objects O' allowing K independent displacements, i.e. inducing a subGCSP S' having $\text{DOR}(S') \geq K$. K and S' are two parameters of our function **Distribute**.

Function **Overloaded-Network** returns the object-constraint network corresponding to S where the fictive constraint R , set with capacity K , is linked to the objects of S' . The maximum flow computation is achieved by a standard

flow algorithm³ like **FordFulkerson** [Ford and Fulkerson, 1962]. This function returns the set V of objects reachable from the virtual constraint R in the residual graph if the maximum flow cannot distribute the whole overload, an empty set V otherwise. Function **Object-Induced-subGCSP** returns the subGCSP S'' induced by V . S'' verifies $\text{DOF}(S'') < K$ or S'' is empty.

Distribute (S : GCSP; K : integer; S' : GCSP) **returns** S'' : GCSP)

Require: $K > 0$, $S' \subset S$ verifies $\text{DOR}(S') \geq K$

Ensure: $S'' \subset S$ verifies $\text{DOF}(S'') < K$, or S'' is empty

$G \leftarrow \text{Overloaded-Network}(S, K, S')$

$V \leftarrow \text{FordFulkerson}(G)$

$S'' \leftarrow \text{Object-Induced-subGCSP}(V, S)$

Return S''

The two differences between our version of the **Distribute** function and Hoffmann *et al.*'s version have already been mentioned: the use of a dedicated constraint for overflow distribution, which allows to distribute the overflow to any subset of objects; and the adaptation of the overflow to the set of objects on which it is applied, which renders overflow application geometrically correct.

Example: The call to **Distribute**($S, 3, dAB$) (Hoffmann *et al.*'s version) for the GCSP in Fig. 1-a is presented in Fig. 2-a. Since the overflow cannot be fully distributed, the subGCSP AB is returned. This is correct from the flow point of view since $\text{DOF}(AB) = 2$ is less than $K = 3$. However, from the geometric point of view, it is incorrect to interpret this result as an over-rigidity in the GCSP.

For the same subGCSP, our **Distribute** function is called differently: since $\text{DOR}(AB) = 2$, the overflow can be at most 2. Fig. 2-b presents the call to **Distribute**($S, 2, AB$). The overflow can be distributed fully: no subGCSP is returned. Further similar calls would allow to conclude that this GCSP is not over-rigid.

Time Complexity: The complexity of our function **Distribute** is dominated by that of function **FordFulkerson**; it is $O(n^2(n + m))$ where n is the number of nodes and m the number of arcs. It is strictly equivalent to the complexity of Hoffmann *et al.*'s version.

Note that if several calls to this function are performed, it could be modified to compute maximum flow in an incremental way, yielding a better complexity.

4.3 Algorithms For Rigidity Detection

Based on the **Distribute** function, several algorithms can be designed to tackle the major problems related to the rigidity concept. [Hoffmann *et al.*, 1997] have proposed the **Dense** and **Minimal_Dense** algorithms to identify a well

²Remember that the DOR represents the number of independent displacements admitted by a subGCSP.

³In [Hoffmann *et al.*, 1997], function **Distribute** is specifically designed for binary constraints and flow distribution is merged with network construction and subGCSP identification.

or over-rigid subGCSP and minimize it (using a classical linear minimization process). These algorithms can be reproduced using our function `Distribute`. This allows us to tackle the same problems in a geometrically correct manner and with a better characterization of rigidity: the extended structural rigidity. We will show on algorithm `Dense` how to introduce our `Distribute` function in existing algorithms.

Algorithm Dense Versus Algorithm Over-Rigid

Schematically, algorithm `Dense` operates by calling the `Distribute` function for each constraint in the GCSP until a non-empty subGCSP is returned. The overload is induced by the dimension of the considered geometric space: it represents the maximum number of independent displacements in this space (3 in 2D, 6 in 3D). `Dense` is supposed to return only over-rigid subGCSPs since returned GCSPs do not admit all the displacements allowed by the considered geometric space.

In fact, `Dense` is incorrect since it may remove more DOFs than the number of displacements admitted by a subGCSP. For instance, two parallel lines admit only 2 displacements in 2D; hence, removing 3 displacements from two parallel lines is geometrically incorrect in 2D.

To obtain a geometrically correct version of algorithm `Dense`, we propose to use the `es_rigidity` instead of the `s_rigidity`, i.e. the DOR is the overload; also, we use our `Distribute` function instead of Hoffmann *et al.*'s one.

This results in a new algorithm, called `Over-Rigid` which performs one call to function `Distribute( $S, \text{DOR}(S'), S'$ )` for each $S' \subset S$ to identify over-`es_rigid` subGCSPs. Indeed, if the call for a given S' returns a non-empty subGCSP S'' , then it verifies $\text{DOF}(S'') < \text{DOR}(S')$, a sufficient condition for being over-`es_rigid` (see Def. 4). Unfortunately, the number of subGCSPs is exponential, which would lead to an exponential number of calls to function `Distribute`. However, we will show that it is sufficient and correct to apply this function only to the *DOR-minimal* subGCSPs (see Def. 6 below), which gives the following algorithm:

Over-Rigid (S : GCSP) returns S'' : GCSP

Ensure: $S'' \subset S$ is over-`es_rigid` or empty

$S'' \leftarrow \text{EmptyGCSP}$

$M \leftarrow \text{DOR-Minimals}(S)$ {builds the set M of all DOR-minimal subGCSPs in S }

while $S'' = \text{EmptyGCSP}$ and $M \neq \emptyset$ **do**

$S' \leftarrow \text{Pop}(M)$

$S'' \leftarrow \text{Distribute}(S, \text{DOR}(S'), S')$

end while

Return S''

Definition 6 DOR-minimal subGCSP

A subGCSP S' in a GCSP S is **DOR-minimal** if it contains no proper subGCSP with the same DOR, i.e. $\forall S'' \subset S', \text{DOR}(S'') < \text{DOR}(S')$.

Example of Over-Rigid application

Consider again the GCSP S presented in figure 1-b. Let $M = \{BC, BDF, CEF, \dots\}$ be the set of DOR-minimal

subGCSPs generated by `DOR-Minimals( $S$ )`. Algorithm `Over-Rigid` then proceeds as follows:

1. First turn, $S' = BC$ and $K = \text{DOR}(BC) = 5$. Fig. 2(c) represents the call to `Distribute( $S, 5, BC$ )`. All the arcs outgoing from the source being saturated, no over-`es_rigid` subGCSP is identified.
2. At this turn, $S' = CEF$ and $K = \text{DOR}(CEF) = 6$. The call to `Distribute( $S, 6, CEF$ )` is represented in figure 2(d). This turn, the arc $S \rightarrow R$ is unsaturated. Since the set of object-nodes traversed during the last search for an augmenting path is $\{A, C, D, E, F\}$, the identified subGCSP is $ACDEF$ which is over-`es_rigid`.

On the same example, algorithm `Dense` would identify each segment (2 points + 1 distance) and each point on the line A as an over-rigid subGCSP, which is false.

Properties of algorithm Over-Rigid

To prove the correctness and completeness of algorithm `Over-Rigid`, we need the following lemmas which establish properties on the DOR concept and on flow distribution:

Lemma 1 Let S be a GCSP and $S' \subset S'' \subset S$ two subGCSPs. Then $\text{DOR}(S') \leq \text{DOR}(S'')$.

Proof: Each unit of DOR in a GCSP represents an independent displacement (translation or rotation). Adding a new object o with some constraints to a subGCSP S' cannot remove the independent displacements already granted to S' since constraints are independent from the global reference system. Thus, $\text{DOR}(S') \leq \text{DOR}(S' \cup \{o\})$. $\square$

Lemma 2 Let $S'' \subset S' \subset S$ be two nested subGCSPs in a GCSP S . If the call to `Distribute( $S, K, S'$ )` returns a non-empty subGCSP S_0 , then a call to `Distribute( $S, K, S''$ )` returns a non-empty subGCSP S_1 .

Proof: Let $G_{S'}$ be the object-constraint network overloaded for S' and $G_{S''}$ the network overloaded for S'' . The only difference between these two networks resides in the fact that there are more arcs of the type $R \rightarrow o$ in $G_{S'}$. Thus, it is more difficult to distribute an overflow in $G_{S''}$ than in $G_{S'}$: if a maximum flow in $G_{S'}$ cannot saturate all the arcs outgoing from the source, a maximum flow in $G_{S''}$ cannot either. $\square$

We will now prove the completeness, correctness and discuss the time complexity of our algorithm.

Correctness of Over-Rigid:

Let S'' be a non-empty subGCSP resulting from `Over-Rigid( $S$ )`. Assuming it has been returned by the call to `Distribute( $S, \text{DOR}(S'), S'$ )`, S'' must verify $\text{DOF}(S'') < \text{DOR}(S')$ since S'' is not empty. Moreover, by design of function `Distribute`, $S' \subset S''$. Since Lemma 1 implies that $\text{DOR}(S') \leq \text{DOR}(S'')$, we can ensure that if S'' is not empty, then $\text{DOF}(S'') < \text{DOR}(S'')$, i.e., S'' is over-`es_rigid`. $\square$

Completeness of Over-Rigid:

Algorithm **Over-Rigid** applies an overload only for each element in the set M of all *DOR-minimal* subGCSPs (computed by **DOR-Minimals**(S)). Lemma 2 ensures that it is sufficient to distribute an overload for each DOR-minimal subGCSP, since any non DOR-minimal subGCSP contains, by definition, DOR-minimal subGCSPs. $\square$

Time Complexity of Over-Rigid:

The complexity of algorithm **Over-Rigid** depends on the number of DOR-minimal subGCSPs. We have proven by enumeration that the number of objects in a DOR-minimal subGCSP is 2 in 2D and 3 in 3D for GCSPs including points, lines and planes constrained by distances, angles, incidences and parallelisms. Thus, for GCSPs in this class, the number of DOR-minimal subGCSPs is $O(n^d)$ where n is the number of objects and d the dimension of the geometric space (2 or 3).

Let us call C_1 the complexity of function **DOR-Minimals**, and C_2 that of function **Distribute**, discussed in the previous section. Then, the worst-case complexity of algorithm **Over-Rigid** is $O(C_1 + n^d * C_2)$.

C_1 is generally the complexity of geometric theorem proving, i.e., it is exponential. However, in some practical classes of GCSPs, like mechanisms or bar frameworks, it is polynomial or even constant. Moreover, heuristic DOR computation can be used when geometric theorem proving is required but not affordable. In these cases, C_1 can be neglected in comparison to C_2 . We end up with $O(n^{d+2} * (n + m))$. In comparison, the complexity of algorithm **Dense** is $O(m * n^2 * (n + m))$. Thus, the overhead to obtain a geometrically correct algorithm is approximately linear in 2D, and quadratic in 3D.

4.4 Other algorithms

Function **Distribute** can be used in a similar way to tackle the major problems related to rigidity: identifying rigid subGCSPs (just by changing the value of the overflow in algorithm **Over-Rigid**), deciding if a GCSP is rigid (by one call to **Over-Rigid** and a DOF count), finding a minimal well- or over-rigid subGCSP (by classical minimization step, as in **Minimal_Dense**). For all these problems, using our new algorithms and the `es_rigidity` instead of the `s_rigidity` leads to geometrically correct and more reliable algorithms.

4.5 Conclusion

The new design of function **Distribute** allows a more general use of this function: the flow distribution is now performed in a geometrically sound manner and allows for checking a better characterization of rigidity. The DOR-minimal concept and its properties have appeared to be the key to obtain a new family of polynomial algorithms for the major problems related to rigidity.

These new algorithms can handle GCSPs in 2D and 3D correctly with respect to the `es_rigidity`. They can handle GCSPs with constraints like parallelism or incidence, which was not possible with Hoffmann *et al.*'s algorithms. Indeed, these constraints introduce geometric properties in GCSPs, leading

to subGCSPs with a DOR different from the number of independent displacements in d -space. This kind of constraints are ubiquitous in practical applications (architecture, CAD, mechanisms) and our new algorithms open a way for reliable industrial use in these domains.

References

- [Bouma *et al.*, 1995] W. Bouma, I. Fudos, C.M. Hoffmann, J. Cai, and R. Paige. Geometric constraint solver. *Computer Aided Design*, 27(6):487–501, 1995.
- [Dufourd *et al.*, 1998] J.-F. Dufourd, P. Mathis, and P. Schreck. Geometric construction by assembling solved subfigures. *Artificial Intelligence*, 99(1):73–119, 1998.
- [Ford and Fulkerson, 1962] L.R. Ford and D.R. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.
- [Hoffmann *et al.*, 1997] C.M. Hoffmann, A. Lomonosov, and M. Sitharam. Finding solvable subsets of constraint graphs. In *Principles and Practice of Constraint Programming CP'97*, pages 463–477, 1997.
- [Hoffmann *et al.*, 2000] C.M. Hoffmann, A. Lomonosov, and M. Sitharam. Decomposition plans for geometric constraint systems. In *Proc. J. Symbolic Computation 2000*, 2000.
- [Jermann *et al.*, 2000] C. Jermann, G. Trombettoni, B. Neveu, and M. Rueher. A constraint programming approach for solving rigid geometric systems. In *Principles and Practice of Constraint Programming, CP 2000*, volume 1894 of *LNCS*, pages 233–248, 2000.
- [Jermann *et al.*, 2002] C. Jermann, B. Neveu, and G. Trombettoni. On the structural rigidity for gcsp. In *Proceedings of the 4th International Workshop on Automated Deduction in Geometry*, 2002.
- [Jermann, 2002] C. Jermann. *Résolution de contraintes géométriques par rigidification récursive et propagation d'intervalles*. Thèse de doctorat, Université de Nice - Sophia Antipolis, 2002.
- [Kramer, 1992] G. Kramer. *Solving Geometric Constraint Systems*. MIT Press, 1992.
- [Laman, 1970] G. Laman. On graphs and rigidity of plane skeletal structures. *J. Eng. Math.*, 4:331–340, 1970.
- [Lamure and Michelucci, 1998] H. Lamure and D. Michelucci. Qualitative study of geometric constraints. In B. Bruderlin and D. Roller, editors, *Geometric Constraint Solving and Applications*, pages 234–258. Springer, 1998.
- [Sitharam, 2000] M. Sitharam. Personal communication on the minimal dense algorithm. University of Florida at Gainesville, 2000.
- [Whiteley, 1987] W. Whiteley. Applications of the geometry of rigid structures. In Henry Crapo, editor, *Computer Aided Geometric Reasoning*, pages 219–254. INRIA, 1987.

In the quest of the best form of local consistency for Weighted CSP

Javier Larrosa

Department of Software
Universitat Politècnica de Catalunya
Barcelona, Spain
larrosa@lsi.upc.es

Thomas Schiex

Dépt. de Biométrie et Intelligence Artificielle
Institut National de Recherche Agronomique
Toulouse, France
Thomas.Schiex@toulouse.inra.fr

Abstract

The weighted CSP (WCSP) framework is a soft constraint framework with a wide range of applications. In this paper, we consider the problem of *maintaining* local consistency during search for solving WCSP. We first refine the notions of *directional arc consistency* (DAC) and *full directional arc consistency* (FDAC) introduced in [Cooper, 2003] for binary WCSP, define algorithms that enforce these properties and study their complexities. We then consider algorithms that maintain either arc consistency (AC), DAC or FDAC during search. The efficiency of these algorithms is empirically studied. It appears that despite its high theoretical cost, the strongest FDAC property is the best choice.

1 Introduction

It is well known that *arc consistency* (AC) plays a preeminent role in efficient constraint solving. In the last few years, the CSP framework has been augmented with so-called *soft constraints* with which it is possible to express preferences among solutions [Schiex *et al.*, 1995; Bistarelli *et al.*, 1997]. Soft constraint frameworks associate costs to tuples and the goal is to find a complete assignment with minimum combined cost. Costs from different constraints are combined with a domain dependent operator $\oplus$. Extending the notion of AC to soft constraint frameworks has been a challenge in the last few years. From previous works we can conclude that the extension is direct as long as the operator $\oplus$ is idempotent. Then, [Schiex, 2000] proposed an extension of AC which can deal with non-idempotent $\oplus$. This definition has three nice properties: (i) it can be enforced in polynomial time, (ii) the process of enforcing AC reveals infeasible values that can be pruned and (iii) it reduces to existing definitions in the idempotent operator case. [Cooper, 2003] further introduced *directional arc consistency* (DAC) and *full directional arc consistency* for strictly monotonic $\oplus$.

Weighted constraint satisfaction problems (WCSP) is a well known soft-constraint framework with a non-idempotent operator $\oplus$. It provides a very general model with several applications in domains such as *resource allocation* [Cabon *et*

al., 1999], *combinatorial auctions* [Sandholm, 1999], *bioinformatics* and *probabilistic reasoning* [Pearl, 1988]. [Larrosa, 2002] introduced AC*, a refinement of the AC definition for WCSP. This definition provides a stronger yet simple and elegant property to be maintained during search.

In this paper, we take the definitions of DAC and FDAC, strengthen and extend them to binary WCSP, defining the DAC* and FDAC* properties. We then define corresponding enforcing algorithms. As in the classical CSP case, we then consider the problem of maintaining AC*, DAC* and FDAC* during search and empirically compare these algorithms. These algorithms have a wide range of applications and allow a nice integration of hard and soft constraints in a common algorithmic framework.

2 Preliminaries

2.1 CSP

A binary *constraint satisfaction problem* (CSP) is a triple $P = (\mathcal{X}, \mathcal{D}, \mathcal{C})$. $\mathcal{X} = \{1, \dots, n\}$ is a set of variables. Each variable $i \in \mathcal{X}$ 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 i . A tuple t is an assignment to a set of variables. Actually, t is an ordered set of values assigned to the ordered set of variables $\mathcal{X}_t \subseteq \mathcal{X}$ (namely, the k -th element of t is the value assigned to the k -th element of $\mathcal{X}_t$). For a subset B of $\mathcal{X}_t$, the projection of t over B is noted $t \downarrow_B$. $\mathcal{C}$ is a set of unary and binary constraints. A unary constraint C_i is a subset of D_i containing the permitted assignments to variable i . A binary constraint C_{ij} is a set of pairs from $D_i \times D_j$ containing the permitted simultaneous assignments to i and j . The set of variables affected by a constraint is called its *scope*. A tuple t is *consistent* if it satisfies all constraints whose scope is included in $\mathcal{X}_t$. A *solution* is a consistent complete assignment. Finding a solution in a CSP is an NP-complete problem. The task of searching for a solution can be simplified by enforcing arc consistency, which may prune values that cannot participate to a solution.

