

SEARCH

Faster Heuristic Search Algorithms for Planning with Uncertainty and Full Feedback

Blai Bonet

Computer Science Department
University of California, Los Angeles
Los Angeles, CA 90024, USA
bonet@cs.ucla.edu

Héctor Geffner

Departamento de Tecnología
ICREA – Universitat Pompeu Fabra
Barcelona 08003, España
hector.geffner@tecn.upf.es

Abstract

Recent algorithms like RTDP and LAO* combine the strength of Heuristic Search (HS) and Dynamic Programming (DP) methods by exploiting knowledge of the initial state and an admissible heuristic function for producing optimal policies without evaluating the entire space. In this paper, we introduce and analyze three new HS/DP algorithms. A first general algorithm schema that is a simple loop in which ‘inconsistent’ reachable states (i.e., with residuals greater than a given ϵ) are found and updated until no such states are found, and serves to make explicit the basic idea underlying HS/DP algorithms, leaving other commitments aside. A second algorithm, that builds on the first and adds a labeling mechanism for detecting solved states based on Tarjan’s strongly-connected components procedure, which is very competitive with existing approaches. And a third algorithm, that approximates the latter by enforcing the consistency of the value function over the ‘likely’ reachable states only, and leads to great time and memory savings, with no much apparent loss in quality, when transitions have probabilities that differ greatly in value.

1 Introduction

Heuristic search algorithms have been successfully used for computing optimal solutions to large deterministic problems (e.g., [Korf, 1997]). In the presence of non-determinism and feedback, however, solutions are not action sequences but policies, and while such policies can be characterized and computed by dynamic programming (DP) methods [Bellman, 1957; Howard, 1960], DP methods take all states into account and thus cannot scale up to large problems [Boutilier *et al.*, 1999]. Recent algorithms like RTDP [Barto *et al.*, 1995] and LAO* [Hansen and Zilberstein, 2001], combine the strength of Heuristic Search and Dynamic Programming methods by exploiting knowledge of the initial state s_0 and an admissible heuristic function (lower bound) h for computing optimal policies without having to evaluate the entire space. In this paper we aim to contribute to the theory and practice of Heuristic Search/Dynamic Programming methods by formulating and analyzing three new HS/DP algorithms. The first algorithm, called FIND-and-REVISE is a general schema

that comprises a loop in which states reachable from s_0 and the greedy policy that have Bellman residuals greater than a given ϵ (we call them ‘inconsistent’ states), are found and updated until no one is left. FIND-and-REVISE makes explicit the basic idea underlying HS/DP algorithms, including RTDP and LAO*. We prove the convergence, complexity, and optimality of FIND-and-REVISE, and introduce the second algorithm, HDP, that builds on it, and adds a labeling mechanism for detecting *solved* states based on Tarjan’s efficient strongly-connected components procedure [Tarjan, 1972]. A state is solved when it reaches only ‘consistent’ states, and solved states are skipped in all future searches. HDP terminates when the initial state is solved. HDP inherits the convergence and optimality properties of the general FIND-and-REVISE schema and is strongly competitive with existing approaches. The third algorithm, HDP(i), is like HDP, except that while HDP computes a value function by enforcing its consistency over all reachable states (i.e., reachable from s_0 and the greedy policy), HDP(i) enforces consistency over the ‘likely’ reachable states only. We show that this approximation, suitably formalized, can lead to great savings in time and memory, with no much apparent loss in quality, when transitions have probabilities that differ greatly in value.

Our motivation is twofold: to gain a better understanding of HS/DP methods for planning with uncertainty, and to develop more effective HS/DP algorithms for both optimal and approximate planning.

2 Preliminaries

2.1 Model

We model non-deterministic planning problems with full feedback with state models that differ from those used in the classical setting in two ways: first, state transitions become *probabilistic*; second, states are fully *observable*. The resulting models are known as Markov Decision Processes (MDPs) and more specifically as *Stochastic Shortest-Path Problems* [Bertsekas, 1995], and they are given by:¹

- M1. a discrete and finite state space S ,
- M2. an initial state $s_0 \in S$,
- M3. a set $G \subseteq S$ of goal states,
- M4. actions $A(s) \subseteq A$ applicable in each state $s \in S$,
- M5. transition probabilities $P_a(s'|s)$ for $s \in S$, $a \in A(s)$,

¹For discounted and other formulations, see [Puterman, 1994].

- M6. positive action costs $c(a, s) > 0$, and
M7. fully observable states.

Due to the presence of full feedback (M7) and the standard Markovian assumptions, the solution of an MDP takes the form of a function π mapping states s into actions $a \in A(s)$. Such a function is called a *policy*. A policy π assigns a probability to every state trajectory $s_0, s_1, s_2, \dots$ starting in state s_0 which is given by the product of the transition probabilities $P_{a_i}(s_{i+1}|s_i)$ where $a_i = \pi(s_i)$. If we further assume that actions in goal states have no costs and produce no changes (i.e., $c(a, s) = 0$ and $P_a(s|s) = 1$ if $s \in G$), the *expected cost* associated with a policy π starting in state s_0 is given by the weighted average of the probability of such trajectories times their cost $\sum_{i=0}^{\infty} c(\pi(s_i), s_i)$. An *optimal solution* is a policy π^* that has a minimum expected cost for all possible initial states. An optimal solution is guaranteed to exist provided the following assumption holds [Bertsekas, 1995]:

- M8. the goal is reachable from every state with non-zero probability.

Since the initial state s_0 of the system is fixed, there is in principle no need to compute a complete policy but a *partial* policy prescribing the action to take in the states that can be reached following the policy from s_0 . Traditional dynamic programming methods like value or policy iteration compute complete policies, while recent heuristic search DP methods like RTDP and LAO* compute partial policies. They achieve this by means of suitable heuristic functions $h(s)$ that provide admissible estimates (lower bounds) of the expected cost to reach the goal from any state s .

2.2 Dynamic Programming

Any heuristic or value function h defines a *greedy policy* π_h :

$$\pi_h(s) \stackrel{\text{def}}{=} \underset{s' \in A(s)}{\operatorname{argmin}} c(a, s) + \sum_{s' \in S} P_a(s'|s) h(s') \quad (1)$$

where the expected cost from the resulting states s' is assumed to be given by $h(s')$. We call $\pi_h(s)$ the *greedy action* in s for the value function h . If we denote the optimal (expected) cost from a state s to the goal by $V^*(s)$, it is well known that the greedy policy π_h is *optimal* when h is the *optimal cost function*, i.e. $h = V^*$.

While due to the possible presence of ties in (1), the greedy policy is not unique, we will assume throughout the paper that these ties are broken systematically using an static ordering on actions. As a result, every value function V defines a *unique* greedy policy π_V , and the *optimal* cost function V^* defines a unique optimal policy π_{V^*} . We define the *relevant states* as the states that are reachable from s_0 using this optimal policy; they constitute a minimal set of states over which the optimal value function needs to be defined.²

Value iteration (VI) is a standard dynamic programming method for solving MDPs and is based on computing the optimal cost function V^* and plugging it into the greedy policy (1). This optimal cost function is the only solution to the fixed

²This definition of ‘relevant states’ is more restricted than the one in [Barto *et al.*, 1995] that includes the states reachable from s_0 by any optimal policy.

point equation:

$$V(s) = \min_{a \in A(s)} c(a, s) + \sum_{s' \in S} P_a(s'|s) V(s') \quad (2)$$

also known as Bellman’s equation. For stochastic shortest path problems like M1-M8 above, the border condition $V(s) = 0$ is also needed for goal states $s \in G$. Value iteration solves (2) by plugging an initial guess for V^* in the right-hand side of (2) and obtaining a new guess on the left-hand side. In the form of VI known as *asynchronous value iteration* [Bertsekas, 1995], this operation can be expressed as:

$$V(s) := \min_{a \in A(s)} c(a, s) + \sum_{s' \in S} P_a(s'|s) V(s') \quad (3)$$

where V is a vector of size $|S|$ initialized arbitrarily (normally to 0) and where the equality in (2) is replaced by assignment. The use of expression (3) for updating a state value in V is called a state update or simply an *update*. In standard (synchronous) value iteration, all states are updated in parallel, while in asynchronous value iteration, only a selected subset of states is selected for update at a time. In both cases, it is known that if all states are updated infinitely often, the value function V converges eventually to the optimal value function. From a practical point of view, value iteration is stopped when the Bellman error or *residual* defined as the difference between left and right in (2):

$$R(s) \stackrel{\text{def}}{=} \left| V(s) - \left(\min_{a \in A(s)} c(a, s) + \sum_{s' \in S} P_a(s'|s) V(s') \right) \right|$$

over all states s is sufficiently small. In the discounted MDP formulation, a bound on the policy loss (the difference between the expected cost of the policy and the expected cost of the optimal policy) can be obtained as a simple expression of the discount factor and the maximum residual. In stochastic shortest path models, no similar closed-form bound is known, although such bound can be computed [Bertsekas, 1995]. Thus, one can execute value iteration until the maximum residual becomes smaller than a given ϵ , then if the bound on the policy loss is higher than desired, the same process can be repeated with a smaller ϵ (e.g., $\epsilon/2$) and so on (see [Hansen and Zilberstein, 2001] for a similar idea). For these reasons, we will take as our *basic task* below, the computation of a value function $V(s)$ with residuals no greater than a given parameter $\epsilon > 0$.

One last definition and a few known results before we proceed. We say that cost function V is *monotonic* iff

$$V(s) \leq \min_{a \in A(s)} c(a, s) + \sum_{s' \in S} P_a(s'|s) V(s') \quad (4)$$

for every $s \in S$. Notice that a monotonic value function never decreases when updated, and moreover, must increase by more than ϵ when updated in a state s whose residual is greater than ϵ . As in the deterministic setting, a non-monotonic cost function can be made monotonic by simply taking the value $V(s)$ to be the max between $V(s)$ and the right-hand side of Bellman’s equation. The following results are well known.

Theorem 1 *a) The optimal values $V^*(s)$ of a model M1-M8 are non-negative and finite; b) the monotonicity and admissibility of a value function are preserved through updates.*

start with a lower bound function $V := h$

repeat

 FIND a state s in the greedy graph G_V with $R(s) > \epsilon$
 REVISE V at s

until no such state is found

return V

Algorithm 1: FIND-and-REVISE

3 Find-and-Revise

The FIND-and-REVISE schema is a general asynchronous VI algorithm that exploits knowledge of the initial state and an admissible heuristic for computing optimal or nearly optimal policies without having to evaluate the entire space. Let us say that a value function V is ϵ -consistent (inconsistent) over a state s when the residual over s is no greater (greater) than ϵ , and that V itself is ϵ -consistent when it is ϵ -consistent over all the states reachable from s_0 and the greedy policy π_V . Then FIND-and-REVISE computes an ϵ -consistent value function by simply searching for inconsistent states in the greedy graph and updating them until no such states are left; see Alg. 1.

The **greedy graph** G_V refers to the graph resulting from the execution of the greedy policy π_V starting in s_0 ; i.e., s_0 is the single root node in G_V , and for every non-goal state s in G_V , its children are the states that may result from executing the action $\pi(s)$ in s .

The procedures FIND and REVISE are the two parameters of the FIND-and-REVISE procedure. For the convergence, optimality, and complexity of FIND-and-REVISE, we assume that FIND searches the graph systematically, and REVISE of V at s updates V at s (and possibly at some other states), both operations taking $O(|S|)$ time.

Theorem 2 (Convergence) *For a planning model M1-M8 with an initial value function h that is admissible and monotonic, FIND-and-REVISE yields an ϵ -consistent value function in a number of loop iterations no greater than $\epsilon^{-1} \sum_{s \in S} V^*(s) - h(s)$, where each iteration has time complexity $O(|S|)$.*

Theorem 3 (Optimality) *For a planning model M1-M8 with an initial admissible and monotonic value function, the value function computed by FIND-and-REVISE approaches the optimal value function over all relevant states as ϵ goes to 0.*

4 Labeling

We consider next a particular instance of the general FIND-and-REVISE schema in which the FIND operation is carried out by a systematic *Depth-First Search* that keeps track of the states visited. In addition, we consider a *labeling scheme* on top of this search that detects, with almost no overhead, when a state is *solved*, and hence, when it can be skipped in all future searches. A state s is defined as *solved* when the value function V is ϵ -consistent over s and over all states reachable from s and the greedy policy π_V . Clearly, when this condition holds no further updates are needed in s or the states reachable from s . The resulting algorithm terminates when the initial state s_0 is solved and hence when an ϵ -consistent value function has been obtained.

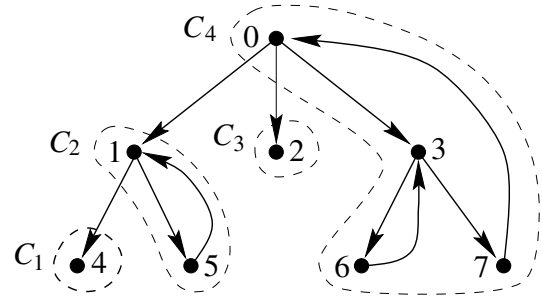


Figure 1: A graph and its strongly-connected components.

Due to the presence of cycles in the greedy graph, bottom-up algorithms common in AO* implementations cannot be used. Indeed, if s is reachable (in the greedy graph) from a descendant s' of s , then bottom-up approaches will be unable to label either state as solved. A labeling mechanism that works in the presence of cycles is presented in [Bonet and Geffner, 2003] for improving the convergence of RTDP. Basically, after each RTDP trial, an attempt is made to label the last unsolved state s in the trial by triggering a systematic search for inconsistent states from s . If one such state is found, it is updated, and a new trial is executed. Otherwise, the state s and all its unsolved descendants are labeled as solved, and a new cycle of RTDP trials and labeling checks is triggered. Here we take this idea and improve it by removing the need of an extra search for label checking. The label checking will be done as part of the FIND (DFS) search with almost no overhead, exploiting Tarjan's linear algorithm for detecting the *strongly-connected components* of a directed graph [Tarjan, 1972], and a correspondence between the strongly-connected components of the greedy graph and the minimal collections of states that can be labeled at the same time.

Consider the (directed) greedy graph G_V and the relation ' $\sim$ ' between pairs of states s and s' that holds when $s = s'$ or when s is reachable from s' and s' is reachable from s in G_V . The strongly-connected components of G_V are the equivalence classes defined by this relation and form a partition of the set of states in G_V . For example, for the greedy graph in Fig. 1, where 2 and 4 are terminal (goal) states, the components are $C_1 = \{4\}$, $C_2 = \{1, 5\}$, $C_3 = \{2\}$, and $C_4 = \{0, 3, 6, 7\}$. Tarjan's algorithm detects the strongly-connected components of a directed graph in time $O(n + e)$ while traversing the graph *depth-first*, where n stands for the number of states ($n \leq |S|$ in G_V) and e for the number of edges.

The relationship between labeling and strongly-connected components in G_V is quite direct. Let us say first that a *component* C is ϵ -consistent when all states $s \in C$ are ϵ -consistent, and that a *component* C is *solved* when every state $s \in C$ is solved. Let's then define G_V^C as the graph whose nodes are the components of G_V and whose directed edges are $C \rightarrow C'$ when some state in C' is reachable from some state in C . Clearly, G_V^C is an *acyclic graph* as two components which are reachable from each other will be collapsed into the same equivalence class. In addition,

1. a state s is solved iff its component C is solved, and furthermore,

2. a component C is solved iff C is consistent and all components C' , $C \rightarrow C'$, are solved.

The problem of *labeling states in the cyclic graph* G_V can thus be mapped into the problem of *labeling the components in the acyclic graph* G_V^C , which can be done in bottom up fashion.

From Fig. 1 is easy to visualize the component graph associated to the greedy graph. Thus, if 2 is the only inconsistent state, for example, we can label the components C_1 and C_2 as solved, while leaving C_3 and C_4 unsolved.