2.2 Weighted CSPs

Valued CSP (as well as *semi-ring* CSP) extend the CSP framework by associating *costs* to tuples [Schiex *et al.*, 1995; Bistarelli *et al.*, 1997]. In general, costs are specified by means of a so-called *valuation structure* defined as 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.

A valuation structure is *idempotent* if $\forall a \in E, (a \oplus a) = a$. It is *strictly monotonic* if $\forall a, b, c \in E, s.t. (a \succ c) \wedge (b \neq \top)$, we have $(a \oplus b) \succ (c \oplus b)$.

Following [Larrosa, 2002], we define Weighted CSP (WCSP) as a specific subclass of valued CSP that relies on a specific valuation structure $S(k)$.

Definition 1 $S(k)$ is a triple $([0, 1, \dots, k], \oplus, \succeq)$ where,

- $k \in [1, \dots, \infty]$.
- $\oplus$ 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 $k = \top$.

A binary WCSP is a tuple $P = (k, \mathcal{X}, \mathcal{D}, \mathcal{C})$. The valuation structure is $S(k)$. $\mathcal{X}$ and $\mathcal{D}$ are variables and domains, as in standard CSP. $\mathcal{C}$ is a set of cost functions. A binary constraint C_{ij} assigns costs to assignments to variables i and j (namely, $C_{ij} : D_i \times D_j \rightarrow [0, \dots, k]$). A unary constraint C_i assigns costs to assignments to variable i (namely, $C_i : D_i \rightarrow [0, \dots, k]$). We assume the existence of a unary constraint C_i for every variable, and a zero-arity constraint, noted $C_\emptyset$ (if no such constraint is defined, we can always define *dummy* ones $C_i(a) = \perp, \forall a \in D_i$ or $C_\emptyset = \perp$).

When a constraint C assigns cost $\top$ to a tuple t , it means that C forbids t , otherwise t is permitted by C with the corresponding cost. The cost of a tuple t , noted $\mathcal{V}(t)$, is the sum over all applicable costs,

$$\mathcal{V}(t) = \sum_{C_{ij} \in \mathcal{C}, \{i,j\} \subseteq \mathcal{X}_t} C_{ij}(t \downarrow_{\{i,j\}}) \oplus \sum_{C_i \in \mathcal{C}, i \in \mathcal{X}_t} C_i(t \downarrow_{\{i\}}) \oplus C_\emptyset$$

Tuple t is *consistent* if $\mathcal{V}(t) < \top$. The usual task of interest is to find a complete consistent assignment with minimum cost, which is NP-hard. Observe that WCSP with $k = 1$ reduces to classical CSP. In addition, $S(k)$ is idempotent iff $k = 1$ and strictly monotonic iff $k = \infty$. Two WCSP defined over the same variables are said to be *equivalent* if they define the same cost distribution on complete assignments.

For simplicity in our exposition, we assume that every constraint has a different scope. For the moment, we also assume that constraints are implemented as tables and that it is possible to consult and modify entries. This is done without loss of generality (see the proof of Theorem 3).

Example 1 Figure 1.a shows a WCSP with valuation structure $S(4)$ (the set of costs is $[0, \dots, 4]$, with $\perp = 0$ and $\top = 4$). It has three variables $\mathcal{X} = \{x, y, z\}$ with values a, b . There are 2 binary constraints C_{xz}, C_{yz} and two non trivial unary constraints C_x and C_z . Unary costs are depicted inside their domain value. Binary costs are depicted as labelled edges connecting the corresponding pair of values (default cost of 1). Zero costs are not shown. One optimal solution is eg. $x = y = z = b$, with cost 2.

Our definition of WCSP is the same as in [Larrosa, 2002]. It differs from usual definitions [Schiex et al., 1995; Bistarelli et al., 1997] which restrict WCSP to the $k = \infty$ case, a

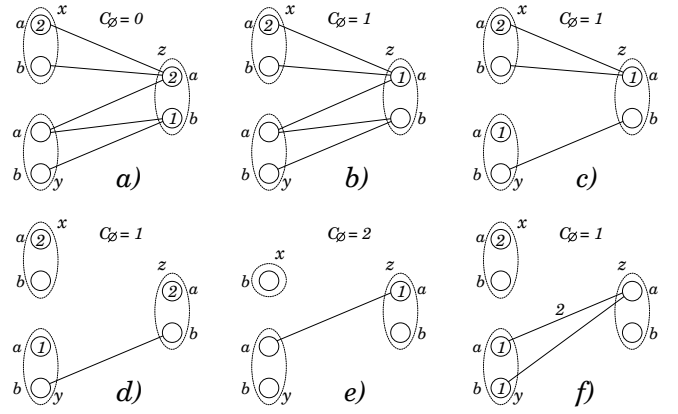


Figure 1: Six equivalent WCSPs (for $k = 4$).

strictly monotonic valuation structure where finite costs cannot lead to deletion. In practice, most *branch and bound*-based solvers maintain an upper bound ub , the maximum acceptable cost so far, and a lower bound lb on the optimal extension of the current assignment. Value pruning occurs as soon as $lb \geq ub$. The WCSP framework makes these two elements explicit: a solver uses the valuation structure $S(ub)$ at every subproblem and $C_\emptyset$ provides the lower bound.

3 Some local consistencies in WCSP

In this Section we define node, arc, directed arc and full directed arc consistencies. For node and arc consistencies, our definitions are equivalent to the NC* and AC* definitions in [Larrosa, 2002]. For DAC, and FDAC, the starred (*) definitions refine the definitions in [Cooper, 2003] to the WCSP case, using node consistency and $C_\emptyset$. In the sequel, we assume that the set of variables $\mathcal{X}$ is totally ordered by $>$.

Definition 2 Let $P = (k, \mathcal{X}, \mathcal{D}, \mathcal{C})$ be a binary WCSP.

- Node consistency: (i, a) is *star node consistent* (NC*) if $C_\emptyset \oplus C_i(a) < \top$. Variable i is NC* if: *i*) all its values are NC* and *ii*) there exists a value $a \in D_i$ such that $C_i(a) = \perp$. Value a is a support for the variable i . P is NC* if every variable is NC*.
- Arc consistency. (i, a) is *arc consistent* (AC) with respect to constraint C_{ij} if there is a value $b \in D_j$ such that $C_{ij}(a, b) = \perp$. Value b is called a support of the value (i, a) . Variable i is AC if all its values are AC wrt. every binary constraint affecting i . P is AC* if every variable is AC and NC*.
- Directional arc consistency. (i, a) is *directional arc consistent* (DAC) wrt. constraint $C_{ij}, j > i$, if there is a value $b \in D_j$ such that $C_{ij}(a, b) \oplus C_j(b) = \perp$. Value b is called a full support of a . Variable i is DAC if all its values are DAC with respect to every $C_{ij}, j > i$. P is DAC* if every variable is DAC and NC*.
- Full directional arc consistency. P is *fully star directional arc consistent* (FDAC*) if it is DAC* and AC*.

In the CSP case (i.e, $k = 1$), being a support for (i, a) is obviously equivalent to being a *full support* for it and both

notions reduce to the classical notion of support. Therefore, AC* and DAC* reduce to their classical definitions in CSP while FDAC* reduces to arc consistency. In WCSP, however, a support is not necessarily a full support. For this reason, DAC* (which requires a full support on one side of each constraint) and AC* (which requires a support on both sides) are incomparable. FDAC* (which requires a support on one side and a full support on the other) cannot be weaker than AC* or DAC*¹. In WCSP, FDAC* can actually be stronger, in the sense that it may provide a better lower bound as the following example shows.

Example 2 *The problem in Figure 1.a is not NC* since z has no support. Problem 1.b is an equivalent NC* problem. It is not DAC* for order xyz since (y, a) has no full support on z . It is not AC* either since eg. (z, a) and (y, a) have no support on x and z respectively. Problem 1.c is an equivalent DAC* problem. It is not AC* since (z, a) has no support on x . Problem 1.d is an equivalent AC* problem. It is not DAC* since (y, b) has no full support on z . Problem 1.e is an equivalent FDAC* problem. The 2 optimal solutions are made obvious here.*

There is a strong relation between directional arc consistency and mini-buckets [Dechter, 1997]. It can easily be shown that given a WCSP defined over the valuation structure $S(\infty)$ and a variable ordering, the lower bound induced by mini-buckets involving at most 2 variables is the same as the lower bound induced by $C_\emptyset$ after the problem is made directional arc consistent. However, the mini-bucket computation provides only a lower bound while DAC enforcing provides both a lower bound and a directional arc consistent equivalent problem. All the work done to compute the lower bound is captured in this problem which offers the opportunity to perform incremental updates of the lower bound.

4 Enforcing Arc Consistencies

The previous node and arc consistency properties can be enforced by applying basic operations until the corresponding property is satisfied: pruning node-inconsistent values, forcing supports to variables (NC*), forcing (full) support to node-consistent values (AC). As pointed out in [Schiex, 2000; Larrosa, 2002], value (resp. variable) supports can be forced by sending costs from binary (resp. unary) constraints to unary constraints (resp. $C_\emptyset$). Full support can be forced by first sending costs from a unary constraint C_j to C_{ij} and then sending the cost from C_{ij} to C_i [Cooper, 2003]. Let us review these concepts before introducing basic algorithms.

Let $a, b \in [0, \dots, k]$, be two costs such that $a \geq b$. $a \ominus b$ is the subtraction of b from a , defined as,

$$a \ominus b = \begin{cases} a - b & : a \neq k \\ k & : a = k \end{cases}$$

The projection of α cost units from $C_{ij} \in \mathcal{C}$ over value (i, a) is a flow of α cost units from the binary constraint to the unary cost $C_i(a)$. It is embodied in the Procedure Project(i, a, j, α).

¹The stronger local property that would require a full support on both sides suffers from the fact that most WCSP don't have an equivalent WCSP that satisfies this property.

For the subtraction to be defined, the maximum flow that can be projected is $\min_{b \in D_j} \{C_{ij}(a, b)\}$. The converse extension of β cost units from value (i, a) to $C_{ij} \in \mathcal{C}$ is a reverse flow of β cost units from (i, a) to the binary constraint. It is embodied in the Procedure Extend(i, a, j, β). The maximum flow that can be extended is $C_i(a)$. Unless stated otherwise, we always assume that maximum flows are projected or extended in the rest of the paper. Similar operations can be defined between unary constraints and $C_\emptyset$.

Procedure Project(i, a, j, α)

```
1  $C_i(a) := C_i(a) \oplus \alpha;$ 
   foreach  $b \in D_j$  do  $C_{ij}(a, b) := C_{ij}(a, b) \ominus \alpha;$ 
```

Procedure Extend(i, a, j, β)

```
2  $C_{ij}(a, b) := C_{ij}(a, b) \oplus \beta;$ 
    $C_i(a) := C_i(a) \ominus \beta;$ 
```

Theorem 1 [Schiex, 2000] *Let $P = (k, \mathcal{X}, \mathcal{D}, \mathcal{C})$ be a binary WCSP. Let $\alpha \leq \min_{b \in D_j} \{C_{ij}(a, b)\}$, $\beta \leq C_i(a)$. The projection of α cost unit of $C_{ij} \in \mathcal{C}$ over (i, a) or the converse extension of β cost unit from (i, a) to C_{ij} transform P into an equivalent problem P' .*

Example 3 *Consider the problem in Figure 1.a. To enforce NC* we must force a support for z by projecting C_z onto $C_\emptyset$. The resulting problem 1.b is NC* but not AC*. To enforce AC*, it suffices to force a support for (y, a) and (z, a) : we project C_{yz} over (y, a) by adding 1 to $C_y(a)$ and subtracting 1 from $C_{yz}(a, a)$ and $C_{yz}(a, b)$ and similarly project C_{xz} over (z, a) . We get problem 1.d which is AC* but not FDAC* using the order xyz since (y, b) has no full support on z . To force a full support for (y, b) , we extend 1 cost unit from $C_z(a)$ to C_{yz} by adding 1 to $C_{yz}(a, a)$ and $C_{yz}(b, a)$ and subtracting 1 from $C_z(a)$. We then project C_{yz} to (y, b) which increases $C_y(b)$ to 1. y is now node-inconsistent, we project y on $C_\emptyset$ and get $C_\emptyset = 2$. x is now node-inconsistent, we prune (x, a) and get problem 1.e which is FDAC* according to order xyz .*

For simplicity, the following descriptions assume that no empty domain is produced and that the initial problem is NC*. It also assumes that the problems have no constraints of arity larger than two. [Larrosa, 2002] defined W-AC*2001, an algorithm based on AC2001 [Bessi re and R gin, 2001] to enforce AC*. It is embodied in the AC*() function of Algorithm 1. It requires two data structures $S(i, a, j)$ and $S(i)$ which respectively store the current value support for (i, a) with respect to constraint C_{ij} and the current variable support for i . The algorithm uses three auxiliary functions : Function ProjectUnary(i) projects C_i onto $C_\emptyset$, Function PruneVar(i) prunes node inconsistent values in D_i and returns **true** if the domain is changed. Function FindSupportAC*(i, j) forces a support on C_{ij} for each value in D_i by projecting C_{ij} on C_i . The main procedure AC*() uses a queue Q containing those variables whose domain has been pruned: adjacent variables may have unsupported values in their domains and new supports must be sought. Q should be initialized with all variables because every variable must find an initial support on every constraint. Ignore for the moment the boolean returned by FindSupportAC* and the use of R . [Larrosa, 2002]

showed that AC^* is time $O(n^2d^3)$ and space $O(ed)$ on general WCSP.

```

Procedure ProjectUnary( $i$ )
   $S(i) := \operatorname{argmin}_{a \in D_i} \{C_i(a)\};$ 
   $\alpha := C_i(S(i));$ 
   $C_\emptyset := C_\emptyset \oplus \alpha;$ 
  foreach  $a \in D_i$  do  $C_i(a) := C_i(a) \ominus \alpha;$ 

Function FindSupportAC*( $i, j$ ) : boolean
   $flag := \text{false};$ 
  foreach  $a \in D_i$  s.t.  $S(i, a, j) \notin D_j$  do
     $S(i, a, j) := \operatorname{argmin}_{b \in D_j} \{C_{ij}(a, b)\};$ 
     $\alpha := C_{ij}(a, S(i, a, j));$ 
    if  $(C_i(a) = \perp) \wedge (\alpha > \perp)$  then  $flag := \text{true};$ 
    Project( $i, a, j, \alpha$ );
  ProjectUnary( $i$ );
  return  $flag;$ 

Function PruneVar( $i$ ) : boolean
   $change := \text{false};$ 
  foreach  $a \in D_i$  s.t.  $(C_i(a) \oplus C_\emptyset = \top)$  do
     $D_i := D_i - \{a\};$ 
     $change := \text{true};$ 
  return  $change;$ 

Procedure AC*( $Q$ )
  while  $(Q \neq \emptyset)$  do
     $j := \operatorname{pop}(Q);$ 
    for  $C_{ij} \in \mathcal{C}$  do
      if FindSupportAC*( $i, j$ ) then  $R := R \cup \{i\};$ 
    foreach  $i \in \mathcal{X}$  do
      if PruneVar( $i$ ) then  $Q := Q \cup \{i\};$ 

```

Algorithm 1: Enforcing AC^* , initially $Q = \mathcal{X}$

[Cooper, 2003] introduced non incremental algorithms for enforcing DAC and FDAC on *strictly monotonic* valuation structures. These algorithms are inadequate for maintaining DAC^* or $FDAC^*$ in a WCSP branch and bound algorithm that relies on a non strictly monotonic valuation structure $S(k), k \neq +\infty$ as soon as a feasible solution is found.

The new basic operation needed to enforce $\{F\}DAC^*$ consists in forcing *full supports* for the values of a variable i on one side of a constraint C_{ij} . As shown in the example, this can be done by extending unary costs from C_j to C_{ij} and then projecting C_{ij} onto variable C_i . However, extending all unary costs may destroy supports for j on C_{ij} . Consider the AC^* Problem 1.d. If we extend 2 cost units from (z, a) to C_{yz} instead of 1 as in the example and then project on C_y , we get Problem 1.f where (z, a) has lost all supports on y . In order to smoothly integrate DAC^* and AC^* enforcing to obtain $FDAC^*$ enforcing, we must obtain full supports for variable i on C_{ij} while preserving *supports* for all values of j on C_{ij} . This is obtained by extending the minimum cost of C_j required for the subsequent projection onto C_i . The correctness of our algorithms is based on the following theorem,

Theorem 2 Let j , a variable whose values are AC wrt. C_{ij} . $\forall a \in D_i, b \in D_j$, let $P[a] = \min_{b \in D_j} \{C_{ij}(a, b) \oplus C_j(b)\}$ and $E[b] = \max_{a \in D_i} \{P[a] - C_{ij}(a, b)\}$. Extending $E[b]$ cost units from (j, b) to C_{ij} for all $b \in D_j$ and projecting $P[a]$ cost units from C_{ij} to (i, a) for all $a \in D_i$ yields

an equivalent WCSP s.t. every node consistent value of i is DAC wrt. C_{ij} and s.t. every value (j, b) is supported by $\operatorname{argmax}_{a \in D_i} \{P[a] - C_{ij}(a, b)\}$ if it is node consistent.

Proof: We denote $C_{ij}^0(a, b)$ the original value of $C_{ij}(a, b)$. We first show that $E[b]$ and $P[a]$ are possible flows. We first prove that $0 \leq E[b] \leq C_j(b)$: we have $P[a] - C_{ij}(a, b) = \min_{b' \in D_j} (C_{ij}(a, b') \oplus C_j(b')) - C_{ij}(a, b) \leq (C_{ij}(a, b) \oplus C_j(b)) - C_{ij}(a, b) \leq (C_{ij}(a, b) + C_j(b)) - C_{ij}(a, b) = C_j(b)$ and therefore $E[b] \leq C_j(b)$. Since j is AC, it has a support (i, a) s.t. $C_{ij}^0(a, b) = \perp \leq P[a]$. Therefore $P[a] - C_{ij}(a, b) \geq 0$ and $E[b] \geq \perp$.