The code that simultaneously checks in depth-first fashion the consistency of the states and the possibility of labeling them is shown in Alg. 2. We call the resulting algorithm, HDP. HDP inherits its convergence and optimality properties from the FIND-and-REVISE schema and the correctness of the labeling mechanism.

We do not have space to explain HDP code in detail, yet it should be clear to those familiar with Tarjan's algorithm; in particular, the use of the state visit number, S.IDX, and the 'low-link' number, S.LOW, for detecting when a new component has been found. The flag *flag* and the (normal) propagation of the visit numbers prevent a component from being labeled as solved when it is inconsistent or can reach an inconsistent component.

Theorem 4 (Correctness) *The value function computed by HDP for a planning model M1-M8, given an initial admissible and monotonic value function, is ϵ -consistent.*

5 Experimental Results

We now evaluate the performance of HDP in comparison with other recent Heuristic Search/DP algorithms such as the second code for LAO* in [Hansen and Zilberstein, 2001], that we call Improved LAO* (ILAO*), and Labeled RTDP (LRTDP), a recent improvement of RTDP that accelerates its convergence [Bonet and Geffner, 2003]. We use parallel Value Iteration as the baseline. We've implemented all these algorithms in C++ and the experiments have been run on a Sun Fire-280R with 750 MHz and 1Gb of RAM.

The domain that we use for the experiments is the racetrack from [Barto *et al.*, 1995]. The states are tuples (x, y, dx, dy) that represent the position and speed of the car in the x, y dimensions. The actions are pairs $a = (ax, ay)$ of instantaneous accelerations where $ax, ay \in \{-1, 0, 1\}$. Uncertainty in this domain comes from assuming that the road is 'slippery' and as a result, the car may fail to accelerate or desynchronize. More precisely, an action $a = (ax, ay)$ has its intended effect with probability $1 - p$, while with probability p the action effects correspond to those of the action $a_0 = (0, 0)$. Also, when the car hits a wall, its velocity is set to zero and its position is left intact (this is different than in [Barto *et al.*, 1995] where for some reason the car is moved to the start position).

We consider the track `large-b` from [Barto *et al.*, 1995], `h-track` from [Hansen and Zilberstein, 2001],³ and five other tracks (squares and rings of different size). Information about these instances can be found in the first three rows

³Taken from the source code of LAO*.

```

HDP( $s : state$ )
begin
  while  $\neg s.SOLVED$  do
    // perform DFS
    index := 0
    DFS( $s$ )
    [reset IDX to  $\infty$  for visited states]
    [clean stack and visited]
  end
end

DFS( $s : state$ )
begin
  // base case
  if  $s.SOLVED \vee s.GOAL$  then
     $s.SOLVED := true$ 
    return false

  // check residual
  if  $s.RESIDUAL > \epsilon$  then
     $s.UPDATE()$ 
    return true

  // mark state as active
  visited.PUSH( $s$ )
  stack.PUSH( $s$ )
   $s.IDX := s.LOW := index$ 
  index := index + 1

  // recursive call
  flag := false
  for  $s' \in s.SUCCESSORS$  do
    if  $s'.IDX = \infty$  then
      flag := flag  $\vee$  DFS( $s'$ )
       $s.LOW := \min\{s.LOW, s'.LOW\}$ 
    else if  $s' \in stack$  then
       $s.LOW := \min\{s.LOW, s'.IDX\}$ 

  // update if necessary
  if flag then
     $s.UPDATE()$ 
    return true

  // try to label
  else if  $s.IDX = s.LOW$  then
    while stack.TOP  $\neq s$  do
       $s' := stack.POP()$ 
       $s'.SOLVED := true$ 
    stack.POP()
     $s.SOLVED := true$ 
  return flag
end

```

Algorithm 2: HDP.

of Table 1, including number of states, optimal cost, and percentage of states that are relevant.

As heuristic, we follow [Bonet and Geffner, 2003], and use the domain independent admissible and monotonic heuristic h_{min} , obtained by replacing the expected cost in Bellman equation by the best possible cost. The total time spent computing heuristic values is roughly the same for the different algorithms (except VI), and is shown separately in the fifth row in the table, along with its value for s_0 . The experiments are carried with three heuristics: $h = h_{min}$, $h = h_{min}/2$, and $h = 0$.

The results are shown in Table 1. HDP dominates the other algorithms over all the instances for $h = h_{min}$, while LRTDP is best (with one or two exceptions) when the weaker heuristics $h_{min}/2$ and 0 are used. Thus, while HDP seems best for exploiting good heuristic information over these instances, LRTDP bootstraps more quickly (i.e., it quickly computes a good value function). We hope to understand the reasons for

algorithm	large-b	h-track	square-1	square-2	ring-1	ring-2	ring-3	ring-4
$ S $	23880	53597	42071	383950	5895	33068	94369	353991
$V^*(s_0)$	18.73356	41.89504	8.07350	11.13041	11.04923	16.09833	22.04798	27.78572
% relevant	18.58	17.12	1.79	0.25	10.70	11.10	10.91	9.98
$h_{min}(s_0)$	16.0	36.0	7.0	10.0	10.0	14.0	19.0	24.0
time for h_{min}	3.157	13.422	4.509	72.853	0.555	4.558	16.745	85.766
$VI(h_{min})$	4.039	12.620	5.873	81.270	0.614	5.287	19.466	90.895
ILAO * (h_{min})	3.776	13.062	0.389	0.942	0.332	6.095	24.422	145.047
LRTDP(h_{min})	3.230	7.618	0.179	0.363	0.143	1.301	4.928	37.992
HDP(h_{min})	1.888	7.547	0.148	0.275	0.121	1.101	4.145	30.511
$VI(h_{min}/2)$	4.069	14.275	6.147	89.248	0.636	5.704	24.385	100.130
ILAO * ($h_{min}/2$)	6.346	26.057	2.106	68.028	0.765	9.801	32.415	174.089
LRTDP($h_{min}/2$)	4.744	12.987	0.813	7.736	0.302	2.480	12.687	152.200
HDP($h_{min}/2$)	6.402	26.663	1.255	20.952	0.331	3.536	15.982	98.033
$VI(h=0)$	5.785	21.021	8.445	89.152	0.675	5.809	20.979	101.616
ILAO * ($h=0$)	9.522	46.754	11.230	187.463	1.290	9.894	45.989	310.876
LRTDP($h=0$)	6.435	15.276	3.286	45.514	0.431	3.682	19.424	261.286
HDP($h=0$)	9.752	42.601	3.579	79.171	0.994	6.173	25.992	157.646

Table 1: Problem data and convergence time in seconds for the different algorithms with different heuristics. Results for $\epsilon = 10^{-3}$ and probability $p = 0.2$. Faster times are shown in bold.

these differences in the future.

6 Approximation

HDP, like FIND-and-REVISE, computes a value function V by enforcing its consistency over the states reachable from s_0 and the greedy policy π_V . The final variation we consider, that we call HDP(i), works in the same way, yet it enforces the consistency of the value function V *only* over the states that are reachable from s_0 and the greedy policy with some *minimum likelihood*.

For efficiency, we formalize this notion of likelihood, using a non-negative integer scale, where 0 refers to a normal outcome, 1 refers to a somewhat surprising outcome, 2 to a still more surprising outcome, and so on. We call these measures *plausibilities*, although it should be kept in mind, that 0 refers to the most plausible outcomes, thus ‘plausibility greater than i ’, means ‘a plausibility *measure* smaller than or equal to i .’

We obtain the transition plausibilities $\kappa_a(s'|s)$ from the corresponding transition probabilities by the following discretization:

$$\kappa_a(s'|s) \stackrel{\text{def}}{=} \lfloor -\log_2(P_a(s'|s) / \max_{s''} P_a(s''|s)) \rfloor \quad (5)$$

with $\kappa_a(s'|s) = \infty$ when $P_a(s'|s) = 0$. Plausibilities are thus ‘normalized’: the most plausible next states have always plausibility 0. These transition plausibilities are then combined by the rules of the κ calculus [Spohn, 1988] which is a calculus isomorphic to the probability calculus (e.g. [Goldschmidt and Pearl, 1996]). The plausibility of a state trajectory given the initial state, is given by the sum of the transition plausibilities in the trajectory, and the plausibility of reaching a state, is given by the plausibility of the most plausible trajectory reaching the state.

The HDP(i) algorithm, for a non-negative integer i , computes a value function V by enforcing its (ϵ)-consistency over the states reachable from s_0 with *plausibility greater than or equal to i* . HDP(i) produces approximate policies fast by pruning certain paths in the search. The simplest case results

from $i = 0$, as the code for HDP(0) corresponds exactly to the code for HDP, except that the possible successors of a state s in the greedy graph are replaced by the *plausible* successors.

HDP(i) computes lower bounds that tend to be quite tight over the states that can be reached with plausibility no smaller than i . At run time, however, executions may contain ‘surprising’ outcomes, taking the system ‘out’ of this envelope, into states where the quality of the value function and its corresponding policy, are poor. To deal with those situations, we define a version of HDP(i), called HDP(i, j), that *interleaves planning and execution* as follows. HDP(i, j) plans from $s = s_0$ by means of the HDP(i) algorithm, then executes this policy until a state trajectory with plausibility measure greater than or equal to j , and leading to a (non-goal) state s' is obtained. At that point, the algorithm replans from s' with HDP(i), and the same execution and replanning cycle is followed until reaching the goal. Clearly, for sufficiently large j , HDP(i, j) reduces to HDP(i), and for large i , HDP(i) reduces to HDP.

Table 2 shows the average cost for HDP(i, j) for $i = 0$ (i.e., most plausible transitions considered only), and several values for j (replanning thresholds). Each entry in the table correspond to an average over 100 independent executions. We also include the average cost for the greedy policy with respect to h_{min} as a bottom-line reference for the figures. Memory in the table refers to the number of evaluated states. As these results show, there is a smooth tradeoff between quality (average cost to the goal) and time (spent in initial planning and posterior replannings) as the parameter j vary. We also see that in this class of problems the h_{min} heuristic delivers a very good greedy policy. Thus, further research is necessary to assess the goodness of HDP(i, j) and the h_{min} heuristic.

7 Related Work

We have built on [Barto *et al.*, 1995] and [Bertsekas, 1995], and more recently on [Hansen and Zilberstein, 2001] and [Bonet and Geffner, 2003]. The crucial difference between

algorithm	h-track			square-2			ring-3			ring-4		
	time	quality	memory	time	quality	memory	time	quality	memory	time	quality	memory
HDP(h_{min})	7.547	41.894	35835	0.275	11.130	7245	4.145	22.047	43358	30.511	27.785	152750
HDP(0, 2)	0.853	42.950	7132	0.021	11.250	819	0.311	22.000	7145	1.758	28.500	24195
HDP(0, 4)	0.826	44.000	7034	0.022	11.500	650	0.327	23.300	6882	1.821	28.400	23857
HDP(0, 16)	0.701	46.800	6100	0.017	11.500	556	0.284	23.600	6331	1.580	30.800	21823
HDP(0, 64)	0.698	46.800	5899	0.014	11.610	564	0.264	25.500	6322	1.518	32.400	21823
greedy(h_{min})	N/A	47.150	356	N/A	12.450	104	N/A	25.390	192	N/A	31.600	241

Table 2: Results of HDP(0, j) for $j = 2, 4, 16, 64$ and greedy policy with respect to h_{min} for $\epsilon = 10^{-3}$ and $p = 0.2$. Each value is the average over 100 executions. N/A in time for the greedy policy means “Not Applicable” since there is no planning.

FIND-and-REVISE and general asynchronous value iteration is the focus of the former on the states that are reachable from the initial state s_0 and the greedy policy. In RTDP, the FIND procedure is not systematic and is carried out by a stochastic simulation that may take time greater than $O(|S|)$ when the inconsistent states are reachable with low probability (this explains why RTDP final convergence is slow; see [Bonet and Geffner, 2003]). LAO*, on the other hand, keeps track of a subset of states, which initially contains s_0 only, and over which it incrementally maintains an optimal policy through a somewhat expensive REVISE (full DP) procedure. This is then relaxed in the second algorithm in [Hansen and Zilberstein, 2001], called Improved LAO* here. The use of an explicit envelope that is gradually expanded is present also in [Dean *et al.*, 1993] and [Tash and Russell, 1994]. Interestingly, these envelopes are expanded by including the most ‘likely’ reachable states not yet in the envelope. The algorithm HDP(i) exploits a similar idea but formulates it in a different form and has a crisp termination condition.

8 Summary

We have introduced and analyzed three HS/DP algorithms that exploit knowledge of the initial state and an admissible heuristic function for solving planning problems with uncertainty and feedback: FIND-and-REVISE, HDP, and HDP(i). FIND-and-REVISE makes explicit the basic idea underlying HS/DP algorithms: inconsistent states are found and updated, until no one is left. We have proved its convergence, complexity, and optimality. HDP adds a labeling mechanism based on Tarjan’s SCC algorithm and is strongly competitive with current algorithms. Finally, HDP(i) and HDP(i, j) offer great time and memory savings, with no much apparent loss in quality, in problems where transitions have probabilities that differ greatly in value, by focusing the updates on the states that are more *likely* to be reached.

Acknowledgements: We thank Eric Hansen and Shlomo Zilberstein for making the code for LAO* available to us. Blai Bonet is supported by grants from NSF, ONR, AFOSR, DoD MURI program, and by a USB/CONICIT fellowship from Venezuela. Héctor Geffner is supported by grant TIC2002-04470-C03-02 from MCyT, Spain.

References

[Barto *et al.*, 1995] A. Barto, S. Bradtke, and S. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72:81–138, 1995.

- [Bellman, 1957] R. Bellman. *Dynamic Programming*. Princeton University Press, 1957.
- [Bertsekas, 1995] D. Bertsekas. *Dynamic Programming and Optimal Control*, (2 Vols). Athena Scientific, 1995.
- [Bonet and Geffner, 2003] B. Bonet and H. Geffner. Labeled RTDP: Improving the convergence of real-time dynamic programming. In *Proc. ICAPS-03*. To appear, 2003.
- [Boutilier *et al.*, 1999] C. Boutilier, T. Dean, and S. Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11:1–94, 1999.
- [Dean *et al.*, 1993] T. Dean, L. Kaelbling, J. Kirman, and A. Nicholson. Planning with deadlines in stochastic domains. In R. Fikes and W. Lehnert, editors, *Proc. 11th National Conf. on Artificial Intelligence*, pages 574–579, Washington, DC, 1993. AAAI Press / MIT Press.
- [Goldszmidt and Pearl, 1996] M. Goldszmidt and J. Pearl. Qualitative probabilities for default reasoning, belief revision, and causal modeling. *Artificial Intelligence*, 84:57–112, 1996.
- [Hansen and Zilberstein, 2001] E. Hansen and S. Zilberstein. LAO*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, 129:35–62, 2001.
- [Howard, 1960] R. Howard. *Dynamic Programming and Markov Processes*. MIT Press, Cambridge, MA, 1960.
- [Korf, 1997] R. Korf. Finding optimal solutions to rubik’s cube using patterns databases. In B. Kuipers and B. Weber, editors, *Proc. 14th National Conf. on Artificial Intelligence*, pages 700–705, Providence, RI, 1997. AAAI Press / MIT Press.
- [Puterman, 1994] M. Puterman. *Markov Decision Processes – Discrete Stochastic Dynamic Programming*. John Wiley and Sons, Inc., 1994.
- [Spohn, 1988] W. Spohn. A general non-probabilistic theory of inductive reasoning. In *Proc. 4th Conf. on Uncertainty in Artificial Intelligence*, pages 149–158, New York, NY, 1988. Elsevier Science Publishing Company, Inc.
- [Tarjan, 1972] R. E. Tarjan. Depth first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.
- [Tash and Russell, 1994] J. Tash and S. Russell. Control strategies for a stochastic planner. In B. Hayes-Roth and R. Korf, editors, *Proc. 12th National Conf. on Artificial Intelligence*, pages 1079–1085, Seattle, WA, 1994. AAAI Press / MIT Press.