After the extension of $E[b]$ cost units, $C_{ij}(a, b)$ will be equal to $C_{ij}^0(a, b) \oplus \max_{a' \in D_i} \{P[a'] - C_{ij}^0(a', b)\} \geq C_{ij}^0(a, b) \oplus (P[a] - C_{ij}^0(a, b))$. Either this is equal to $\top$ and obviously $C_{ij}(a, b) \geq P[a]$ or else $C_{ij}^0(a, b) \oplus (P[a] - C_{ij}^0(a, b)) = P[a]$ and again $C_{ij}(a, b) \geq P[a]$.

Since $P[a] = \min_{b \in D_j} \{C_{ij}^0(a, b) \oplus C_j(b)\}$ the value b for which this minimum is reached will either be a full support for (i, a) if $P[a] \neq \top$ or (i, a) will be deleted.

On the other side, consider value (j, b) and $a = \operatorname{argmax} (P[a] - C_{ij}^0(a, b))$. After extension and projection, either $C_{ij}(a, b) = C_{ij}^0(a, b) \oplus (P[a] - C_{ij}^0(a, b)) \ominus P[a] \leq P[a] \ominus P[a]$ and either $P[a] < \top$ and a is a support of (j, b) or $P[a] = \top$ and a is node inconsistent. $\square$

Based on this theorem, Function FindFullSupportAC*(i, j) forces full supports for all the values of i on C_{ij} while taking care of supports for values in D_j . It returns **true** whenever the cost of a value (i, a) has been increased from $\perp$. The Procedure DAC^* has been designed to be used alone to enforce DAC^* or in conjunction with AC^* to enforce $FDAC^*$. Therefore, whenever a value is pruned, DAC^* inserts its variable in Q to inform AC^* of the deletion. DAC^* further uses a *priority queue* R that contains those variables such that a unary cost has been increased from $\perp$: in this case, some values in lower variables may have lost full support and new supports need to be found. The main loop iterates while R is not empty. At each iteration, the highest variable j is fetched from R . Node inconsistent values (due to unary cost and lower bound increments) are removed using PruneVar() and pruned variables are inserted in Q . Then new full supports are sought for every lower variable connected to j . Finally, all variables are processed to enforce NC^* which can be lost during the process, due to lower bound increments. Pruned variables are inserted in Q . $FDAC^*$ simply enforces AC^* and DAC^* simultaneously: the enforcement of AC^* empties Q but may add variables to R , and the enforcement of DAC^* empties R but may add variables to Q . $FDAC^*$ is achieved when both R and Q are simultaneously empty. Correction of both algorithms follows from theorem 2.

Theorem 3 The complexity of DAC^* is time $O(ed^2)$ and space $O(ed)$. n, e and d are the number of variables, constraints and largest domain size respectively.

Proof: FindFullSupportAC*(i, j) and PruneVar(i) have complexities $O(d^2)$ and $O(d)$ respectively. The only way a variable j may enter the queue R is because some null unary cost $C_j(b)$ has been increased in FindFullSupportAC*. R being a

```

Function FindFullSupportAC*( $i, j$ ) : boolean
   $flag := \text{false};$ 
  foreach  $a \in D_i$  s.t.  $C_{ij}(a, S(i, a, j)) \oplus C_j(S(i, a, j)) > \perp$  do
     $S(i, a, j) := \text{argmin}_{b \in D_j} \{C_{ij}(a, b) \oplus C_j(b)\};$ 
     $P[a] := C_{ij}(a, S(i, a, j)) \oplus C_j(S(i, a, j));$ 
    if  $(P[a] > \perp) \wedge (C_i(a) = \perp)$  then  $flag := \text{true};$ 
  foreach  $b \in D_j$  do
     $S(j, b, i) := \text{argmax}_{a \in D_i} \{P[a] - C_{ij}(a, b)\};$ 
     $E[b] := P[S(j, b, i)] - C_{ij}(a, b);$ 
  foreach  $b \in D_j$  do  $\text{Extend}(j, b, i, E[b]);$ 
  foreach  $a \in D_i$  do  $\text{Project}(i, a, j, P[a]);$ 
   $\text{ProjectUnary}(i);$ 
  return  $flag;$ 

Procedure DAC*()
  while  $(R \neq \emptyset)$  do
     $j := \text{pop}(R);$ 
    if  $\text{PruneVar}(j)$  then  $Q := Q \cup \{j\};$ 
    foreach  $C_{ij} \in \mathcal{C}$  s.t.  $i < j$  do
      if  $\text{FindFullSupportAC}^*(i, j)$  then  $R := R \cup \{i\};$ 
    foreach  $i \in \mathcal{X}$  do
      if  $\text{PruneVar}(i)$  then  $Q := Q \cup \{i\};$ 

Procedure FDAC*()
  while  $(Q \neq \emptyset) \vee (R \neq \emptyset)$  do
     $\text{AC}^*();$ 
     $\text{DAC}^*();$ 

```

Algorithm 2: DAC* and FDAC*. Initially, $Q = R = \mathcal{X}$.

priority queue, when a variable j is extracted from R , all the variables before j in R have already been processed. Since $\text{FindFullSupportAC}^*$ can only increase non zero unary costs of variables strictly lower than j , j will never be reintroduced in R and therefore each variable j is added to the queue R at most once. The queue is implemented as an array of booleans and a pointer to the highest *true* element. Adding new elements to R means updating the pointer, the *pop* operation consists on returning the value of the pointer and searching for the new highest *true* element. Clearly, DAC^* only traverses the array once. Thus, the complexity of the *while* loop is in $O(ed^2)$. Since the complexity of the second *foreach* loop is in $O(nd)$, the global complexity is in $O(ed^2)$.

As it is, DAC^* is space $O(ed^2)$ since it modifies binary constraints. As [Larrosa, 2002], we note that when a binary constraint C_{ij} is modified (line 1 and 2 in Project and Extend), it is by addition or subtraction of costs that depend either on i or j . It is therefore possible to record only row and column changes, the current value of $C_{ij}(a, b)$ being obtained as $C_{ij}^0(a, b) \oplus F(i, j, a) \oplus F(j, i, b)$ where $C_{ij}^0(a, b)$ denotes the original constraint. There is one $F(i, j, a)$ entry per constraint-value pair which is space $O(ed)$. $\square$

Theorem 4 *The complexity of FDAC^* is time $O(end^3)$ and space $O(ed)$.*

Proof: Regarding space, there is no difference with DAC^* and the same proof applies. Regarding time, a variable j enters Q only if a value has been deleted. Therefore, each variable j is added to Q at most $d + 1$ times (once at initialization and then upon value deletion at lines 5, 6 or 4). There-

fore, line 3 of Procedure AC^* is executed at most $2e(d + 1)$ times and line 4 at most nd times. Globally, line 7 of Algorithm 2 will therefore use $O(n^2d^2 + ed^3)$ elementary operations. For the same reason, line 8 is executed at most $O(nd)$ times. Since DAC^* is in $O(ed^2)$, this can generate $O(end^3)$ elementary operations. Globally, the algorithm is time $O(n^2d^2 + ed^3 + end^3) = O(end^3)$. $\square$

A consequence of these complexity results is that all algorithms terminate (even in the $S(\infty)$ structure).

5 Experimental results

In this Section we perform an empirical evaluation of the effect of maintaining various forms of arc consistency during search. We consider a depth-first search maintaining either NC^* , AC^* , DAC^* or FDAC^* which yields the algorithms MNC^* , MAC^* , MDAC^* and MFDAC^* . For comparison, we include results obtained with PFC-RDAC [Larrosa *et al.*, 1999], which is normally considered as a reference algorithm.

For variable selection we use the *dom/deg* heuristic which for each variable computes the ratio of the domain-size divided by the future degree (i.e., degree considering future variables only) and selects the variable with the smallest value. For value selection we consider values in increasing order of unary cost C_i . The variable ordering used for directional arc consistencies is lexicographic.

We consider the Max-CSP problem, where the goal is to find a complete assignment with a maximum number of satisfied constraints in an overconstrained CSP. It can easily be formulated as a WCSP. We experiment with binary random problems using the well-known four-parameters model [Smith, 1994]. A random CSP class is defined by $\langle n, d, e, t \rangle$ where n is the number of variables, d is the domain size, e is the number of binary constraints (i.e, graph connectivity), and t the number of forbidden tuples in each constraint (i.e, tightness). Pairs of constrained variables and their forbidden tuples are randomly selected using a uniform distribution. Samples have 50 instances and we report average values. The experiments were performed on a 800 MHz Pentium III computer.

For fixed values of n , d and e and increasing tightness t , most problems are solved almost instantly until the cross-over point is reached. Then, problems become overconstrained and much harder to solve. We denote t^o the lowest tightness where every instance in our sample is overconstrained. Based on this, we define different categories of problems:

- For graph density, we define two problem types: *sparse* (S) with $e = 2.5n$ and , *dense* (D) with $e = \frac{n(n-1)}{8}$.
- For tightness, we define two problem types: *loose* (L) with $t = t^o$, and *tight* (T) with $t = d^2 - 0.25t^o$.

Combining the different types, we obtain 4 different classes, each being denoted by a pair of characters (SL, ST, DL and DT). In each class, the domain size is set to 10 and the number of variables n is used as a varying parameter. Figure 2 shows the average cpu time used with SL, ST, DL and DT from left to right. In each plot, the five algorithms are listed in increasing order of efficiency, from top to bottom. In all cases, the search effort seems to grow exponentially with n .

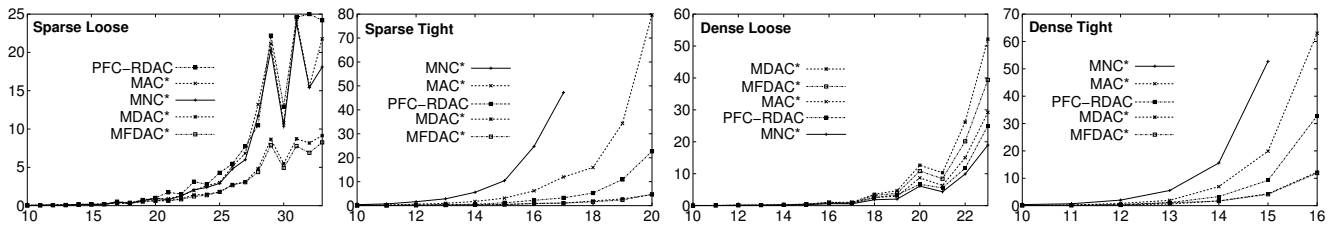


Figure 2: Cpu-time in seconds for an increasing number of variable on our 4 classes of problems. In each case, the 4 algorithms are listed in increasing order of efficiency from top to bottom.

For all classes except the DL class, MFDAC* is the most efficient algorithm, with only minor differences with MDAC* (sometimes they are so closed that the two lines can hardly be distinguished). The best performance of MFDAC* is obtained in the ST problems, where it is up to 5 times faster than PFC-RDAC, 20 times faster than MAC* and 50 times faster than MNC*. For the DL class, however, MNC* is the most efficient algorithm, followed by PFC-RDAC, MAC*, MFDAC* and MDAC*. The differences between the algorithms are however more limited than in previous classes (MNC* is twice faster than MFDAC*).

The ability of directional arc consistency to collect costs along the constraints in order to bring them together in the same variable allows to build stronger lower bounds. This is confirmed by the analysis of the number of nodes expanded by each algorithm (not reported here for lack of space) where MDAC* and MFDAC* always expand less nodes than PFC-RDAC, MNC* or MAC*, with a ratio that can reach 300 between the extreme algorithms on eg. ST problems. On the DL problems however, this ratio is much more limited, typically bounded by 4. With loose constraints, the upper bound reaches low values early in the search which allows pruning at high levels of the search tree and makes sophisticated lower bounds less significant.

It is worth to mention at this point that PFC-RDAC heuristically assigns a direction to every constraint in each subproblem and this has a strong influence on the efficiency on random Max-CSP. Similarly, the behavior of AC, DAC and FDAC based algorithms depends on the order in which variables are fetched from Q and R (i.e., on the variable ordering used to define DAC) and on the order in which values are considered for projection. In our current implementation, Q is implemented as a stack, values are considered in lexicographic ordering and the DAC variable ordering is lexicographic. This leaves room for further improvement.

6 Conclusion and Future Work

In this paper we have refined two local consistency properties and adapted them to WCSP. We have developed enforcing algorithms and have studied their complexity.

As in classical CSP, we observe that the choice of the right level of local consistency to maintain during search is important. Despite its theoretical cost, the strongest local consistency we considered (FDAC*) appears to be the best level for solving WCSP. In the future, we want to extend these algorithms to non binary constraints, apply them to other prob-

lems and take into account heuristics for the variable and value ordering used in AC, DAC and FDAC enforcing.

References

- [Bessière and Régin, 2001] C. Bessière and J-C. Régin. Refining the basic constraint propagation algorithm. In *Proc. of the 14th IJCAI*, pages 309–315, 2001.
- [Bistarelli et al., 1997] S. Bistarelli, U. Montanari, and F. Rossi. Semiring based constraint solving and optimization. *Journal of the ACM*, 44(2):201–236, 1997.
- [Cabon et al., 1999] B. Cabon, S. de Givry, L. Lobjois, T. Schiex, and J.P. Warners. Radio link frequency assignment. *Constraints Journal*, 4:79–89, 1999.
- [Cooper, 2003] Martin C. Cooper. Reduction operations in fuzzy or valued constraint satisfaction. *Fuzzy Sets and Systems*, 134(3), 2003.
- [Dechter, 1997] R. Dechter. Mini-buckets: A general scheme for generating approximations in automated reasoning. In *Proc. of IJCAI’97*, Nagoya, Japan, 1997.
- [Larrosa et al., 1999] J. Larrosa, P. Meseguer, and T. Schiex. Maintaining reversible DAC for Max-CSP. *Artificial Intelligence*, 107(1):149–163, 1999.
- [Larrosa, 2002] Javier Larrosa. On arc and node consistency in weighted CSP. In *Proc. AAAI’02*, 2002.
- [Pearl, 1988] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems, Networks of Plausible Inference*. Morgan Kaufmann, Palo Alto, 1988.
- [Sandholm, 1999] T. Sandholm. An algorithm for optimal winner determination in combinatorial auctions. In *Proc. of IJCAI’99*, pages 542–547, 1999.
- [Schiex et al., 1995] T. Schiex, H. Fargier, and G. Verfaille. Valued constraint satisfaction problems: hard and easy problems. In *Proc. of IJCAI’95*, pages 631–637, 1995.
- [Schiex, 2000] T. Schiex. Arc consistency for soft constraints. In *CP’2000*, volume 1894 of *LNCS*, pages 411–424, 2000.
- [Smith, 1994] B. Smith. Phase transition and the mushy region in constraint satisfaction. In *Proc. of the 11st ECAI*, pages 100–104, 1994.

A Fast and Simple Algorithm for Bounds Consistency of the AllDifferent Constraint

Alejandro López-Ortiz¹, Claude-Guy Quimper¹, John Tromp², Peter van Beek¹

¹School of Computer Science
University of Waterloo
Waterloo, Canada

²CWI
P.O. Box 94079, 1090 GB
Amsterdam, Netherlands

Abstract

In constraint programming one models a problem by stating constraints on acceptable solutions. The constraint model is then usually solved by interleaving backtracking search and constraint propagation. Previous studies have demonstrated that designing special purpose constraint propagators for commonly occurring constraints can significantly improve the efficiency of a constraint programming approach. In this paper we present a fast, simple algorithm for bounds consistency propagation of the alldifferent constraint. The algorithm has the same worst case behavior as the previous best algorithm but is much faster in practice. Using a variety of benchmark and random problems, we show that our algorithm outperforms existing bounds consistency algorithms and also outperforms—on problems with an easily identifiable property—state-of-the-art commercial implementations of propagators for stronger forms of local consistency.

1 Introduction

Many interesting problems can be modeled and solved using constraint programming. In this approach one models a problem by stating constraints on acceptable solutions, where a constraint is simply a relation among several unknowns or variables, each taking a value in a given domain. The constraint model is then usually solved by interleaving backtracking search and constraint propagation. In constraint propagation the constraints are used to reduce the domains of the variables by ensuring that the values in their domains are locally consistent with the constraints.

Previous studies have demonstrated that designing special purpose constraint propagators for commonly occurring constraints can significantly improve the efficiency of a constraint programming approach (e.g., [Régin, 1994; Stergiou and Walsh, 1999]). In this paper we study constraint propagators for the alldifferent constraint. An alldifferent constraint over a set of variables states that the variables must be pairwise different. The alldifferent constraint is widely used in practice and because of its importance is offered as a builtin constraint in most, if not all, major commercial and research-based constraint systems.

Several constraint propagation algorithms for the alldifferent constraint have been developed, ranging from weaker to stronger forms of local consistency (see [van Hoes, 2001] for an excellent survey). Régin [1994] gives an $O(n^{2.5})$ algorithm for domain consistency of the alldifferent constraint, where n is the number of variables, that is based on relating alldifferent constraints to matchings. Leconte [1996] gives an $O(n^2)$ algorithm for range consistency, a weaker form of consistency than domain consistency, that is based on identifying Hall intervals. Puget [1998], building upon the work of Leconte [1996], gives an $O(n \log n)$ algorithm for bounds consistency, which is in turn a weaker form of local consistency than range consistency. Mehlhorn and Thiel [2000], building upon the work of Régin, give an algorithm for bounds consistency that is $O(n)$ plus the time needed to sort the bounds of the domains, and thus has the same worst-case behavior as Puget's algorithm in the general case.

In this paper we present a fast and simple algorithm for bounds consistency propagation of the alldifferent constraint. The algorithm has the same worst case behavior as the previous best algorithm but is much faster in practice. Using a variety of benchmark and random problems, we show that our algorithm outperforms existing bounds consistency algorithms and also outperforms—on problems with an easily identifiable property—state-of-the-art commercial implementations of propagators for stronger forms of local consistency.

A longer version of the paper containing proofs and additional experimentation is available [López-Ortiz et al., 2003].

2 Background

A *constraint satisfaction problem* (CSP) consists of a set of n variables, $\{x_1, \dots, x_n\}$; a finite domain $dom(x_i)$ of possible values for each variable x_i and a collection of m constraints, $\{C_1, \dots, C_m\}$. Each constraint C is a constraint over some set of variables, denoted by $vars(C)$, that specifies the allowed combinations of values for the variables in $vars(C)$. Given a constraint C , we use the notation $t \in C$ to denote a tuple t —an assignment of a value to each of the variables in $vars(C)$ —that satisfies the constraint C . We use the notation $t[x]$ to denote the value assigned to variable x by the tuple t . A *solution* to a CSP is an assignment of a value to each variable that satisfies all of the constraints.

We assume in this paper that the domains are totally ordered. The minimum and maximum values in the domain

$\text{dom}(x)$ of a variable x are denoted by $\min(\text{dom}(x))$ and $\max(\text{dom}(x))$, and the interval notation $[a, b]$ is used as a shorthand for the set of values $\{a, a + 1, \dots, b\}$.

CSPs are usually solved by interleaving backtracking search and constraint propagation. During the backtracking search when a variable is assigned a value, constraint propagation ensures that the values in the domains of the unassigned variables are “locally consistent” with the constraints.

Definition 1 (Support) Given a constraint C , a value $a \in \text{dom}(x)$ for a variable $x \in \text{vars}(C)$ has: (i) a *domain support* in C if there exists a $t \in C$ such that $a = t[x]$ and $t[y] \in \text{dom}(y)$, for every $y \in \text{vars}(C)$; and (ii) an *interval support* in C if there exists a $t \in C$ such that $a = t[x]$ and $t[y] \in [\min(\text{dom}(y)), \max(\text{dom}(y))]$, for every $y \in \text{vars}(C)$.

Definition 2 (Local Consistency) A constraint C is: (i) *bounds consistent* if for each $x \in \text{vars}(C)$, each of the values $\min(\text{dom}(x))$ and $\max(\text{dom}(x))$ has an interval support in C ; (ii) *range consistent* if for each $x \in \text{vars}(C)$, each value $a \in \text{dom}(x)$ has an interval support in C ; and (iii) *domain consistent* if for each $x \in \text{vars}(C)$, each value $a \in \text{dom}(x)$ has a domain support in C .

A CSP can be made locally consistent by repeatedly removing unsupported values from the domains of its variables.

Example 1 (Puget [1998]) Consider the CSP with six variables $x_1, \dots, x_6$; with the following domains, $x_1 \in [3, 4]$, $x_2 \in [2, 4]$, $x_3 \in [3, 4]$, $x_4 \in [2, 5]$, $x_5 \in [3, 6]$, and $x_6 \in [1, 6]$; and a single constraint $\text{alldifferent}(x_1, \dots, x_6)$. Enforcing bounds consistency on the constraint reduces the domains of the variables as follows: $x_1 \in [3, 4]$, $x_2 \in [2]$, $x_3 \in [3, 4]$, $x_4 \in [5]$, $x_5 \in [6]$, and $x_6 \in [1]$.

Given an alldifferent constraint over n variables, where each of the variables x_i , $1 \leq i \leq n$, has an interval domain $\text{dom}(x_i) = [\min_i, \max_i]$, our task is to make the constraint bounds consistent. More precisely, we must update the domain of each variable x_i to $\text{dom}'(x_i) = [\min'_i, \max'_i]$, the minimum and maximum value with interval support, or report that no solution exists.

3 Our result

In this section we present our algorithm for bounds consistency propagation of the alldifferent constraint and analyze its worst-case complexity.

3.1 Finding Hall intervals

Following Leconte [1996] and Puget [1998], we analyze the task in terms of *Hall intervals*. An interval I is a Hall interval if its size equals the number of variables whose domain is contained in I . Clearly, any solution must use all the values in I for those variables, making these values unavailable for any other variable. Puget shows that an algorithm for updating lower bounds can also be used to update upper bounds. The lower bound for variable x_i gets updated where $\min'_i \geq b + 1$, whenever a Hall interval $[a, b]$ with $a \leq \min_i \leq b < \max_i$ is found. This condition implies that any Hall interval $[a', b]$

with $a' < a$ causes the same update. Thus, for the purpose of updating lower bounds, it suffices to restrict attention to *left-maximal* Hall intervals: those $[a, b]$ for which a is minimal.

Puget’s algorithm first sorts the variables in increasing order of $\max_i$. We assume for convenience that $\max_i \leq \max_j$, for $i < j$. The algorithm then processes each of the variables in turn, maintaining a set of counters which count how many of the variables processed so far have a minimum bound of at least k . More precisely, after processing x_i , the counter c_k^i denotes the cardinality of the set $\{j \leq i : \min_j \geq k\}$. The algorithm stores the counters in a balanced binary tree, allowing updates in $O(\log n)$ time per variable.

Conceptually, our algorithm is similar to Puget’s. The difference is in the maintenance of the counters. The key observation is that not all counters are relevant.

Define $v_k^i = \max_i + 1 - k - c_k^i$ as the *capacity* of k after processing x_i . Intuitively this corresponds to as of yet unused values in the current interval.

If $v_k^i = 0$ and $k \leq \max_i$ then we have c_k^i equal to the cardinality of interval $[k, \max_i]$, which is thus recognized as a Hall interval. Capacities increase when $\max_{i+1} > \max_i$ and decrease when $\min_{i+1} \geq k$. More precisely, from the definition of capacity we have, $v_k^{i+1} - v_k^i = \max_{i+1} - \max_i - (c_k^{i+1} - c_k^i)$. Note that c_k^{i+1} is the cardinality of the set $\{j \leq i + 1 : \min_j \geq k\} = \{j \leq i : \min_j \geq k\} \cup \{j = i + 1 : \min_j \geq k\} = c_k^i + \delta$, where $\delta = 1$ if $k \leq \min_{i+1}$ and 0 otherwise. Hence,

$$v_k^{i+1} = v_k^i + \max_{i+1} - \max_i - \delta. \quad (1)$$

Lemma 1 (Domination) Let $k < k'$. If $v_k^i \leq v_{k'}^i$, then $v_k^{i'} \leq v_{k'}^{i'}$ for any $i' > i$.

Capacity v_k^i is *dominated* by $v_{k'}^i$ in the sense that the former cannot reach 0 before the latter, and if both reach 0, then the Hall interval starting at k' is not left-maximal. If k is not equal to any $\min_i$, then it is always dominated by the next greater $\min_i$, hence we need only remember capacities for which k equals some $\min_i$. The *critical set* C is the set of such indices of undominated capacities. This set starts out as $C^0 = \bigcup \{\min_i\}$ and becomes smaller over time as variables are processed and capacities become dominated. We denote by C^i the critical set after processing x_i . The next lemma shows that we can effectively test when each particular capacity v_k^i becomes zero or negative.

Lemma 2 (ZeroTest) Let $k, l \in C^i$ be successive critical indices, such that $k \leq \min_i < l$ and let $d = v_k^i - v_l^i$ be their difference in capacity. Suppose that for all $\min_i < m \leq \max_i$, capacity $v_m^i \geq 0$. Then when v_k^i is positive we have $\max_i + 1 - l + d \geq 2$ and when v_k^i is zero or negative we have $v_k^i = \max_i + 1 - l + d$.

Our algorithm maintains the critical set C as a linked list in which each index points back to the preceding one. The update rule (Equation 1) increases all capacities v_k^{i-1} , $k > \min_i$, by a certain amount, namely, $\max_{i+1} - \max_i - \delta$, and all capacities v_k^{i-1} , $k \leq \min_i$, by 1 less than that amount.

This means that differences between adjacent critical capacities remain constant, except in one place: between capacities v_k and v_l of the ZeroTest Lemma, where the difference is reduced by 1. Therefore, testing for a zero or negative capacity need only be done at this v_k . Our linked list data structure is designed to perform this operation efficiently. All dominated indices form forests pointing toward the next critical index. A dummy index at the end, which never becomes dominated (3 larger than the largest max suffices), ensures that every dominated index has a critical one to point to. When a difference in capacity, say between indices $k_1 < k_2$, is reduced from 1 to 0, k_2 becomes dominated. It must then point to the next critical index, say k_3 , which instead of pointing to k_2 must now point to k_1 .

Example 2 Consider the CSP introduced in Example 1. The variables are ordered by non-decreasing max. After placing a final dummy index at 9, we get initial capacities (relative to $\max = 4$) of $v_1^0 = 4, v_2^0 = 3, v_3^0 = 2, v_9^0 = -4$. We represent this with the data structure $1 \xleftarrow{1} 2 \xleftarrow{1} 3 \xleftarrow{6} 9$, which tracks the differences between adjacent critical capacities. Initially, these are the differences between the indices themselves. Processing $x_1 \in [3, 4]$ reduces the difference between v_3 and v_9 from 6 to 5: $1 \xleftarrow{1} 2 \xleftarrow{1} 3 \xleftarrow{5} 9$. Processing $x_2 \in [2, 4]$ reduces the difference between v_2 and v_3 from 1 to 0, causing v_3 to become dominated by v_2 : $1 \xleftarrow{1} 2 \xleftarrow{5} 9$. Processing $x_3 \in [3, 4]$ reduces the difference between v_2 and v_9 from 5 to 4: $1 \xleftarrow{1} 2 \xleftarrow{4} 9$. The zero capacity $v_2 = \max + 1 - 9 + 4 = 0$ signals a Hall interval $[2, \max] = [2, 4]$. Processing $x_4 \in [2, 5]$ increases $\max$ to 5 while reducing the difference between v_2 and v_9 from 4 to 3: $1 \xleftarrow{1} 2 \xleftarrow{3} 9$. The zero capacity $v_2 = \max + 1 - 9 + 3 = 0$ signals a Hall interval $[2, \max] = [2, 5]$. Processing $x_5 \in [3, 6]$ increases $\max$ to 6 while reducing the difference between v_2 and v_9 from 3 to 2: $1 \xleftarrow{1} 2 \xleftarrow{2} 9$. The zero capacity $v_2 = \max + 1 - 9 + 2 = 0$ signals a Hall interval $[2, \max] = [2, 6]$. Finally, processing $x_6 \in [1, 6]$ reduces the difference between v_1 and v_2 from 1 to 0, causing v_2 to become dominated by v_1 : $1 \xleftarrow{2} 9$. The zero capacity $v_1 = \max + 1 - 9 + 2 = 0$ signals a Hall interval $[1, \max] = [1, 6]$.

3.2 Updating bounds

Finding Hall intervals is only part of the solution. We also need to efficiently update the bounds. For this we use another linked list structure, in which indices inside a Hall interval point to the location representing its upper end, while those outside of any Hall interval point left toward the next such index. We store the list of bounds in a sorted array named **bounds**. Intervals are hereafter numbered by their order of occurrence in this array. The linked list is implemented as an array **t** using indices to the **bounds** as pointers. The differences between critical capacities appearing above the arrows in Example 2 are stored in an array **d**. The algorithm shown in Figure 1 solves one half of the problem: updating all lower bounds. Variable **n** holds the number of intervals,

```

for  $i \leftarrow 1$  to  $nb + 1$  do
   $t[i] \leftarrow h[i] \leftarrow i - 1$ 
   $d[i] \leftarrow \text{bounds}[i] - \text{bounds}[i - 1]$ 
for  $i \leftarrow 0$  to  $niv - 1$  do
   $x \leftarrow \text{maxsorted}[i].\text{minrank}$ 
   $y \leftarrow \text{maxsorted}[i].\text{maxrank}$ 
   $z \leftarrow \text{pathmax}(t, x + 1)$ 
   $j \leftarrow t[z]$ 
   $d[z] \leftarrow d[z] - 1$ 
  if  $d[z] = 0$  then
     $t[z] \leftarrow z + 1$ 
     $z \leftarrow \text{pathmax}(t, t[z])$ 
     $t[z] \leftarrow j$ 
   $\text{pathset}(t, x + 1, z, z)$ 
  if  $d[z] < \text{bounds}[z] - \text{bounds}[y]$  then
    L return Failure
  if  $h[x] > x$  then
     $w \leftarrow \text{pathmax}(h, h[x])$ 
     $\text{maxsorted}[i].\text{min} \leftarrow \text{bounds}[w]$ 
     $\text{pathset}(h, x, w, w)$ 
  if  $d[z] = \text{bounds}[z] - \text{bounds}[y]$  then
     $\text{pathset}(h, h[y], j - 1, y)$ 
     $h[y] \leftarrow j - 1$ 
return Success

```

Algorithm 1: Algorithm for updating lower bounds.

while **nb** holds the number of unique bounds. The algorithm uses the following arrays:

- **maxsorted**[0..n-1]: holds intervals sorted by max.
- **bounds**[0..nb+1]: sorted array of min's and max's.
- **t**[0..nb+1]: holds the critical capacity pointers; that is, $t[i]$ points to the predecessor of i in the bounds list.
- **d**[0..nb+1] holds the differences between critical capacities; i.e., the difference of capacities between interval i and its predecessor in **t** viz. $t[i]$.
- **h**[0..nb+1] holds the Hall interval pointers; i.e., if $h[i] < i$ then the half-open interval $[\text{bounds}[h[i]], \text{bounds}[i])$ forms a Hall interval, and otherwise holds a pointer to the Hall interval it belongs to. This Hall interval is represented by a tree, with the root containing the value of its right end.

The algorithm uses two functions for retrieving/updating pointer information, namely: $\text{pathmax}(a, x)$ which follows the chain $x, a[x], a[a[x]], \dots$, until it stops increasing, returning the maximum found and $\text{pathset}(a, x, y, z)$ which sets each of the entries $a[x], a[a[x]], \dots, a[w]$ to z , where w is such that $a[w]$ equals y . The values **minrank** and **maxrank** give the index in array **bounds** of the min and $(\max + 1)$ of an interval.

The algorithm examines each interval in turn, sorted by their upper bounds. It then updates capacities accordingly, followed by path compression operations on the underlying data structures. At each step we test for failure (a negative capacity) or a newly discovered Hall interval (a zero capacity,

which indicates that the width of the interval is equal to the number of variables whose domain falls within that interval).

Example 3 Table 1 shows a trace of the algorithm for updating lower bounds (Figure 1) when applied to the CSP from Examples 1 & 2. Each row represents an iteration where a variable is processed. In the first graph the nodes are the elements of the vector **bounds**. The arrows illustrate the content of the vector **t** and the numbers over them are given by the vector **d**. The nodes of the second graph are also the values found in vector **bounds** but the arrows are given by the vector **h** that keeps track of the Hall intervals.

Initial situation	
$-1 \xleftarrow{2} 1 \xleftarrow{1} 2 \xleftarrow{1} 3 \xleftarrow{2} 5 \xleftarrow{1} 6 \xleftarrow{1} 7 \xleftarrow{2} 9$	
$-1 \leftarrow 1 \leftarrow 2 \leftarrow 3 \leftarrow 5 \leftarrow 6 \leftarrow 7 \leftarrow 9$	
$x_1 \in [3, 4]$	
$-1 \xleftarrow{2} 1 \xleftarrow{1} 2 \xleftarrow{1} 3 \xleftarrow{1} 5 \xleftarrow{1} 6 \xleftarrow{1} 7 \xleftarrow{2} 9$	
$-1 \leftarrow 1 \leftarrow 2 \leftarrow 3 \leftarrow 5 \leftarrow 6 \leftarrow 7 \leftarrow 9$	
$x_2 \in [2, 4]$	
$-1 \xleftarrow{2} 1 \xleftarrow{1} 2 \xleftarrow{1} 5 \xleftarrow{1} 6 \xleftarrow{1} 7 \xleftarrow{2} 9$	
$-1 \leftarrow 1 \leftarrow 2 \leftarrow 3 \leftarrow 5 \leftarrow 6 \leftarrow 7 \leftarrow 9$	
$x_3 \in [3, 4]$ new Hall interval $[2, 4]$	
$-1 \xleftarrow{2} 1 \xleftarrow{1} 2 \xleftarrow{1} 6 \xleftarrow{1} 7 \xleftarrow{2} 9$	
$-1 \leftarrow 1 \leftarrow 2 \leftarrow 3 \leftarrow 5 \leftarrow 6 \leftarrow 7 \leftarrow 9$	
$x_4 \in [2, 5], x'_4 \leftarrow [5, 5]$ Hall: $[2, 5]$	
$-1 \xleftarrow{2} 1 \xleftarrow{1} 2 \xleftarrow{1} 7 \xleftarrow{2} 9$	
$-1 \leftarrow 1 \leftarrow 2 \leftarrow 3 \leftarrow 5 \leftarrow 6 \leftarrow 7 \leftarrow 9$	
$x_5 \in [3, 6], x'_5 \leftarrow [6, 6]$ Hall: $[2, 6]$	
$-1 \xleftarrow{2} 1 \xleftarrow{1} 2 \xleftarrow{2} 9$	
$-1 \leftarrow 1 \leftarrow 2 \leftarrow 3 \leftarrow 5 \leftarrow 6 \leftarrow 7 \leftarrow 9$	
$x_6 \in [1, 6]$ Hall: $[1, 6]$	
$-1 \xleftarrow{2} 1 \xleftarrow{2} 9$	
$-1 \leftarrow 1 \leftarrow 2 \leftarrow 3 \leftarrow 5 \leftarrow 6 \leftarrow 7 \leftarrow 9$	

Table 1: Trace of the example.

3.3 Time complexity

The running time of the algorithm is dominated by the various calls to **pathmax** and **pathset**. Since each chain followed in a **pathmax** call is also followed in a subsequent **pathset** call, we can restrict our analysis to the time spent in the

latter. Consider the right-running chains in array **t**. Lemma 3 shows that all but a logarithmic number of indices see a rise in value as a result of a path compression operation.

Lemma 3 For any increasing sequence $0 < i_1 < i_2 < \dots < i_k < n$ there are at most $\log n$ elements i_j such that $\lfloor \log(i_k - i_j) \rfloor = \lfloor \log(i_{j+1} - i_j) \rfloor$.

This implies that a linear number of path compressions take at most $O(n \log n)$ steps. The situation with array **h** is similar. It follows then that the algorithm runs in time $O(n \log n)$.