Comparing Best-First Search and Dynamic Programming for Optimal Multiple Sequence Alignment

Heath Hohwald, Ignacio Thayer, Richard E. Korf

Computer Science Department
University of California, Los Angeles
Los Angeles, CA 90095
Email: {heath,iet,korf}@cs.ucla.edu

Abstract

Sequence alignment is an important problem in computational biology. We compare two different approaches to the problem of optimally aligning two or more character strings: bounded dynamic programming (BDP), and divide-and-conquer frontier search (DCFS). The approaches are compared in terms of time and space requirements in 2 through 5 dimensions with sequences of varying similarity and length. While BDP performs better in two and three dimensions, it consumes more time and memory than DCFS for higher-dimensional problems.

1 Introduction and Overview

Aligning multiple DNA or protein sequences, known as sequence alignment, is an important problem in computational biology. Sequence alignment is useful for comparing genomes and finding genes, for determining evolutionary linkage of different biological sequences, and for predicting protein sequence secondary and tertiary structure [Durbin *et al.*, 1998], [Waterman, 1995]. We consider two algorithms for finding an optimal alignment of two or more sequences: bounded dynamic programming (BDP), and divide-and-conquer frontier search (DCFS), a best-first search algorithm.

Dynamic programming (DP) is the traditional approach for solving sequence alignment problems. DCFS is a more recent approach, and the two methods have been compared for the problem of aligning two and three sequences [Korf and Zhang, 2000]. We extend this work by comparing the two algorithms for simultaneously aligning larger numbers of sequences, and find that DCFS is both faster and can solve larger problems, suggesting that the more traditional DP approach is not the best choice for aligning more than three sequences.

We begin by introducing to the multiple sequence alignment problem, and show how it can be mapped to the problem of finding a lowest-cost corner-to-corner path in a d -dimensional grid. We describe the dynamic programming approach in Section 3, and the DCFS algorithm in Section 4. We discuss the heuristic evaluation functions we use for sequence alignment in Section 5. In Section 6, we explain

the methodology used in comparing the two approaches, and present our results. Future work is discussed in Section 7, and we conclude in Section 8.

2 Sequence Alignment and The Grid Problem

2.1 Pairwise sequence alignment and cost function

Two sequences are aligned by inserting gaps into each of the sequences so that each character in one sequence corresponds to a character or gap in the other. Each pair of corresponding symbols (gaps or characters) in the sequences can be characterized as a match (two identical characters), substitution (two different characters), or gap (gap in one sequence but not the other). For example, given the sequences ACGTACGTACGT and ATGTCGTCACGT, one alignment is as follows:

```
ACGTACGT-ACGT
ATGT-CGTCACGT
```

The cost of an alignment is calculated by assigning a cost to each position of the two sequences, and then summing the costs over all positions. For example, a cost function might charge 0 units for matching characters, 1 unit for a substitution, and 2 units for a gap. If this cost function is applied to the alignment above, the result is a total cost of 5, since we have two gaps (marked by a '-'), and one substitution (T for C in the second position). An optimal alignment is one that minimizes a given cost function. For these sequences and this cost function, this alignment is optimal.

2.2 Multiple sequence alignment and cost function

The problem of optimally aligning multiple sequences is a natural extension of pairwise alignment: insert gaps in the sequences such that a given cost function is minimized. For multiple alignments, we use the sum-of-pairs (SP) cost function, i.e. the cost of an alignment of multiple sequences is the sum of the cost of all induced pairwise alignments. The cost of each position is still determined by a given cost function such as that described in the previous section. As an example, consider the following alignment of three sequences:

- 1) AGTTA-
- 2) AGCT-G
- 3) -GACAG

Using the same cost function above, the cost of this alignment is 17, obtained by summing the costs of the pairwise alignments of sequences [1,2] (5), [1,3] (6), and [2,3] (6).

2.3 The grid problem

The sequence alignment problem can be mapped to the problem of finding a lowest-cost corner-to-corner path in a directed grid, where each dimension corresponds to one of the sequences [Needleman and Wunsch, 1970]. We associate a cost with each edge in the grid, and the cost of an alignment is the sum of the edge costs along the corresponding path. In the directed-grid problem, we are only allowed to move toward the goal, either orthogonally or diagonally.

In two dimensions, the directed-grid problem is to find the lowest-cost path in an $x \times y$ grid, where x and y are the lengths of the two strings. The path goes from the upper-lefthand corner to the lower-righthand corner, and the legal moves are to the right, down, or diagonally right and down. For example, for the two sequences in Subsection 2.1, the corresponding grid and the optimal solution path are shown in Figure 1. The horizontal move corresponds to the gap in the vertical sequence, while the vertical move corresponds to the gap in the horizontal sequence. Diagonal moves correspond to matches or substitutions.

In three dimensions, we have a three-dimensional grid of size $x \times y \times z$ and can move in one of seven directions from each node in the grid: to the right (along the x axis), down (along the y axis), back (along the z axis), right and down, right and back, down and back, and finally right, down, and back. In general, the k -dimensional directed-grid problem seeks a lowest-cost path in a k -dimensional grid, with $2^k - 1$ possible moves from each node. Aligning k sequences of length l requires a grid of size l^k .

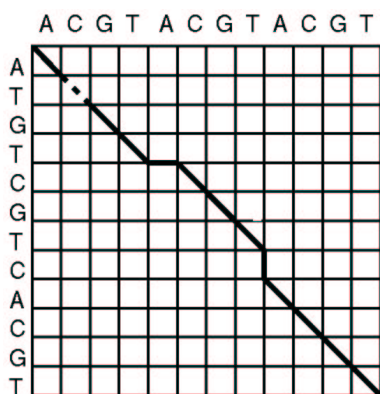


Figure 1: Optimal alignment of the two sequences from Subsection 2.1. The solid line represents the solution path, and the dashed line represents a substitution.

3 Dynamic Programming

3.1 Standard dynamic programming

Dynamic programming (DP) is a general technique that can be used to find a lowest-cost path in a directed grid. Note that any optimal path passing through a node n must include an optimal path passing through one of its predecessors. To determine the lowest-cost path to a node n , we do not need to consider all possible paths to the node; rather, we only need to know the costs to reach each of n 's immediate predecessors.

This observation yields an efficient algorithm for searching a directed grid, which we describe for the case of an $x \times y$ grid. We scan the grid from left to right and from the top to the bottom, and at each node n we store the cost of a lowest-cost path from the start node to n . This cost is the minimum obtained by adding the cost of the edge from the left to the cost of the node to the left, the cost of the edge from above to the cost of the node above, and the cost of the edge from diagonally above and left to the cost of the node diagonally above and to the left. When we reach the goal node, we have the cost of an optimal path and can trace back through the grid to find the associated path. DP has both $O(x \times y)$ time and space complexity, since we store costs for all nodes and compute the cost at each node in constant time. DP can be generalized to k dimensions, where the time and space complexity is $O(l^k)$ for a hypercube with length l .

3.2 Hirschberg's divide-and-conquer method

Using a divide-and-conquer approach proposed by [Hirschberg, 1975], the space complexity of DP is reduced from $O(l^k)$ to $O(l^{k-1})$. This reduction allows much larger problems to be solved. Hirschberg's method is best illustrated in the two-dimensional grid problem. We find a node n in the middle row of the grid that is guaranteed to be on an optimal path and then recursively solve two subproblems: finding an optimal path from the start node to n , and finding an optimal path from n to the goal node. In order to find such a node n , we first calculate the costs from left to right in each row from the top to the middle row, and then calculate costs from the right to left in each row from the bottom to the middle row. When we have forward and backward costs to reach the middle row, we add the corresponding entries for each node and take the node of minimum total cost to be node n , a node on the optimal corner-to-corner path. Storing these costs requires two rows in memory: one for the top-down computation and one for the bottom-up computation, thus reducing the space requirement from $O(l^2)$ to $O(l)$. The same idea can be applied in k dimensions, reducing the space complexity from $O(l^k)$ to $O(l^{k-1})$.