The theoretical performance of the algorithm can be improved further by observing that the union operations are always performed over sets whose bounds appear consecutively in a left to right ordering. This is known as the *interval union-find* problem. Gabow and Tarjan [1985] gave a linear time solution in a RAM computer provided that the keys fit in a single word of memory. This is a reasonable assumption in current architectures with 32 or 64 bit words. Using this technique we obtain a linear time algorithm which matches the theoretical performance of Mehlhorn and Thiel's solution. We implemented this algorithm on the Intel x386 architecture using direct assembly code calls from a C++ program. However, in practice, the $O(n \log n)$ solution outperformed both Mehlhorn and Thiel's algorithm and the algorithm using the interval union find data structure (see [López-Ortiz et al., 2003] for additional discussion).

4 Experimental results

We implemented our new bounds consistency algorithm (denoted hereafter as BC) and Mehlhorn and Thiel's bounds consistency algorithm (denoted MT) using the ILOG Solver C++ library, Version 4.2 [ILOG S. A., 1998]¹. The ILOG Solver already provides implementations of Leconte's range consistency algorithm (denoted RC), Régin's domain consistency algorithm (denoted DC), and an algorithm that simply removes the value of an instantiated variable from the domains of the remaining variables (denoted as VC, for value consistency). To compare against Puget's bounds consistency algorithm, we use the runtime results reported by Puget [1998] for RC and our own runtime results for RC as calibration points. We believe this is valid as Puget also uses a similar vintage of ILOG Solver and when we compared, we were careful to use the same constraint models and variable orderings.

We compared the algorithms experimentally on various benchmark and random problems. All the experiments were run on a 300 MHz Pentium II with 228 MB of main memory. Each reported runtime is the average of 10 runs except for random problems where 100 runs were performed.

We first consider a problem introduced by Puget ([1998]; denoted here as Pathological problems) that were "designed to show the worst case behavior of each algorithm". A Pathological problem consists of a single alldifferent constraint over $2n + 1$ variables with $\text{dom}(x_i) = [i - n, 0]$, $0 \leq i \leq n$, and $\text{dom}(x_i) = [0, i - n]$, $n + 1 \leq i \leq 2n$. The problems were solved using the lexicographic variable ordering. Here, our

¹The code discussed in this section is available on request from vanbeek@uwaterloo.ca

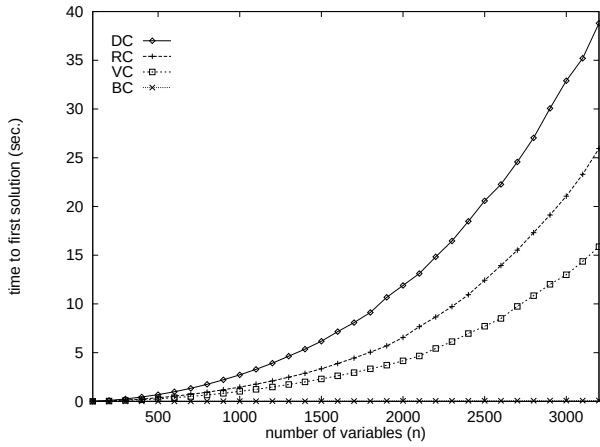


Figure 1: Time (sec.) to first solution for Pathological problems.

BC propagator offers a clear performance improvement over propagators for stronger forms of local consistency (see Figure 1). Comparing against the best previous bounds consistency algorithms, our BC propagator is approx. 2 times faster than MT and, using RC as our calibration point to compare against the experimental results reported by Puget [1998], approx. 5 times faster than Puget’s algorithm.

We next consider the Golomb ruler problem (see [Gent and Walsh, 1999], Problem 6). Following Smith et al. [2000] we modeled the problem using auxiliary variables (their “ternary and all-different model”) and we used the lexicographic variable ordering. This appears to be the same model as Puget [1998] uses in his experiments as the number of fails for each problem and each propagator are the same. Here, our BC propagator is approximately 1.6 times faster than the next fastest propagator used in our experiments (see Table 2) and, again using RC as our calibration point, approximately 1.5 times faster than Puget’s bounds consistency algorithm.

m	VC	RC	DC	MT	BC
8	0.9	0.5	0.6	0.6	0.3
9	9.0	3.8	4.6	4.4	2.3
10	87.3	31.7	39.5	36.5	18.9
11	1773.6	688.1	871.2	841.4	437.6

Table 2: Time (sec.) to optimal solution for Golomb rulers.

We next consider instruction scheduling problems for single-issue processors with arbitrary latencies. Instruction scheduling is one of the most important steps for improving the performance of object code produced by a compiler. Briefly, in the model for these problems there are n variables, one for each instruction to be scheduled, latency constraints of the form $x_i \leq x_j + d$ where d is some small integer value, a single alldifferent constraint over all n variables, and redundant constraints called “distance constraints” In our experiments, we used fifteen representative hard problems that were taken from the SPEC95 floating point, SPEC2000 floating point and MediaBench benchmarks. The minimum do-

n	VC	RC	DC	MT	BC
69		0.02	0.04	0.04	0.02
70		0.02	0.05	0.04	0.02
111	0.08	0.07	0.12	0.11	0.07
211					
214	40.64	0.67	1.20	0.94	0.46
216		0.31	1.08	0.72	0.38
220		0.29	0.93	0.66	0.32
377		0.84	3.94	2.41	0.79
381	0.28	0.50	3.18	1.15	0.39
394		2.66	7.15	2.66	1.65
556					
690		1.91	26.88	3.95	1.61
691		14.47	40.50	6.30	3.14
856		7.72	17.09	10.86	5.48
1006		10.35	87.23	15.90	5.95

Table 3: Time (sec.) to optimal solution for instruction scheduling problems. A blank entry means the problem was not solved within a 10 minute time bound.

main size variable ordering heuristic was used in the search (see Table 3). On these problem too, our BC propagator offers a clear performance improvement over the other propagators.

To systematically study the scaling behavior of the algorithms, we next consider random problems. The problems consisted of a single alldifferent constraint over n variables and each variable x_i had its initial domain set to $[a, b]$, where a and b , $a \leq b$, were chosen uniformly at random from $[1, n]$. The problems were solved using the lexicographic variable ordering. In these “pure” problems nearly all of the run-time is due to the alldifferent propagators, and one can clearly see the quadratic behavior of the RC and DC propagators and the nearly linear incremental behavior of the BC propagator (see Figure 2). On these problems, VC (not shown) could not solve even the smallest problems ($n = 100$) within a 10 minute time bound and MT (also not shown) was 2.5 – 3 times slower than our BC propagator.

Having demonstrated the practicality of our algorithm, we next study the limits of its applicability. Schulte and Stuckey [2001] investigate cases where it can be proven a priori that maintaining bounds consistency during the search, rather than a stronger form of local consistency such as domain consistency, does not increase the size of the search space. The Golomb ruler problem is one such example. In general, of course, this is not the case and using bounds consistency can exponentially increase the search space.

To systematically study the range of applicability of the algorithms, we next consider random problems with holes in the domains of the variables. The problems consisted of a single alldifferent constraint over n variables. The domain of each variable was set in two steps. First, the initial domain of the variable was set to $[a, b]$, where a and b , $a \leq b$, were chosen uniformly at random from $[1, n]$. Second, each of the values $a + 1, \dots, b - 1$ is removed from the domain with some given probability p . The resulting problems were then solved using both the lexicographic and the minimum domain size variable ordering heuristics. These problems are trivial for

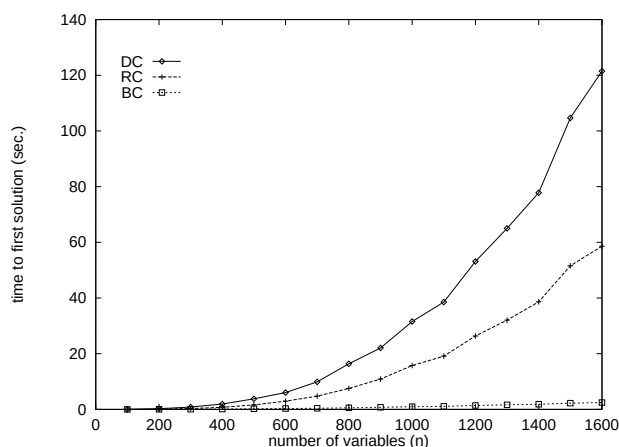


Figure 2: Time (sec.) to first solution or to detect inconsistency for random problems.

domain consistency, but not so for bounds and range consistency. We recorded the percentage that were not solved by BC and RC within a fixed time bound (see Figure 3). If there are no holes in the domains of the variables, then bounds consistency is equivalent to range and domain consistency. As the number of holes increases, the performance of bounds and range consistency decreases and they become less appropriate choices. The range of applicability of BC can be extended somewhat following a suggestion by Puget [1998] of combining bounds consistency with value consistency (denoted as BC+ and MT+). On these problems, BC, BC+, and RC are theoretically equivalent when using the lexicographic variable ordering and BC+ and RC are experimentally equivalent when using minimum domain (see Figure 3).

We also performed experiments on n -queens, quasigroup existence, and sport league scheduling problems. Interestingly, in these experiments, RC was never the propagator of choice. On problems where holes arise in the domains, DC was the best choice (except for on n -queens problems, where VC was considerably faster), and on problems where holes do not arise in the domains, BC was the clear best choice. Clearly, whether the domains have holes in them is a property that is easily identified and tracked during the search. Thus, the best choice of propagator could be automatically selected, rather than left to the constraint modeler to specify as is currently the case.

5 Conclusions

We presented an improved bounds consistency constraint propagation algorithm for the important alldifferent constraint. Using a variety of benchmark and random problems, we showed that our algorithm significantly outperforms the previous best bounds consistency algorithms for this constraint and can also significantly outperform propagators for stronger forms of local consistency.

Acknowledgements

We thank Kent Wilken for providing the instruction scheduling problems used in our experiments.

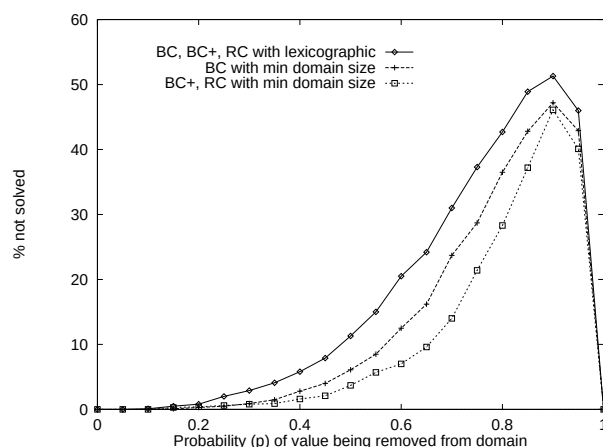


Figure 3: Percentage not solved within a cutoff of 5 seconds for problems with 100 variables. The cutoff was chosen to be the value that was at least two orders of magnitude slower than DC, the fastest propagator on these problems.

References

- [Gabow and Tarjan, 1985] H. N. Gabow and R. E. Tarjan. A linear-time algorithm for a special case of disjoint set union. *JCSS*, 30(2):209–221, 1985.
- [Gent and Walsh, 1999] I. P. Gent and T. Walsh. CSPlib: A benchmark library for constraints. In *CP-99*, pp. 480–481.
- [ILOG S. A., 1998] ILOG Solver 4.2 user's manual, 1998.
- [Leconte, 1996] M. Leconte. A bounds-based reduction scheme for constraints of difference. In *Proc. of the Constraint-96 Int'l Workshop on Constraint-Based Reasoning*, pp. 19–28, Key West, Florida, 1996.
- [López-Ortiz et al., 2003] A. López-Ortiz, C. G. Quimper, J. Tromp and P. van Beek. Faster practical algorithms for the all-diff constraint. Technical Report, CS-2003-05, School of Computer Science, University of Waterloo, 2003.
- [Mehlhorn and Thiel, 2000] K. Mehlhorn and S. Thiel. Faster algorithms for bound-consistency of the sortedness and alldifferent constraint. In *CP-2000*, pp. 306–319.
- [Puget, 1998] J.-F. Puget. A fast algorithm for the bound consistency of alldiff constraints. *AAAI-98*, pp. 359–366.
- [Régin, 1994] J.-C. Régin. A filtering algorithm for constraints of difference in CSPs. In *AAAI-94*, pp. 362–367.
- [Schulte and Stuckey, 2001] C. Schulte and P. J. Stuckey. When do bounds and domain propagation lead to the same search space. In *PPDP-2001*, pp. 115–126.
- [Smith et al., 2000] B. M. Smith, K. Stergiou, and T. Walsh. Using auxiliary variables and implied constraints to model non-binary problems. In *AAAI-2000*, pp. 182–187.
- [Stergiou and Walsh, 1999] K. Stergiou and T. Walsh. The difference all-difference makes. In *IJCAI-99*, pp. 414–419.
- [van Hoeve, 2001] W. J. van Hoeve. The alldifferent constraint: A survey. Submitted manuscript. Available from <http://www.cwi.nl/~wjvh/papers/alldiff.pdf>, 2001.

Solving Constraint Optimization Problems in Anytime Contexts

Samir Loudni

Ecole des Mines de Nantes
F-44307 Nantes Cedex 3 – France
samir.loudni@emn.fr

Patrice Boizumault

GREYC, CNRS UMR 6072, Université de Caen
F-14032 Caen Cedex – France
patrice.boizumault@info.unicaen.fr

Abstract

This paper presents a new hybrid method for solving constraint optimization problems in *anytime* contexts. Discrete optimization problems are modelled as *Valued CSP*. Our method (VNS/LDS+CP) combines a Variable Neighborhood Search and Limited Discrepancy Search with Constraint Propagation to efficiently guide the search. Experiments on the CELAR benchmarks demonstrate significant improvements over other competing methods. VNS/LDS+CP has been successfully applied to solve a real-life *anytime* resource allocation problem in computer networks.

1 Introduction

It is today indisputable that Constraint Programming (CP) fulfills industrial needs of optimization **off-line**. Many systems which use this technology are already operational in different fields of activity, such as planning, scheduling and resource allocation. In **on-line** optimization context, problems dynamically change according to the evolution of the external environment, and their resolution must respect strong *temporal constraints* (*anytime* contexts). However, CP is not yet adapted for solving such *on-line* problems : no guarantee is supplied on the execution times, and current CP solvers do not lend themselves to the combination with other search methods which seem better adapted to respect temporal constraints, such as *stochastic search methods*.

Our objective is to conceive *anytime* algorithms based on CP and to guarantee the following main properties : to produce in a very short time solutions of good quality, and more importantly to improve them gradually as computation time increases. To our knowledge, our proposal is one of the first attempt to efficiently address the problem of building *anytime* algorithms with Constraint Programming.

This paper presents a new hybrid method, called VNS/LDS+CP, dedicated to the efficient computation of high quality solutions (possibly suboptimal) in an *anytime* context. Our method combines a Variable Neighborhood Search [Hansen and Mladenovic, 1997] (it performs moves like local search), and a Limited Discrepancy Search [Harvey and Ginsberg, 1995] with Constraint Propagation to efficiently guide the search.

Section 2 describes the general context in which we take place. Section 3 reviews the hybridization of search algorithms and justifies our first choices. Section 4 details our proposal (VNS/LDS+CP). Section 5, gives computational experiments on CELAR benchmarks ; both mechanisms of VNS/LDS+CP are then finely studied. Section 6 shows how VNS/LDS+CP has been successfully applied to solve a difficult network problem. Finally, we conclude and draw a few perspectives.

2 Anytime constraint optimization context

2.1 Anytime algorithms

The term *anytime* algorithm was coined by Dean and Boddy in the mid-1980s in the context of their work on time-dependent planning [Boddy and Dean, 1994]. Contrary to a standard algorithm, where no result is available until its ending, an *anytime* algorithm can be stopped, at any time, to provide a solution of increasing quality over time.

Performance profiles describe the evolution of the solution quality (produced by the algorithm) as a function of the computation time. They provide a more precise description of the performances and of the behavior of an algorithm than the best known solution reported by *usual algorithms*. In the sequel of this paper, we consider *mean performance profiles* : such curves are built empirically by collecting statistics on the performance of the algorithm over many input instances.

There are two kinds of *anytime* algorithms, namely, *interruptible* and *contract* algorithms [Zilberstein, 1996]. A contract algorithm requires that the computing time must be known prior to its activation. It must produce a solution within a contract of time, and if it is interrupted before the end of the contract, there is no guarantee about the quality of the solution. However, for an interruptible algorithm, no information is (a priori) available about the allowed computation time, and a solution may be asked for at any time. The algorithm must always provide a *good* solution, and more importantly, continue to refine it.

In this paper we are interested by the design of interruptible algorithms, since every interruptible algorithm is trivially a contract algorithm, but the converse is not so immediate.

2.2 Valued Constraint Satisfaction Problems

The Valued CSP (VCSP) [Bistarelli *et al.*, 1999; Schiex *et al.*, 1995] framework is an extension of the CSP (*Constraint Satis-*

fraction Problem) framework, which allows *over-constrained* problems or preferences between solutions to be dealt with.

More formally, whereas a CSP is defined as a triplet $(\mathcal{V}, \mathcal{D}, \mathcal{C})$, where $\mathcal{V}$ is a set of *variables*, $\mathcal{D}$ a set of finite *domains* associated to the variables and $\mathcal{C}$ a set of *constraints* between the variables, a VCSP can be defined as a CSP provided with a valuation structure $\mathcal{S}$ and a valuation function φ . The valuation structure $\mathcal{S}$ is a triplet $(E, \succ, \otimes)$, where E is a valuation set, $\succ$ a total order on E , $\top$ and $\perp$ the maximum and the minimum element in E and $\otimes$ a binary operation closed on E . The valuation function φ , defined from $\mathcal{C}$ to E , associates a valuation to each constraint; the valuation of a constraint denotes its importance.

Let $\mathcal{A}$ be an assignment of all the variables and $\mathcal{C}_{unsat}(\mathcal{A})$ be the set of the constraints unsatisfied by $\mathcal{A}$. The valuation of $\mathcal{A}$ is the aggregation of the valuation of all the constraints in $\mathcal{C}_{unsat}(\mathcal{A})$: $\varphi(\mathcal{A}) = \otimes \varphi(c)$ for $c \in \mathcal{C}_{unsat}(\mathcal{A})$.

The objective is to find a complete assignment with minimum valuation. Specific frameworks depend on the retained operator $\otimes$: Classical CSP ($\wedge$); Possibilistic CSP ($\max$); Lexicographic CSP ($\cup$ on multi-sets); Weighted CSP (Σ). As many real-life problems use an additive criterion, we only consider weighted CSP (WCSP) in the sequel of this paper. From an algorithmic point of view, WCSPs are generally the most difficult to solve [Schiex *et al.*, 1995].