3.3 Eppstein's divide-and-conquer method

While Hirschberg's algorithm is bidirectional, he also mentioned a unidirectional version of the algorithm that he attributed to Eppstein [Hirschberg, 1997], hereafter called Eppstein's divide-and-conquer method. The algorithm is best explained in two dimensions. Instead of maintaining two rows of costs, one for each direction of search, we proceed row by

row from the start row to goal row and maintain only a single row of costs. After we pass the middle row, we maintain with each node a pointer back to its ancestor on the middle row along a lowest-cost path from the initial node. When we reach the goal state, we have a pointer to a node on the middle row that is on a lowest-cost corner-to-corner path. As in Hirschberg's method, we then have two smaller path-finding problems that we solve recursively. This algorithm also has $O(l^{k-1})$ space complexity.

3.4 Bounded dynamic programming

One limitation of both these algorithms is that they evaluate every node in the grid. Bounded dynamic programming (BDP) is an extension of DP that introduces an upper bound on the solution cost. It was presented by [Spouge, 1989] as a method of finding optimal lattice paths. Given an upper bound on the total solution cost, we can prune nodes of equal or greater cost since they cannot lie on a solution path of lower cost. For each node n we compute $g(n) + h(n)$, where $g(n)$ is the cost to reach n from the initial node, and $h(n)$ is a lower bound on the cost of reaching the goal node from node n . If this sum is equal to or exceeds the upper bound, we know that n does not lie on a lower-cost solution path. When aligning sequences of equal length, an initial upper bound can be computed by directly aligning the sequences without any gaps.

3.5 Iterative upper bounds

In general, a better upper bound on the solution cost will prune more nodes. An initial bound obtained from directly aligning the sequences is usually much greater than the true cost. A better approach is to use iterative deepening [Korf, 1985]. Starting with an upper bound that is less than the optimal solution cost, the algorithm will fail to reach the goal node by pruning too many nodes. The bound is then iteratively increased, and search repeated, until the goal is reached. This strategy has been applied to bounded dynamic programming by [Ukkonen, 1985].

Normally, the upper bound is increased to the lowest cost among all nodes pruned in the current iteration. We can do better by quickly running a few shallow searches with bounds likely to be below the optimal solution cost. By regressing the depths reached on the bounds used, we can estimate the solution cost, since we know the depth of the goal node. We use this predicted bound as the starting point for traditional iterative deepening, thus eliminating many iterations with bounds that are too low. This method of using BDP with iterative upper bounds is referred to as BDP-IUB [Korf and Zhang, 2000]. The principle of iterative upper bounds can be applied in conjunction with either Hirschberg's or Eppstein's algorithm, but in practice, Eppstein's algorithm is faster and is used in the results that follow.

When using iterative deepening, no iterations of the algorithm that use an upper bound less than the optimal solution cost can reach the goal. On the other hand, using an upper bound greater than the optimal solution cost examines more nodes than necessary. Usually we don't know the solution cost in advance, and have no choice but to use iterative deepening. However, if we knew the optimal solution cost,

this would represent a perfect bound (PB). Using this optimal bound with BDP is the best case scenario in terms of performance and does not depend on any particular iterative deepening scheme. We refer to this scenario as BDP-PB, and use this algorithm below when comparing performance with DCFS.

3.6 Disadvantages of dynamic programming

While BDP-IUB is easy to implement in two dimensions, extending it to three or more dimensions is much more difficult. Other researchers have also noted this difficulty [Zhang, 2002]. While we were able to implement a BDP-IUB algorithm written explicitly for three dimensions, we were not able to extend the implementation to four or more dimensions using the same approach. In higher dimensions, we used a generalized implementation that explicitly checks for legal operators at each node. This introduces a constant factor to the time complexity of DP since processing each node takes longer than it would in an implementation tailored to a specific dimension.

Another drawback of DP is that it can be used to find lowest-cost paths in directed grids, but not in more general grids where we allow moves in all directions. The reason is that dynamic programming must know a priori which nodes are the parents of a given node. For the general shortest-path problem, we must use a best-first search algorithm, such as DCFS, which is a generalization of Eppstein's algorithm.

4 Divide-and-Conquer Frontier Search

Divide-and-conquer frontier Search (DCFS) is a recent general heuristic search algorithm [Korf, 1999]. Best-first-search algorithms, such as Dijkstra's algorithm [Dijkstra, 1959] or A* [Hart *et al.*, 1968], normally store both a Closed list of nodes which have been expanded, and an Open list of nodes which have been generated but not yet expanded. DCFS stores only the Open list, which corresponds to the frontier of the searched area. The size of the Closed list is often much larger than the size of the Open list, and DCFS reduces the space requirement of aligning k strings of length l from $O(l^k)$ to $O(l^{k-1})$ [Korf, 1999].

Since DCFS does not store the Closed list, it is necessary to prevent the generation of nodes that have already been expanded. To do this, every node stores a list of operators to its neighbors. When a node n is expanded, the operator from each neighbor n' back to n is marked as used. Therefore, when n' is expanded, used operators are not applied, and n is not regenerated.

Unlike standard best-first search, when DCFS reaches the goal node, it cannot retrace pointers to discover the solution path, since the Closed list is not saved. Alternatively, storing a path with each Open node would require space linear in the path length for each node. In order to recover the solution path, before starting the search, a hyperplane is chosen that divides the search space in half. Every node n on the Open list beyond this hyperplane stores a pointer to the node in the hyperplane that is on the optimal path to n . In two dimensions, this hyperplane is simply a row or column halfway along one of the dimensions. We choose the hyperplane to

split the longest dimension of the space. When the algorithm encounters the goal node, it has a pointer to a node on the middle plane that is on the optimal solution path. The algorithm then recurses to find an optimal path from the start node to the optimal middle node, and also from the middle node to the goal node.

DCFS is a general heuristic search algorithm. We use divide-and-conquer frontier A* search (DCFA*) with the cost function $f(n) = g(n) + h(n)$, where $g(n)$ is the cost of a path from the start node to node n , and $h(n)$ is a heuristic estimate of reaching the goal from node n .

5 Heuristic Evaluation Functions

An accurate heuristic evaluation function is important to limit the number of nodes visited during search. In two dimensions, we use the distance of a node from the corner-to-corner diagonal times the gap cost as a heuristic, since any solution path must return to the diagonal in order to arrive at the goal node. In general, the heuristic cost of a node at (x, y) is $(g_x - x) - (g_y - y)$ times the gap cost, where (g_x, g_y) are the coordinates of the goal node.

In three dimensions, we can compute a better lower bound. Since we are using the sum-of-pairs cost function, we can use the sum of the optimal pairwise alignments as a lower-bound heuristic. Obviously, the cost of an optimal pairwise alignment is always less than or equal to the cost of any other alignment of the two strings as part of a multiple alignment. For example, an optimal alignment of three sequences is:

- 1) C-ATTTACGGA-ATACGGATAT
- 2) ATACAGCCAGGGAAATTGA-A-
- 3) A-ACAGCAGGATATTCCTA-AT

The cost of this alignment is 44, since the costs of the alignment shown for the pairs [1,2], [1,3], [2,3] are 18, 13, and 13 respectively. The heuristic estimate for the cost of aligning these strings is only 43, however, since the *optimal* pairwise alignment of the pair [1,2] is only 17, as shown below.

```
CATTTACGGAATACGGATAT
ATACAGCCAGGGAAATTGAA
```

The heuristic evaluation of a particular node in the grid is a lower bound on the lowest-cost path from that node to the goal. This corresponds to the alignment of the sequence suffixes that correspond to that node. For a three-dimensional alignment, we precompute and store three two-dimensional matrices, one for each pair of strings. Each entry of each matrix contains the cost of optimally aligning the corresponding remaining suffixes of the pair of strings. To compute the overall heuristic for any node in the cube, we sum the corresponding elements from each of the three matrices.