3 Hybridizations of algorithms

3.1 Complete search vs local search

Exact (or complete) tree search methods, such as *Branch and Bound*, are able to produce both an optimal solution, and a proof of optimality. But, because of their exponential worst-case behavior, they may be extremely time consuming. Moreover, it has been experimentally observed that, due to their systematic way of exploring the search space, the quality of their intermediate solutions is usually very poor.

Due to their opportunistic way of exploring the search space, approximate (or incomplete) methods, based on *stochastic local search* (as *Simulated Annealing*, *Tabu Search*), can provide good solutions within a reasonable computing time. But, such methods do not guide fast enough the search towards the best neighbor solutions. Indeed, they may waste a lot of time trying to improve the current solution with no success; this is the case when they remain blocked in *local minima* during a prohibitive time. Such situation is not suitable in an *anytime* context, since the quality of solutions has to be improved gradually as fast as possible, as the computing time increases. Moreover, pure local search algorithms generally require a lot of time to adjust their *noise* parameters; their efficiency strongly depends on the value of these parameters, which are generally application-dependent.

3.2 Hybrid algorithms

Hybrid algorithms [Focacci *et al.*,] provide appropriate compromises between both kinds of search. More precisely, they are very efficient by combining the advantages of both *constraint propagation* (complete search) and *opportunistic exploration* of the search space (local search).

Algorithm 1: The general VNS/LDS+CP algorithm

```

VNS( $\pi$  : Problem,  $Size, discrep$  : Integer,  $Str$  : Strategy)
begin
   $size \leftarrow Size$ 
   $s^* \leftarrow InitialAssignment(\pi)$ 
   $s \leftarrow s^*$ 
   $ub \leftarrow Cost(s^*)$ 
  for  $i \leftarrow 1$  to MAXMOVES do
    1    $rel \leftarrow Relax(s, size, Str)$  // relax solution
    2    $s \leftarrow Rebuild(s, rel, discrep)$  // rebuild solution
        if  $Cost(s) < Cost(s^*)$  then
          L  $s^* \leftarrow s$ 
    3    $size \leftarrow ControlNeighborhoodSize(size)$ 
  return  $s^*$ 
end

```

Intertwined hybridizations are the most attractive and relevant hybridizations of algorithms where both complete and local search are closely mingled during computation. The first kind of such hybrid algorithms belongs to the family of local search methods and uses ideas from CP in order to make *large neighborhoods* (LNS [Shaw, 1998]) more tractable. A second kind belongs to the family of exact search and uses some local search principles to improve the partial solutions at each node of the search tree [Prestwich, 2000] or to explore a set of solutions close to the greedy path in a search tree [Caseau *et al.*, 1999].

3.3 Hybrid algorithms for anytime contexts

VNS/LDS+CP is an *Intertwined hybridization* algorithm. From local search, we have retained a *Variable Neighborhood Search* (VNS) which extends the principle of large neighborhoods (LNS) by *dynamically adjusting* the neighborhood size, when the current solution is a *local optimum*. This choice will partially remedy to the weaknesses of pure local search methods. Indeed, the more the neighborhood is potentially large the more there are chances that it contains good solutions and thus improves quickly the current solution. However, as neighborhoods grow larger, finding the best neighbor may require a too expensive computational effort. So, we have selected the LDS partial search combined with constraint propagation (lower bounds computation) to efficiently explore these neighborhoods.

4 The VNS/LDS+CP Method

VNS/LDS+CP is basically a local search method, as depicted in algorithm 1. It differs from classic local search methods by the size of the neighborhoods, which is adjusted dynamically during the search. It starts with an initial complete assignment s^* ; then, at each move, it *relaxes* (or unassigns variables) a large part of the current solution s and then *rebuilds* it (re-assigns variables) by selecting the best neighbor that strictly improves the cost of the current solution. The algorithm ends when the maximum number of local moves (MAXMOVES) is reached.

LDS explores the neighborhood defined by the relaxed part of the solution. It benefits from constraint propagation based on *lower bounds* computation, and on *dynamic heuristics* for variable and value ordering. Moreover, only judicious neighborhoods, related to *conflict variables* (i.e. variables occur-

Algorithm 2: Relaxing and rebuilding a solution

```
Relax ( $s$  : Solution,  $size$  : Integer,  $Str$  : Strategy)
begin
1   $\nu \leftarrow \text{GenerateNeighborhood}(s, Str)$ 
    $rel \leftarrow \emptyset$ 
   while  $|rel| \neq size$  and  $\nu \neq \emptyset$  do
2     $x \leftarrow \text{RandomElement}(\nu)$ 
      $rel \leftarrow rel \cup \{x\}$ 
      $\nu \leftarrow \nu \setminus \{x\}$ 
   return  $rel$ 
end

Rebuild ( $s$  : Solution,  $rel$  : VarSet,  $discrep$  : Integer)
begin
  if  $|rel| = 0$  then
    if  $Cost(s) < ub$  then
       $ub \leftarrow Cost(s)$ 
       $BestSolution \leftarrow s$ 
    else
       $x_i \leftarrow \text{ChooseVariables}(rel)$ 
       $D_{x_i} \leftarrow \text{SortDomain}(x_i)$ 
       $k \leftarrow 0$ 
      for  $j \leftarrow 1$  to  $DomainSize(x_i)$  and  $k \leq discrep$  do
         $s[i] \leftarrow D_{x_i}[j]$ 
         $lb \leftarrow LB(s, x_i)$ 
        if  $lb < ub$  then
           $Rebuild(s, rel \setminus \{x_i\}, discrep - k)$ 
           $k \leftarrow k + 1$ 
      return  $BestSolution$ 
end
```

ing in *violated constraints*), are considered. Such a choice prevents LDS from modifying the value of conflict variables.

4.1 Relaxing a solution

Algorithm 2 describes how to select the variables to be relaxed. The function of line (1) computes the set ν of current variables candidate to relaxation, according to a strategy Str . A basic strategy is to select all the variables that are in conflict. $size$ elements of ν are randomly chosen to constitute rel the set of variables to be relaxed. This choice enables a balance between choosing variables according to a specific strategy and completely at random. Experiments have shown that the introduction of some randomness enables the search to escape quickly from *local minima*.

4.2 Control of the Neighborhood Size

Initially, the neighborhood size ($size$) takes a *minimum value*. To control the neighborhood size, different strategies have been implemented. The best strategy we have found, increases systematically $size$ by one, each time the method does not improve the cost of the current solution.

4.3 Rebuilding a solution

Algorithm 2 defines the function `Rebuild`. The global variables ub and lb record the upper and lower bounds of the problem optimum. $LB(s, x_i)$ computes a lower bound of the sub-problem, limited to the variables after i (i included). $discrep$ sets the maximum number of choices that we can diverge from the heuristic (*discrepancies*).

Limited Discrepancy Search

The idea of LDS [Harvey and Ginsberg, 1995] is to explore heuristically good solutions that might be at a limited distance from *greedy solution*. LDS ensures a more balanced ex-

ploration of the search tree, and speeds up the reconstruction step, thus improving the performance profile of our method. LDS starts from the solution computed by the value heuristic, and successively explores solutions by increasing the number of discrepancies, until the fixed maximal number of discrepancies is reached.

We have used a generalized version of LDS for n-ary trees. The discrepancy is measured according to the rank of the value chosen for every variable in the order given by the heuristic on values. We count a single discrepancy for the *second cheapest* value of one variable, two discrepancies for either the *third cheapest* value of one variable, or the second cheapest value of two variables, and so on. We only perform *one iteration* of LDS, for a fixed discrepancy. This prevents re-visiting leaf nodes.

Constraint Propagation

One of the main strengths of our approach lies in the use of constraint propagation to prune useless sub-trees and to guide the choice of values during the reconstruction, while keeping this step fast enough. At each node of the search tree, *lower bounds* are computed in order to locally exclude all partial solutions which cannot lead to complete assignments of better valuation than the current best solution. To our knowledge, no method based on large neighborhoods (in particular VNS) uses such a mechanism.

To compute lower bounds, we have adapted the following algorithms [Larrosa *et al.*, 1999] for WCSPs : (i) *Partial Forward Checking* and *Directed Arc Consistency* (PFC-DAC), all the constraints are directed, and DAC are computed from a *directed constraint graph* (we have used specific data-structures to relax the condition of static variable ordering on DAC) ; (ii) *Reversible DAC* (PFC-RDAC), constraints directions can change dynamically during search, looking for a good directed graph causing a high lower bound ; (iii) *Maintaining Reversible DAC* (PFC-MRDAC), DAC are maintained during search, propagating the effect of value pruning.

Variables and Values heuristics

Our heuristic for variable ordering first selects the variable having the lowest ratio *domain cardinality divided by future degree*. Constraint propagation, based on Forward Checking + Directed Arc Consistency, allows us to use a *dynamic minimum inconsistency count* value ordering. During search, for each value, so-called *Inconsistency Counts* (ic) and *Directed Arc Consistency counts* (dac) which record the look-ahead effects on future (unassigned) variables, are computed. Values are selected by their increasing $ic + dac$.

5 Experimentations on CELAR Benchmarks

5.1 Instances of CELAR

The CELAR (French *Centre d'Électronique de l'Armement*) has made available a set of 11 instances of the *Radio Link Frequency Assignment Problem* (RLFAP) [Cabon *et al.*, 1999]. Most of them can be naturally cast as binary WCSPs. For our experiments, we have selected instance 6 (which involves 200 variables, having an average of 40 values in their domain, and 1322 constraints), instance 7 (which involves 400 variables,

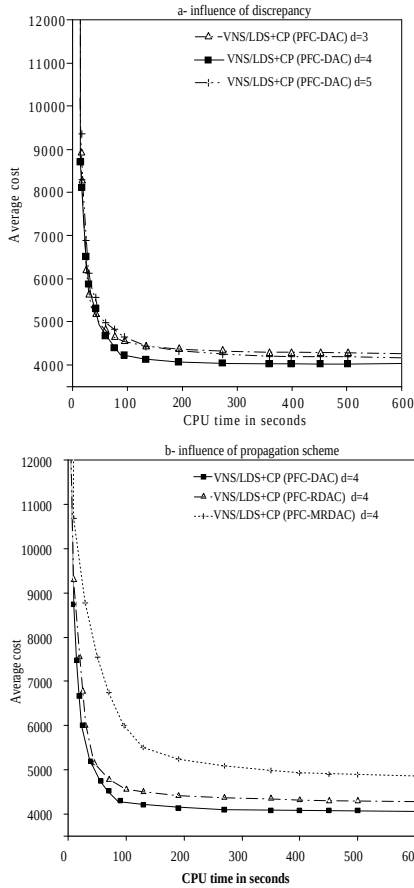


Figure 1: Performance profiles for VNS/LDS+CP, for different setting parameters, on instance 6.

having an average of 40 values in their domain, and 2866 constraints), and instance 6-Sub1 (which involves 28 variables, having an average of 40 values in their domain, and 314 constraints) extracted from instance 6. These instances (6 and 7) are the most difficult ones. Whereas the optimum of instances 6 and 6-Sub1 are known (respectively 3389 and 2669), the optimum of instance 7 is still unknown; the best known upper bound is equal to 343592.

5.2 Experimental Methodology

Our method has 4 parameters : the maximum number of local moves (MAXMOVES), the initial neighborhood size ($size$), the number of discrepancies ($discrep$), and the propagation scheme. For all experiments, MAXMOVES has been set to 150 and $size$ to 4 ; it is the best value we have found among the following set of values : $\{2, 3, 4, 5, 6\}$. We carried out experiments with different values of discrepancies ($discrep \in \{2, 3, 4, 5\}$) and with three different propagation mechanisms : PFC-DAC, PFC-RDAC and PFC-MRDAC.

For each combination of parameter settings, we ran VNS/LDS+CP 50 times on the considered instances. During each run, the best cost of the current solution has been recorded at regular time intervals (10 seconds). All plotted figures are *mean performance profiles* over the 50 runs. The methods have been implemented in choco [Laburthe, 2000].

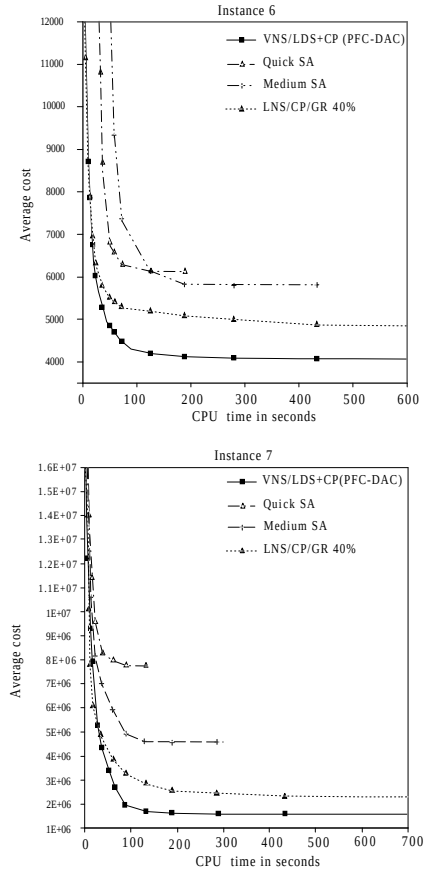


Figure 2: Comparing performance profiles.

5.3 Setting parameters for VNS/LDS+CP

Number of discrepancies

Figure 1 (top) shows the *performance profiles* obtained with PFC-DAC, for different settings of $discrep$ on instance 6. Performance profiles for $discrep = 2$ and 3 are almost the same. We give only the curve associated to $discrep = 3$. Results indicate, as expected, that the number of discrepancies has a great impact on the quality of the computed solutions. LDS with ($discrep = 4$) gives the best results. The poor results for ($discrep > 4$) are probably due to the fact that our value ordering does not make so many mistakes. Indeed, we exploit the propagation scheme to guide the value choices. Thus, few discrepancies are necessary to repair heuristic errors. In contrast, for low discrepancies ($discrep \leq 3$), the value heuristic too closely limits the solutions we can find, and so, it provides poor guidance to good solutions. For the instance 7, $discrep = 4$ is also the best value.

Propagation mechanism

We turn now to choose the propagation scheme which produces significant lower bounds and, as global effect, leads to a better behavior of our method. Figure 1 (bottom) reports the *performance profiles* obtained for the three propagation algorithms, with $discrep = 4$ on instance 6. LDS with PFC-DAC leads to very substantial profits. Because of the overhead of propagation over small neighborhoods these benefits decrease clearly when using stronger propagation (PFC-

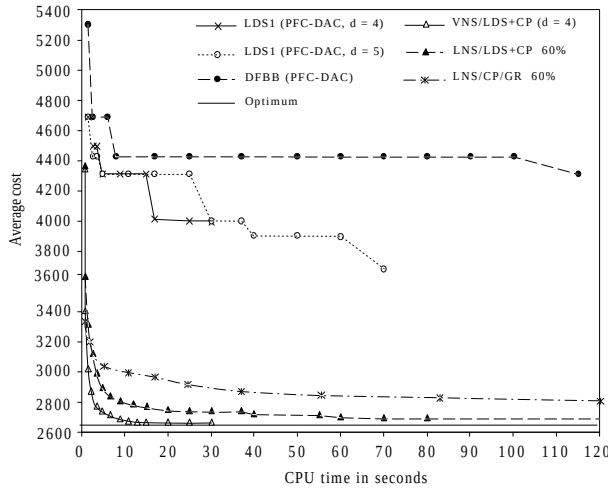


Figure 3: Comparing performance profiles on 6-Sub1.

RDAC and PFC-MRDAC). Indeed, compared to PFC-DAC, the two other propagation schemes perform more work per node to finally obtain results very close to PFC-DAC. So, the more complex and time-consuming propagation schemes do not pay-off for this context.

5.4 Comparisons and discussion

We have compared VNS/LDS+CP ($discrep = 4$ with PFC-DAC) with LNS/CP/GR [Lobjois *et al.*, 2000], another hybrid method which also relies on solving VCSPs in an interruptible context, and also with two standard versions of *Simulated-Annealing* (SA) : *Quick* and *Medium*.

LNS/CP/GR is based on the principle of LNS with neighborhood size being *constant* during all the search. To rebuild the relaxed variables, it uses a *greedy algorithm* combined with constraint propagation. But this propagation (performed variable per variable) is only used for selecting the best values for each variable, which is fundamentally different from our CP based on usual propagation for WCSP. For instances 6 and 7, the percentage of unassigned variables represents 40% of the total number of variables [Lobjois *et al.*, 2000].

Figure 2 compares the performances of the four methods. The *performance profile* of VNS/LDS+CP is always better than those of the two versions of SA. At the beginning, the profile of VNS/LDS+CP is very close to that of LNS/CP/GR. But after few seconds of computation (≤ 20 seconds on instance 6 and ≤ 35 seconds on instance 7), VNS/LDS+CP becomes very effective and always provides solutions of better quality, thus clearly outperforming LNS/CP/GR on both instances. This behavior can be explained by the fact that, at the beginning, the size of the neighborhood is relatively small and consequently the benefits of constraint propagation are poor. But as soon as this size becomes sufficiently large, our propagation mechanism (lower bounds computation) produces more pruning, and becomes very effective. This leads to a better compromise between quality and time for VNS/LDS+CP. This confirms our choice for computing significant lower bounds. Note however, the good *anytime* behavior of LNS/CP/GR compared with those of SA.

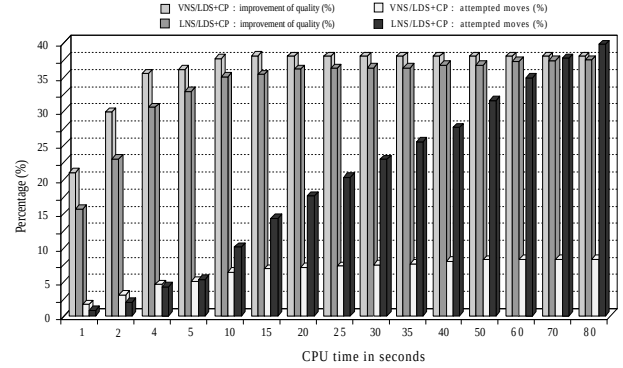


Figure 4: Comparing VNS and LNS on instance 6-Sub1.

5.5 Study of mechanisms VNS and LDS+CP

In order to evaluate the contribution of each component of VNS/LDS+CP we have carried out experiments on instance 6-Sub1, with $MAXMOVES = 150$, and $size = 4$; the two other instances being out of reach from complete searches. Five methods have been compared : the *Depth First Branch and Bound* (DFBB), LDS with only one iteration, two variants of LNS (LNS/CP/GR and LNS/LDS+CP) which mainly differ by their reconstruction mechanism, and VNS/LDS+CP. For the two variants of LNS, the best percentage of unassigned variables represents 60% of the total number of variables.

Once again, $discrep = 4$ constitutes the best parameter setting for VNS/LDS+CP. The behavior of VNS/LDS+CP is quite similar to those observed on instances 6 and 7, which clearly offers better performance profiles than the other competing methods. On this instance, VNS/LDS+CP almost finds the optimum in a very short time at each run. We can draw some remarks :

- pure LDS (or only one iteration of LDS) is completely inefficient in an *anytime* context, even if LDS leads to better performance profiles than DFBB.
- the efficiency of VNS/LDS+CP is certainly partly due to the reconstruction mechanism. Indeed, the comparison between performance profiles of LNS/CP/GR and LNS/LDS+CP show that, with an effective propagation (lower bounds computation) combined with LDS, LNS obtains a gain in quality of 7% after 20 seconds.
- the use of variable size neighborhoods is a key point in the efficiency of VNS/LDS+CP. Indeed, it produces better performance profiles than LNS/LDS+CP. After 2 seconds, VNS obtains a gain in quality of 9% ; this gain reaches a value of 5% after 15 seconds. This amelioration comes from the speed of the exploration of relatively small neighborhoods (in particular during the first moves of VNS). This greatly improves the quality of the computed *upper bound* which will enable a better pruning at the next reconstruction step. This is not the case for LNS, which needs much more time to obtain good upper bounds, due to the important size of the neighborhoods.

To evaluate the improvement speed of the quality of solutions provided by VNS and LNS, figure 4 compares

VNS/LDS+CP and LNS/LDS+CP, in term of the average percentage of improvement, $Q_p(t)$, with respect to the initial cost, C_0 , and in term of the number of attempted moves (in percentage). $Q_p(t) = 100 \cdot (C_0 - C(t))/C_0$, where $C(t)$ is the valuation of the best solution found at each instant t .

As depicted in figure 4, VNS/LDS+CP provides the better improvement of the initial cost C_0 . This difference in quality (in favor of VNS) is very significant during the first instants. After only 4 seconds of computation, VNS improves C_0 by 35%, with a percentage of attempted moves equal to 4% of MAXMOVES. For comparison, LNS needs 15 seconds and practically three times more of local moves ($\approx 14\%$) to reach the same improvement of quality. Moreover, the number of moves attempted by LNS grows larger with time, while for VNS this number remains low and quasi constant.

6 Resource allocation in ATM networks

VNS/LDS+CP has been successfully applied to a real-life *anytime* resource allocation problem. This application, developed for France Telecom R&D, takes place in an ATM (Asynchronous Transfer Mode) network administration context. The problem consists in planning demands of connection over a period of one year. Reservations must be computed within *at most one minute* per demand.

A new demand is accepted if both bandwidth requirements and quality of service are satisfied. If not, we try to *reroute* some already accepted connections in order to find a route for the new demand. If rerouting fails, the demand is rejected. First, a routing heuristic computes shortest paths which minimize capacity violations on the links. From these violations, a sub-area of the network that can be modified by rerouting some already accepted connections is determined and modelled as a VCSP. Then, rerouting is performed with VNS/LDS+CP. Experimental results show that rerouting with VNS/LDS+CP enables to admit an average of 67% of demands that are rejected with a greedy algorithm without rerouting [Loudni *et al.*, 2003].

7 Conclusions and further work

VNS/LDS+CP is a new hybrid method for solving constraint optimization problems (modelled as VCSPs) in an *anytime* context. It has been successfully applied to a real-life *anytime* resource allocation problem in ATM networks (67% of rejected demands are now accepted).

Experiments on the CELAR benchmarks have shown that VNS/LDS+CP provides better performance profiles than the other competing methods. Moreover, our method is robust : for all considered instances, the best parameters setting is always the same. Finally, VNS-LDS+CP is stable : the distribution of the medians $1/4$, $1/2$ and $3/4$ shows that the variation of the evolution of solution quality is negligible within the fifty runs. However, the difference between the best run and those of the medians is relatively distant for instance 7.

From an anytime point of view, in addition to the properties of monotonicity, interruptibility and measurable quality, our performance profiles also verify the important property of *diminishing returns*, which guarantees that the improvement in solution quality is larger at the early stages of computation.

Performance profiles show a decelerating phase leading to a quasi stable plateau. To escape from this plateau, we intend to study two possibilities : to cooperate with other efficient pure local search method [Voudouris and Tsang, 1998] or to re-start our method.

In this paper we have mainly focused on the design of *anytime* algorithms. Works remain to be done concerning techniques for monitoring and control (see [Hansen and Zilberstein, 2001]). We are currently investigating such techniques for meta-level control. The use of *nogoods* would help to obtain more relevant neighborhoods.

References

- [Bistarelli *et al.*, 1999] S. Bistarelli, U. Montanari, F. Rossi, T. Schiex, and G. Verfaillie H. Fargier. Semiring-based cps and valued cps: Frameworks, properties, and comparison. *Constraints*, 4(3):199–240, 1999.
- [Boddy and Dean, 1994] Mark Boddy and Thomas L. Dean. Deliberation scheduling for problem solving in time-constrained environments. *Artificial Intelligence*, 67:245–295, 1994.
- [Cabon *et al.*, 1999] B. Cabon, S. de Givry, L. Lobjois, T. Schiex, and J.P. Warners.W.K. Radio link frequency assignment. *Constraints*, 4(1):79–89, 1999.
- [Caseau *et al.*, 1999] Y. Caseau, F. Laburthe, and G. Silverstein. A meta-heuristic factory for vehicle routing problems (meta-programming for meta-heuristics). In *Proceedings of CP-99*, number 1713 in LNCS, pages 144–158, VA, USA, 1999.
- [Focacci *et al.*,] F. Focacci, F. Laburthe, and A. Lodi. Local search and constraint programming. To appear in F. Glover and G. Kochenberger (Eds.), *HandBook on Metaheuristics*, Kluwer Academic Publishers.
- [Hansen and Mladenovic, 1997] P. Hansen and N. Mladenovic. Variable neighborhood search. *Computers And Operations Research*, 24:1097–1100, 1997.
- [Hansen and Zilberstein, 2001] Eric A. Hansen and Shlomo Zilberstein. Monitoring and control of anytime algorithms: A dynamic programming approach. *Artificial Intelligence*, 126:139–157, 2001.
- [Harvey and Ginsberg, 1995] William D. Harvey and Matthew L. Ginsberg. Limited discrepancy search. In *Proceedings of IJCAI'95*, Montreal, August 1995.
- [Laburthe, 2000] François Laburthe. Choco: implementing a cp kernel. In *CP00 Post Conference Workshop on Techniques for Implementing Constraint programming Systems (TRICS)*, Singapore, September 2000.
- [Larrosa *et al.*, 1999] J. Larrosa, P. Meseguer, and T. Schiex. Maintaining reversible dac for solving max-csp. *Artificial Intelligence*, 107(1):149–163, 1999.
- [Lobjois *et al.*, 2000] L. Lobjois, M. Lemaître, and G. Verfaillie. Large neighbourhood search using constraint propagation and greedy reconstruction for valued csp resolution. In *Proceedings of the ECAI2000 Workshop on Modelling and Solving Problems with Constraints*, 2000.
- [Loudni *et al.*, 2003] Samir Loudni, Patrice Boizumault, and Philippe David. On-line resources allocation for atm networks with rerouting. In *Proceedings of Integration of AI and OR Techniques in Constraints Programming for Combinatorial Problems (CP-AI-OR'03)*, Montréal, Canada, May 2003.
- [Prestwich, 2000] S. Prestwich. A hybrid search architecture applied to hard random 3-sat and low-autocorrelation binary sequences. In *Proceedings of CP 2000*, number 1894 in LNCS, pages 337–352, Singapore, September 2000.
- [Schiex *et al.*, 1995] Thomas Schiex, Hélène Fargier, and Gerard Verfaillie. Valued constraint satisfaction problems: Hard and easy problems. In *Proceedings of IJCAI'95*, Montreal, August 1995.
- [Shaw, 1998] Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *Proceedings of CP-98*, pages 417–431, 1998.
- [Voudouris and Tsang, 1998] C. Voudouris and E. Tsang. Solving the radio link frequency assignment problem using guided local search. In *Proceedings of NATO symposium on Frequency Assignment, Sharing and Conservation in Systems*, Aalborg, Denmark, 1998.
- [Zilberstein, 1996] Shlomo Zilberstein. Using anytime algorithms in intelligent systems. *AI Magazine*, 17(3):73–83, 1996.

Scenario-based Stochastic Constraint Programming

Suresh Manandhar and Armagan Tarim

Department of Computer Science
University of York, England
email: {suresh,at}@cs.york.ac.uk

Toby Walsh

Cork Constraint Computation Centre
University College Cork, Ireland.
email: tw@4c.ucc.ie.

Abstract

To model combinatorial decision problems involving uncertainty and probability, we extend the stochastic constraint programming framework proposed in [Walsh, 2002] along a number of important dimensions (e.g. to multiple chance constraints and to a range of new objectives). We also provide a new (but equivalent) semantics based on scenarios. Using this semantics, we can compile stochastic constraint programs down into conventional (non-stochastic) constraint programs. This allows us to exploit the full power of existing constraint solvers. We have implemented this framework for decision making under uncertainty in stochastic OPL, a language which is based on the OPL constraint modelling language [Hentenryck *et al.*, 1999]. To illustrate the potential of this framework, we model a wide range of problems in areas as diverse as finance, agriculture and production.

1 Introduction

Many decision problems contain uncertainty. Data about events in the past may not be known exactly due to errors in measuring or difficulties in sampling, whilst data about events in the future may simply not be known with certainty. For example, when scheduling power stations, we need to cope with uncertainty in future energy demands. As a second example, nurse rostering in an accident and emergency department requires us to anticipate variability in workload. As a final example, when constructing a balanced bond portfolio, we must deal with uncertainty in the future price of bonds. To deal with such situations, [Walsh, 2002] has proposed an extension of constraint programming, called *stochastic constraint programming*, in which we distinguish between decision variables, which we are free to set, and stochastic (or observed) variables, which follow some probability distribution. This framework combines together some of the best features of traditional constraint satisfaction, stochastic integer programming, and stochastic satisfiability.

In this paper, we extend the expressivity of this framework considerably by adding multiple chance constraints, as well as a range of objective functions like maximizing the downside. We show how such stochastic constraint programs can

be compiled down into conventional (non-stochastic) constraint programs using a scenario-based interpretation. This compilation allows us to use existing constraint solvers without any modification, as well as call upon the power of hybrid solvers which combine constraint solving and integer programming techniques. We also propose a number of techniques to reduce the number of scenarios and to generate robust solutions. We have implemented this framework for decision making under uncertainty in a language called stochastic OPL. This is an extension of the OPL constraint modelling language [Hentenryck *et al.*, 1999]. Finally, we describe a wide range of problems that we have modelled in stochastic OPL that illustrate some of its potential.

2 Stochastic constraint programs

In a one stage stochastic constraint satisfaction problem (stochastic CSP), the decision variables are set before the stochastic variables. The stochastic variables, independent of the decision variables, take values with probabilities given by a probability distribution. This models situations where we act now and observe later. For example, we have to decide now which nurses to have on duty and will only later discover the actual workload. We can easily invert the instantiation order if the application demands, with the stochastic variables set before the decision variables. Constraints are defined (as in traditional constraint satisfaction) by relations of allowed tuples of values. Constraints can, however, be implemented with specialized and efficient algorithms for consistency checking.

We allow for both hard constraints which are always satisfied and “chance constraints” which may only be satisfied in some of the possible worlds. Each chance constraint has a threshold, θ and the constraint must be satisfied in at least a fraction θ of the worlds. A one stage stochastic CSP is satisfiable iff there exists values for the decision variables so that, given random values for the stochastic variables, the hard constraints are always satisfied and the chance constraints are satisfied in at least the given fraction of worlds. Note that [Walsh, 2002] only allowed for one (global) chance constraint so the definition here of stochastic constraint programming is strictly more general.

In a two stage stochastic CSP, there are two sets of decision variables, V_{d1} and V_{d2} , and two sets of stochastic variables, V_{s1} and V_{s2} . The aim is to find values for the variables in V_{d1} ,

so that given random values for V_{s1} , we can find values for V_{d2} , so that given random values for V_{s2} , the hard constraints are always satisfied and the chance constraints are again satisfied in at least the given fraction of worlds. Note that the values chosen for the second set of decision variables V_{d2} are conditioned on both the values chosen for the first set of decision variables V_{d1} and on the random values given to the first set of stochastic variables V_{s1} . This can model situations in which items are produced and can be consumed or put in stock for later consumption. Future production then depends both on previous production (earlier decision variables) and on previous demand (earlier stochastic variables).

An m stage stochastic CSP is defined in an analogous way to one and two stage stochastic CSPs. Note that [Walsh, 2002] insisted that the stochastic variables take values independently of each other. This prevents us representing a number of common situations. For example, if the market goes down in the first quarter, it is probably more likely to go down in the second quarter. A second stage stochastic variable representing the market index is therefore dependent on the first stage stochastic variable representing the market index. There is, however, nothing in the semantics given for stochastic constraint programs nor in the solution methods proposed in [Walsh, 2002] that used this assumption. We therefore allow later stage stochastic variables to take values which are conditioned by the earlier stage stochastic variables.

A stochastic constraint optimization problem (stochastic COP) is a stochastic CSP plus a cost function defined over the decision and stochastic variables. In [Walsh, 2002], the only goal considered was to find a solution that satisfies the stochastic CSP which minimizes or maximizes the expected value of the objective function. We now extend this to a much wider range of goals. For example, we might wish to limit the downside (i.e. maximize the least value of the cost function), or to minimize the spread (i.e. minimize the difference between the least and the largest value of the cost function).

3 Scenario-based semantics

In [Walsh, 2002], a semantics for stochastic constraint programs is given based on policies. A policy is a tree of decisions. Each path in a policy represents a different possible scenario (set of values for the stochastic variables), and the values assigned to decision variables in this scenario. To find satisfying policies, [Walsh, 2002] presents backtracking and forward checking algorithms which explores the implicit AND/OR graph. Stochastic variables give AND nodes as we must find a policy that satisfies all their values, whilst decision variables give OR nodes as we only need find one satisfying value.

An alternative semantics, which suggests an alternative solution method, comes from a scenario-based view [Birge and Louveau, 1997]. A scenario is any possible set of values for the stochastic variables. Thus, a scenario is associated with each path in the policy. Within each scenario, we have a conventional (non-stochastic) constraint program to solve. We simply replace the stochastic variables by the values taken in the scenario, and ensure that the values found for the decision variables are consistent across scenarios. Note that

certain decision variables are shared across scenarios. The first stage decisions are, for example, shared by all scenarios. The great advantage of this approach is that we can use conventional constraint solvers to solve stochastic constraint programs. We do not need to implement specialized solvers. Of course, there is a price to pay as the number of scenarios grows exponentially with the number of stages. However, our results show that a scenario-based approach is feasible for many problems. Indeed, we observe much better performance using this approach on the production planning example introduced in [Walsh, 2002]. In addition, as we discuss later, we have developed a number of techniques like Latin hypercube sampling to reduce the number of scenarios considered.

4 Stochastic OPL

We have implemented this framework on top of the OPL constraint modelling language [Hentenryck *et al.*, 1999]. An OPL model consists of two parts: a set of declarations, followed by an instruction. Declarations define the data types, (input) data and the (decision) variables. An OPL instruction is either to satisfy a set of constraints or to maximize/minimize an objective function subject to a set of constraints. We have extended the declarations to include the declaration of stochastic variables, and the instructions to include chance constraints, and a range of new goals like maximizing the expectation of an objective function.

4.1 Variable declaration

We now declare both decision and stochastic variables. Stochastic variables are set according to a probability distribution using a command of the form:

```
stoch <Type> <Id> <Dist>;
```

Where <Type> is (as with decision variables) a data type (e.g. a range of values, or an enumerated list of values), <Id> is (as with decision variables) the variable name, and <Dist> defines the probability distribution of the stochastic variable(s). Probability distributions include `uniform`, `poisson (lambda)`, and user defined via a list of (not necessarily normalized) values. Other types of distribution can be supported as needed. We insist that stochastic variables are arrays, with the last index describing the stage. Here are some examples:

```
stoch 0..1 market[Years] uniform;
stoch 100..102 demand[Quarter] {1,2,3};
```

In the first, we have a 0/1 variable in each year which takes either value with equal probability. In the last, we have a demand variable for each quarter, which takes the value 100 in 1 out of 6 cases, 101 in 2 out of 6 cases, and 102 in the remaining 3 cases.

4.2 Constraint posting

We can post both hard constraints (as in OPL) and chance constraints. Chance constraints hold in some but not necessarily all scenarios. They are posted using a command of the form:

```
prob(<Constraint>) <ArithOp> <Expr>;
```

Where `<Constraint>` is any OPL constraint, `<ArithOp>` is any of the arithmetically comparison operations (`=`, `<`, `>`, `<=`, or `>=`) and `<Expr>` is any arithmetic expression (it may contain decision variables or may just be a rational or a float in the range 0 to 1). For example, the following command specifies the chance constraint that in each quarter the demand (a stochastic variable) does not exceed the production (a decision variable) plus the stock carried forward in each quarter (this auxiliary is modelled, as in conventional constraint programming, by a decision variable) with 80% probability:

```
forall(i in 1..n)
  prob(demand[i] <=
        production[i]+stock[i])
    >= 0.80;
```

Constraints which are not chance constraints are hard and have to hold in all possible scenarios. For example, the stock carried forwards is computed via the hard constraint:

```
forall(i in 1..n)
  stock[i+1] = stock[i] + production[i]
               - demand[i];
```

4.3 Optimization

Stochastic OPL supports both stochastic constraint satisfaction and optimization problems. We can maximize or minimize the expectation of an objective function. For example, in the book production example of [Walsh, 2002], we can minimize the expected cost of storing surplus books. Each book costs \$1 per quarter to store. This can be specified by the following (partial) model:

```
minimize expected(cost)
subject to
  cost =
    sum(i in 1..n) max(stock[i+1],0);
forall(i in 1..n)
  stock[i+1] = stock[i] + production[i]
               - demand[i];
```

Stochastic OPL also supports a number of other optimization goals. For example:

```
minimize spread(profit)
maximize downside(profit)
minimize upside(cost)
```

The spread is the difference between the value of the objective function in the best and worst scenarios, whilst the downside (upside) is the minimum (maximum) objective function value a possible scenario may take.