In the general case of aligning k sequences using the sum-of-pairs cost, the alignment of any d sequences induced by the multiple alignment will cost at least as much as the optimal alignment of those d sequences. Therefore in four or more dimensions we can also include optimal alignments of three sequences in the heuristic. While the three-dimensional alignments are more expensive to compute and occupy more space than two-dimensional alignments, including them gives

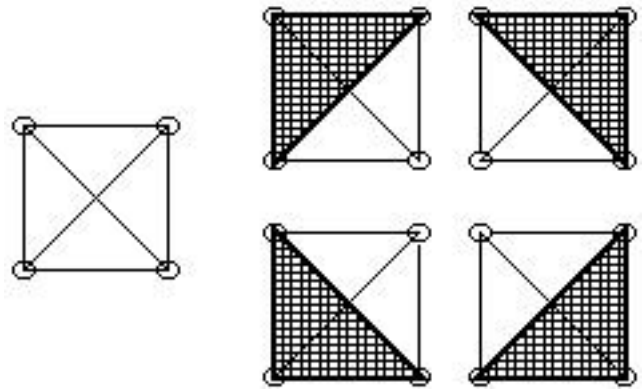


Figure 2: The five ways to partition the edges of a complete graph of four nodes, only four of which include a triangle.

a more accurate heuristic function, resulting in fewer nodes visited during the search.

To guarantee that the heuristic be a lower bound, we cannot include the cost of aligning the same pair of sequences more than once. For example, the cost of a pair of sequences that are part of a three-way alignment cannot be included in another three-way alignment, or as a pairwise alignment. Consider a complete graph with a node for each sequence, and an edge between every pair of nodes, representing that pair of sequences. In this graph a triangle represents a three-way alignment. We need to partition the edges of this graph into groups of single edges and triangles, without including any edge in more than one group, ideally in a way that maximizes the resulting heuristic value.

For example, we can partition the edges of a complete graph of four nodes (K_4) into six single edges, or one triangle and three single edges. In general, we want to include as many three-way alignments as possible, in order to make the heuristic function more accurate. There are four different ways to partition the edges of K_4 into one triplet and three pairs (see Figure 2). For the complete graph on five nodes, K_5 , there are 15 different ways to partition the nodes into two triplets and four pairs. One way of selecting which partitioning to use is to evaluate all of them and choose the one which gives the largest heuristic evaluation for the original sequences. The hope is that a larger heuristic evaluation of the start node is an indicator of larger heuristic values throughout the search. We found that evaluating all possible partitions, instead of randomly choosing one, used more time than it saved and did not yield significant savings.

We represent the three-dimensional heuristics with an octree, which stores only certain parts of a cube, along with enough information to generate the other parts of the cube if needed. [McNaughton *et al.*, 2002] Computing parts of this octree on demand saves a significant amount of space. We found that storing three-dimensional heuristics in an octree leads to an overall savings on hard problems because the time used to calculate the heuristics is offset by time saved during the search.

	$\lambda = 1$		$\lambda = 0.75$		$\lambda = 0.5$		$\lambda = 0.25$		$\lambda = 0$	
	Time (s)	Nodes	Time (s)	Nodes	Time (s)	Nodes	Time (s)	Nodes	Time (s)	Nodes
BDP-PB	59.82	77,653	59.83	77,690	60.06	220,262	78.12	21,109,562	89.79	34,838,768
DCFA*	0.06	5,601	0.07	5,601	0.27	32,971	38.20	762,740	57.90	1,286,730

Table 1: Average results over 50 generated problems of generalized versions of the two algorithms on 4 dimensional problems, length 400. The table displays average time in seconds, the average number of nodes expanded for DCFA*, and the average number of nodes visited for BDP-PB.

	$\lambda = 1$		$\lambda = 0.75$		$\lambda = 0.5$		$\lambda = 0.25$		$\lambda = 0$	
	Time (s)	Nodes	Time (s)	Nodes	Time (s)	Nodes	Time (s)	Nodes	Time (s)	Nodes
BDP-PB	15.41	33,896	15.39	33,896	15.33	39,199	16.08	351,878	17.61	1,009,293
DCFA*	0.02	2,701	0.02	2,701	0.02	5,246	0.78	97,408	1.46	110,742

Table 2: Average results over 50 generated problems of the generalized versions of the two algorithms on 5 dimensional problems, length 90. The table displays average time in seconds, the average number of nodes expanded for DCFA*, and the average number of nodes visited for BDP-PB.

6 Comparison of the Approaches

6.1 Methodology

We compared divide-and-conquer frontier A* (DCFA*) and bounded dynamic programming with perfect bounds (BDP-PB) on problems of aligning 2, 3, 4 and 5 sequences. Our goal was to determine which algorithm could solve harder problems within our space limits. The difficulty of a problem is determined by the number, length, and similarity of the sequences. We ran the tests on a 1.8 Ghz Intel Pentium 4 with 1 GB of RAM, 8 KB L1 cache, and 512 KB L2 cache. For DCFA*, we always used a fixed-size hash table of 16 million nodes which occupies 512 MB of memory. We tested both algorithms on both randomly-generated sequences and on real sequences from BALiBASE [Thomson *et al.*, 1999].

For the randomly-generated sequences, we varied sequence similarity to see how it affected the performance of the algorithms. We first generated a reference sequence uniformly from an alphabet of four letters, simulating DNA sequences, and then generated the actual sequences from the reference sequence. Each character in the actual sequences was generated independently, with a probability λ of matching the corresponding character in the reference sequence, and probability $1 - \lambda$ of being randomly generated, including the matching character. We varied λ from $0 \rightarrow 1$ in increments of 0.25. Problem instances with $\lambda = 1$ are identical sequences, which are easy to align, and were run in order to get an upper bound on solvable problem size. Problem instances with $\lambda = 0$ are the hardest problems, since the sequences are completely random. We ran both algorithms on 50 problems for each similarity, dimension, and length, and averaged the results. We found little variability between different random trials of the same experiment.

For actual sequences, we used real protein sequences from the BALiBASE database of benchmark alignments [Thomson *et al.*, 1999]. Proteins are represented by strings from an alphabet of 20 amino acids. The pairwise cost function we used charges 0 units for a match, 1 unit for a substitution, and 2 units for a gap. For multiple sequences we used the sum-of-pairs cost function. For problems in four and five dimensions, there is a choice of using two-dimensional or three-

dimensional heuristics. We ran the algorithms using both heuristics and reported the faster of the two times.

6.2 Experimental results

Our previous work [Korf and Zhang, 2000] showed that DP outperforms DCFA* in two and three dimensions, with random strings. Our new results corroborate these findings; in two dimensions, BDP-IUB aligns completely random sequences much faster than DCFA*. For example, for length 20,000 it is an average of almost 50 times faster. In three dimensions, the disparity is somewhat smaller, with DP aligning completely random sequences of length 5,500 an average of about 18 times faster. DCFA* exceeds our one hour time limit for larger problems.

In four and five dimensions, however, DCFA* was able to solve much larger problems than BDP, due to memory constraints. DP allocates memory based on problem size. In four dimensions, it was able to align sequences up to length 400, independent of the similarity of the sequences. In contrast, DCFA* was able to align four completely random sequences of length 800 in about 17 minutes.

Furthermore, for problems that both algorithms could solve, DCFA* was significantly faster than BDP. In these experiments we compared DCFA* with BDP-PB, where we use the optimal solution cost as our initial upper bound. In a realistic setting where this bound is not known a priori, BDP would fare even worse in comparison. The average results of aligning four sequences of length 400 are shown in Table 1. As the sequences become more similar, the performance advantage of DCFA* increases.