5 Compilation of stochastic OPL

These stochastic extensions are compiled down into conventional (non-stochastic) OPL models automatically by exploiting the scenario-based semantics. The compiler is written in Lex and Yacc, with a graphical interface in Visual C++. Compilation involves replacing stochastic variables by their possible values, and decision variables by a ragged array of decision variables, one for each possible scenario. Consider again the chance constraint:

```
prob(
  demand[i] <= production[i]+stock[i])
  >= 0.80;
```

This is compiled into a sum constraint of the form:

```
sum(j in Scenarios) p[j]*
  (demand[i,j] <=
   production[i,j]+stock[i,j])
  >= 0.80;
```

Where `Scenarios` is the set of scenarios, `p[j]` is the probability of scenario `j`, `demand[i,j]` is the demand in scenario `j` and quarter `i`, etc. Note that the bracketing of the inequality reifies the constraint so that it takes the value 1 if satisfied and 0 otherwise.

Hard constraints are also transformed. Consider, for example, the hard constraint:

```
wealth[t] = bonds[t] + stocks[t];
```

This is compiled into a forall constraint of the form:

```
forall(j in Scenarios)
  wealth[t,j] = bonds[t,j] + stocks[t,j];
```

Where `wealth[t,j]` is the wealth at time `t` in scenario `j`, etc. Maximization and minimization instructions are also transformed. Consider, for example, the optimization instruction:

```
maximize expected(wealth[n])
subject to ...
```

This is compiled into an instruction of the form:

```
maximize sum(j in Scenarios)
  p[j]*wealth[n,t]
subject to ...
```

The rest of the stochastic OPL model is transformed in a similar manner.

6 Value of information and stochastic solutions

For stochastic optimization problems, we compute two statistics which quantify the importance of randomness. The value of a stochastic solution (VSS) is the difference in the objective function for the stochastic problem (call it the stochastic solution, SS) and the objective value for the deterministic problem computed by replacing stochastic variables by their expectations (call it the expected value solution, EVS): $VSS = SS - EVS$. This computes the benefit of knowing the distributions of the stochastic variables. Clearly, VSS is non-negative. We also compute the expected value of the wait-and-see solution (WSS). To calculate this, we give the stochastic variables values according to their probability distributions, and then find the best values for the decision variables. The difference between WSS and SS is the expected value of perfect information (EVPI): $EVPI = WSS - SS$. This measures how much more you can expect to win if you have perfect information about the stochastic components of the problem. In other words, EVPI measures the value of knowing the future with certainty. This is therefore the most that should be spent in gathering information about the uncertain world.

7 Scenario reduction

One problem with a scenario-based approach is the large number of scenarios, each of which introduces new decision variables. We have therefore implemented several techniques to reduce the number of scenarios. The simplest is to consider just a single scenario in which stochastic variables take their expected values. This is supported with the command:

```
scenario expected;
```

The user may also be content to consider just the most probable scenarios and ignore rare events. We support this with the command:

```
scenario top <Num>;
```

Another option is to use Monte Carlo sampling. The user can specify the number of scenarios to sample using a command of the form:

```
scenario sample <Num>;
```

The probability distributions of the stochastic variables is used to bias the construction of these scenarios.

We also implemented one of the best sampling methods from experimental design, and one of the best scenario reduction methods from operations research. Latin hypercube sampling [McKay *et al.*, 1979], ensures that a range of values for a variable are sampled. Suppose we want n sample scenarios. We divide the unit interval into n intervals, and sample a value for each stochastic variable whose cumulative probability occurs in each of these interval. We then construct n sample scenarios from these values, enforcing the condition that the samples use each value for each stochastic variable exactly once. More precisely, let $f_i(a)$ be the cumulative probability that X_i takes the value a or less, $\pi_i(j)$ be the j th element of a random permutation π_i of the integers $\{0, \dots, n-1\}$, and r be a random number uniformly drawn from $[0, 1]$. Then, the j th Latin hypercube sample value for the stochastic variable X_i is:

$$f_i^{-1}\left(\frac{\pi_i(j) + r}{n}\right)$$

Finally, we implemented a scenario reduction method used in stochastic programming due to Dupacova, Growe-Kuska and Romisch [Dupacova *et al.*, 2003]. They report power production planning problems on which this method offers 90% accuracy sampling 50% of the scenarios and 50% accuracy sampling just 2% of the scenarios. The method heuristically deletes scenarios to approximate as closely as possible the original scenarios according to a Fortet-Mourier metric on the stochastic parameter space.

8 Some examples

To illustrate the potential of this framework for decision making under uncertainty, we now describe a wide range of problems that we have modelled. In the first problem, we compare a scenario-based approach to the previous tree search methods for solving stochastic constraint satisfaction problems. In the next three problems, we illustrate the effectiveness of the different scenario reduction techniques.

8.1 Production planning

This problem comes from [Walsh, 2002]. The results in Table 1 show that a scenario-based approach offers much better performance on this problem than the forward checking or backtracking tree search algorithms also introduced in this paper. The problem involves planning production over m quarters. In each quarter, we expect to sell between 100 and 105 copies of a book. To keep customers happy, we want to satisfy demand over all m quarters with 80% probability. This problem is modelled by an m stage stochastic CSP. There are m decision variables, x_i representing production in each quarter. There are also m stochastic variables, y_i representing demand in each quarter. To limit stock carried forward, we use a simple heuristic which picks the smallest possible values for the decision variables. An alternative is to convert the problem into an optimization problem with a cost to keep books in store. We do not explore this option here, though it is very easy to implement in stochastic OPL, as we cannot then compare our results with those of the forward checking or backtracking algorithms from [Walsh, 2002].

8.2 Portfolio management

This portfolio management problem of [Birge and Louveaux, 1997] can be modelled as a stochastic COP. Suppose we have $\$P$ to invest in any of I investments and we wish to exceed a wealth of $\$G$ after t investment periods. To calculate the utility, we suppose that exceeding $\$G$ is equivalent to an income of $q\%$ of the excess while not meeting the goal is equivalent to borrowing at a cost $r\%$ of the amount short. This defines a concave utility function for $r > q$. The uncertainty in this problem is the rate of return, which is a random variable, on each investment in each period. The objective is to determine the optimal investment strategy, which maximizes the investor's expected utility.

The problem has 8 stages and 5760 scenarios. To compare the effectiveness of the different scenario reduction algorithms, we adopt a two step procedure. In the first step, the scenario reduced problem is solved and the first period's decision is observed. We then solve the full-size (non scenario reduced) problem to optimality with this first decision fixed. The difference between the objective values of these two solutions is normalized by the range [optimal solution, observed worst solution] to give a normalized error for committing to the scenario reduced first decision. In Fig. 1, we see that Dupacova et al's algorithm is very effective, that Latin hypercube sampling is a small distance behind, and both are far ahead of the most likely scenario method (which requires approximately half the scenarios before the first decision is made correctly).

8.3 Yield management

Farmers must deal with uncertainty since weather and many other factors affect crop yields. In this example (also taken from [Birge and Louveaux, 1997]), we must decide on how many acres of his fields to devote to various crops before the planting season. A certain amount of each crop is required for cattle feed, which can be purchased from a wholesaler if not raised on the farm. Any crop in excess of cattle feed can be sold up to the EU quota; any amount in excess of this quota

No. Stages	Backtracking (BT)		Forward Checking (FC)		Scenario-Based (SB)		
	Nodes	CPU/sec	Nodes	CPU/sec	Failure	Choice Points	CPU/sec
1	28	0.01	10	0.01	4	5	0.00
2	650	0.09	148	0.03	4	8	0.02
3	17,190	2.72	3,604	0.76	8	24	0.16
4	510,356	83.81	95,570	19.07	42	125	1.53
5	15,994,856	3,245.99	2,616,858	509.95	218	690	18.52
6	—	—	—	—	1260	4035	474.47

Table 1: A Comparison of BT, FC and SB Approaches on the Book Production Problem (Sec.8.1)

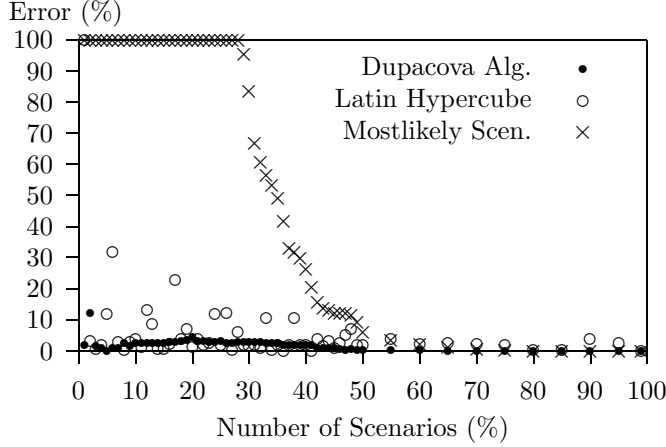


Figure 1: Portfolio Management

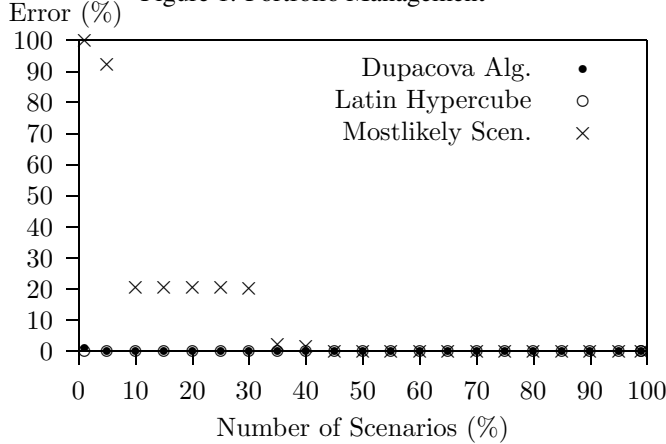


Figure 2: Agricultural Yield Management

will be sold at a low price. Crop yields are uncertain, depending upon weather conditions during the growing season. This problem has 4 stages and 10,000 scenarios. In Fig. 2, we again see that Dupacova et al's algorithm and Latin hypercube sampling are very effective, and both are far ahead of the most likely scenario method (which requires approximately one third the scenarios before the first decision is made correctly).

8.4 Production/Inventory control

Uncertainty plays a major role in production and inventory planning. In this simplified production/inventory planning example, there is a single product, a single stocking point,

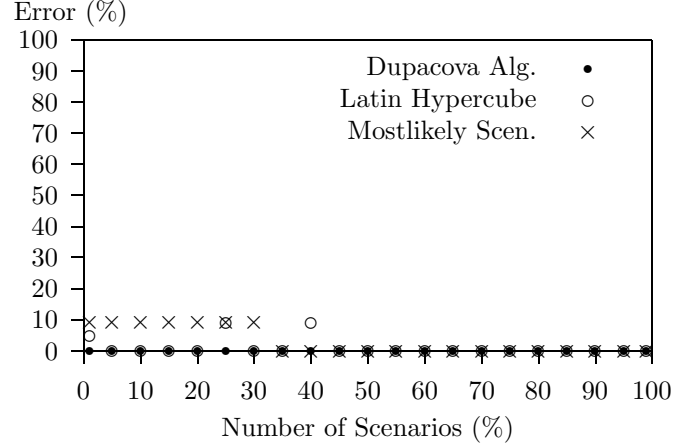


Figure 3: Production/Inventory Control

production capacity constraints and stochastic demand. The objective is to find the minimum expected cost policy. The cost components take into account holding costs, backlogging costs, fixed replenishment (or setup) costs and unit production costs. The optimal policy gives the timing of the replenishments as well as the order-up-to-levels. Hence, the exact order quantity can be known only after the realization of the demand, using the scenario dependent order-up-to-level decisions. This problem has 5 stages and 1,024 scenarios. In Fig. 3, we again see that Dupacova et al's algorithm and Latin hypercube sampling are very effective, but both are now only a small distance ahead of the most likely scenario method.

9 Robust solutions

Inspired by robust optimization methods in operations research [Kouvelis and Yu, 1996], stochastic OPL also allows us to find robust solutions to stochastic constraint programs. That is, solutions in which similar decisions are made in the different scenarios. It will often be impossible or undesirable for all decision variables to be robust. We therefore identify those decision variables whose values we wish to be identical across scenarios using commands of the form:

```
robust <Var>;
```

For example, in production/inventory problem of Sec.8.4 the decision variables "order-up-to-levels" and "replenishment periods" can be declared as robust variables. The values of these two sets of decision variables are then fixed at the beginning of the planning horizon. A robust solution dampens the nervousness of the solution, an area of very active re-

search in production/inventory management. As the expected cost of the robust solution is always higher, the tradeoff between nervousness and cost may have to be taken into account.

10 Related and future work

Stochastic constraint programs are closely related to Markov decision problems (MDPs) [Puterman, 1994]. Stochastic constraint programs can, however, model problems which lack the Markov property that the next state and reward depend only on the previous state and action taken. The current decision in a stochastic constraint program will often depend on all earlier decisions. To model this as an MDP, we would need an exponential number of states. Another significant difference is that stochastic constraint programs by using a scenario-based interpretation can immediately call upon complex and powerful constraint propagation techniques.

Stochastic constraint programming was inspired by both stochastic integer programming and stochastic satisfiability [Littman *et al.*, 2000]. It is designed to take advantage of some of the best features of each framework. For example, we are able to write expressive models using non-linear and global constraints, and to exploit efficient constraint propagation algorithms. In operations research, scenarios are used in stochastic programming. Indeed, the scenario reduction techniques of Dupacova, Growe-Kuska and Romisch [Dupacova *et al.*, 2003] implemented here are borrowed directly from stochastic programming.

There are a number of extensions of conventional constraint satisfaction problem to model constraints that are uncertain, probabilistic or not necessarily satisfied. For example, in probabilistic constraint satisfaction each constraint has a certain probability independent of all other probabilities of being part of the problem [Fargier and Lang, 1993] whilst in semi-ring constraint satisfaction each tuple in a constraint has a value associated with it [Bistarelli *et al.*, 1996]. However, none of these extensions deal with variables that may have uncertain or probabilistic values. Stochastic constraint programming could, however, easily be combined with most of these techniques.

11 Conclusions

To model combinatorial decision problems involving uncertainty and probability, we have extended the stochastic constraint programming framework proposed in [Walsh, 2002] along a number of important dimensions. In particular, we have relaxed the assumption that stochastic variables are independent, and added multiple chance constraints as well as a range of objective functions like maximizing the downside. We have also provided a new (but equivalent) semantics for stochastic constraint programs based on scenarios. Based on this semantics, we can compile stochastic constraint programs down into conventional (non-stochastic) constraint programs. The advantage of this compilation is that we can use the full power of existing constraint solvers without any modification. We have also proposed a number of techniques to reduce the number of scenarios, and to generate robust solutions.

We have implemented this framework for decision making under uncertainty in a language called stochastic OPL. This is an extension of the OPL constraint modelling language [Hentenryck *et al.*, 1999]. To illustrate the potential of this framework, we have modelled a wide range of problems in areas as diverse as finance, agriculture and production. There are many directions for future work. For example, we want to allow the user to define a limited set of scenarios that are representative of the whole. As a second example, we want to explore more sophisticated notions of solution robustness (e.g. limiting the range of values used by a decision variable).

Acknowledgements

This project was funded by EPSRC under GR/R30792, and the Science Foundation Ireland. We thank the members of the APES Research Group and 4C Lab for their feedback.

References

- [Birge and Louveaux, 1997] J. R. Birge and F. Louveaux. *Introduction to Stochastic Programming*. Springer-Verlag, New York, 1997.
- [Bistarelli *et al.*, 1996] S. Bistarelli, H. Fargier, U. Montanari, F. Rossi, T. Schiex, and G. Verfaillie. Semi-ring based CSPs and valued CSPs: Basic properties and comparison. In M. Jample, E. Freuder, and M. Maher, editors, *Over-Constrained Systems*, LNCS 1106, pages 111–150. Springer-Verlag, 1996.
- [Dupacova *et al.*, 2003] J. Dupacova, N. Growe-Kuska, and W. Romisch. Scenario reduction in stochastic programming: an approach using probability metrics. *Mathematical Programming*, Series A 95:493–511, 2003.
- [Fargier and Lang, 1993] H. Fargier and J. Lang. Uncertainty in constraint satisfaction problems: a probabilistic approach. In *Proceedings of ECSQARU*, LNCS 747. Springer-Verlag, 1993.
- [Hentenryck *et al.*, 1999] P. Van Hentenryck, L. Michel, L. Perron, and J-C. Regin. Constraint programming in OPL. In G. Nadathur, editor, *Principles and Practice of Declarative Programming*, LNCS 1702, pages 97–116. Springer-Verlag, 1999.
- [Kouvelis and Yu, 1996] P. Kouvelis and G. Yu. *Robust Discrete Optimization and Its Applications*. Nonconvex optimization and its applications: volume 14. Kluwer, 1996.
- [Littman *et al.*, 2000] M. L. Littman, S. M. Majercik, and T. Pitassi. Stochastic boolean satisfiability. *Journal of Automated Reasoning*, 2000.
- [McKay *et al.*, 1979] M. D. McKay, R. J. Beckman, and W. J. Conover. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 21(2):239–245, 1979.
- [Puterman, 1994] M. L. Puterman. *Markov Decision Process: Discrete Stochastic Dynamic Programming*. John Wiley and Sons, 1994.
- [Walsh, 2002] Toby Walsh. Stochastic constraint programming. In *Proceedings of the 15th ECAI*, European Conference on Artificial Intelligence. IOS Press, 2002.