In five dimensions, BDP-PB was able to solve problems of length 90, whereas DCFA* was able to align completely random sequences of length 250 in about 5 minutes, and could align some random sequences of length 350. Both algorithms are compared on problems of aligning five sequences of length 90 in Table 2. Comparing both tables, we see the relative performance of BDP-PB on random problems decreases when moving from four to five sequences. While BDP-PB takes about 50% longer than DCFA* to align four completely random sequences of length 400, it takes about

Sequence Set	# Seq.	Min. Seq. Length	Max. Seq. Length	Sol. Cost	BDP-PB	DCFA
1uky (ref1)	4	186	200	1,213	7.78	6.45
3grs (ref1)	4	201	237	1,309	8.83	4.30
1zin (ref1)	4	206	216	813	5.15	1.97
2hsdA (ref1)	4	225	263	1,388	11.96	5.46
1ajsA (ref3)	4	365	387	2,102	56.23	27.28
1pamA (ref3)	4	404	490	2,757	*	31.78
451c (ref1)	5	70	87	727	9.38	0.57
1plc (ref1)	5	88	99	608	18.03	0.40
9rnt (ref1)	5	96	103	539	*	0.46
2mhr (ref1)	5	110	117	719	*	0.66
5ptp (ref1)	5	222	245	1,648	*	7.45
1ton (ref1)	5	224	244	1,978	*	26.69
2cba (ref1)	5	237	259	2,113	*	21.41
kinase (ref1)	5	263	276	2,440	*	41.26
glg (ref1)	5	438	486	3,665	*	88.90

Table 3: Results of the generalized versions of DCFA* and BDP-PB for 4 and 5 dimensional real protein sequence sets, with minimum and maximum sequence lengths for each set indicated. Times are in seconds. Problems unsolvable by BDP-PB, due to memory limitations, are denoted by a '*’.

12 times longer when aligning five completely random sequences of length 90.

Our results in four and five dimensions were obtained using dimension-independent implementations of both algorithms in four and five dimensions. We were not able to test performance of hand-tailored versions for four and five dimensions due to the difficulties discussed in Subsection 3.6. While the overhead of these dimension-independent implementations contributes to the disparity in speed between the two algorithms, the space requirement of BDP-PB is the more serious drawback. We believe BDP-PB runs slower in four and five dimensions because bounds checking becomes more complicated and cache performance decreases. The book-keeping necessary to maintain bounded regions becomes significant, as reflected by the nearly uniform performance for five-dimensional problems.

DCFA* can run larger problems in higher dimensions because the amount of memory required by BDP-PB depends on problem size and dimension. Much of this memory will not be used since many nodes lie outside the bounds, and will never be visited. The problem gets worse as the sequences become more similar, and as the dimensionality increases. DCFA*, on the other hand, allocates a fixed-size hash table, allowing it to align longer sequences as they become more similar. While DCFA* uses memory more efficiently than BDP-PB, both algorithms are limited by space, with the programs filling the available memory in less than an hour.

Performance of the algorithms on real sequence sets is presented in Table 3. As for randomly-generated sequences, DCFA* is faster than BDP-PB for four and five dimensional problems. A number of the real sequence sets were solvable by DCFA* but not BDP-PB because of memory constraints. In addition, there were some sequence sets not solvable by either algorithm. We used groups of 5 or fewer sequences from the first and third sets of BALiBASE (ref1 and ref3). Neither algorithm could solve problems from the other sets, which involved very large numbers of sequences.

7 Future Work

The most significant drawback of BDP in higher dimensions is its space requirements. One possible solution is to dynamically allocate space only for the nodes actually visited, allowing larger problems to fit in memory. Combining BDP with dynamic node allocation can be viewed as a hybrid algorithm between dynamic programming and best first search. The main drawback to this hybrid method is that node processing becomes significantly more complicated to implement since there is no longer a direct mapping between a node’s location in memory and its position in the grid as in traditional dynamic programming. The mapping of memory to position is an integral component of DP, and thus this hybrid algorithm is fundamentally different than standard DP. Cache performance, one of the main advantages of DP, would also decrease.

8 Conclusions

We compared two classes of algorithms to optimally solve sequence alignment problems: bounded dynamic programming and divide-and-conquer frontier search. We determined the effects of including three-dimensional alignments in the heuristic function. We then compared divide-and-conquer frontier A* (DCFA*) to bounded dynamic programming with perfect bounds (BDP-PB) on problems of aligning two, three, four and five sequences with varying degrees of similarity. We also compared performance on real protein sequence sets. Our results indicate that BDP’s inefficient use of memory does not allow it to solve problems which DCFA* can solve, and in practice is significantly slower in 4 and 5 dimensions. This surprising result contrasts with findings in lower dimensions, where BDP is the method of choice.

9 Acknowledgments

Thanks to Weixiong Zhang for his invaluable help in proof-reading a draft of this paper, supplying us with three di-

mensional BDP code, and for helpful feedback throughout. The research was supported by NSF under grant No. EIA-0113313.

References

- [Dijkstra, 1959] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.
- [Durbin *et al.*, 1998] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. *Biological sequence analysis*. Cambridge University Press, 1998. ISBN: 0–521–62971–3.
- [Hart *et al.*, 1968] P.E. Hart, N.J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Systems Science and Cybernetics*, SSC-4(2):100–107, July 1968.
- [Hirschberg, 1975] D. S. Hirschberg. A linear space algorithm for computing longest common subsequences. *Communications of the ACM*, 18:341–343, 1975.
- [Hirschberg, 1997] D. S. Hirschberg. Serial computations of Levenshtein distances. In A. Apostolico and Z. Galil, editors, *Pattern matching algorithms*, chapter 4, pages 123–141. Oxford University Press, 1997.
- [Korf and Zhang, 2000] Richard E. Korf and Weixiong Zhang. Divide-and-conquer frontier search applied to optimal sequence alignment. In *Proceedings of the 7th Conference on Artificial Intelligence (AAAI-00) and of the 12th Conference on Innovative Applications of Artificial Intelligence (IAAI-00)*, pages 910–916, Menlo Park, CA, July 30– 3 2000. AAAI Press.
- [Korf, 1985] Richard E. Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, 27(1):97–109, 1985. Reprinted in Chapter 6 of *Expert Systems, A Software Methodology for Modern Applications*, P.G. Raeth (Ed.), IEEE Computer Society Press, Washington D.C., 1990, pp. 380–389.
- [Korf, 1999] Richard E. Korf. A divide and conquer bidirectional search: First results. In Dean Thomas, editor, *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99-Vol2)*, pages 1184–1191, S.F., July 31–August 6 1999. Morgan Kaufmann Publishers.
- [McNaughton *et al.*, 2002] Matthew McNaughton, Paul Lu, Jonathan Schaeffer, and Duane Szafron. Memory-efficient A* heuristics for multiple sequence alignment. *Proceedings of the Eighteenth National Conference on Artificial Intelligence*, July-August 2002.
- [Needleman and Wunsch, 1970] S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J. Mol. Biol.*, 48:443–453, 1970.
- [Spouge, 1989] J. L. Spouge. Speeding up dynamic programming algorithms for finding optimal lattice paths. *SIAM J. Appl. Math.*, 49(5):1552–1566, October 1989.
- [Thomson *et al.*, 1999] J.D. Thomson, F. Plewniak, and O. Poch. BALiBASE: a benchmark alignment database for the evaluation of multiple alignment programs. *Bioinformatics*, 15(1):87–88, 1999.
- [Ukkonen, 1985] E. Ukkonen. Algorithms for approximate string matching. *Information and Control*, 64:100–118, 1985.
- [Waterman, 1995] M. S. Waterman. *Introduction to Computational Biology*. Chapman & Hall, 1995.
- [Zhang, 2002] Weixiong Zhang. Personal communication, August 2002.

Factored A* Search for Models over Sequences and Trees

Dan Klein

Department of Computer Science
Stanford University
Stanford, CA 94305-9040
klein@cs.stanford.edu

Christopher D. Manning

Department of Computer Science
Stanford University
Stanford, CA 94305-9040
manning@cs.stanford.edu

Abstract

We investigate the calculation of A* bounds for sequence and tree models which are the explicit intersection of a set of simpler models or can be bounded by such an intersection. We provide a natural viewpoint which unifies various instances of factored A* models for trees and sequences, some previously known and others novel, including multiple sequence alignment, weighted finite-state transducer composition, and lexicalized statistical parsing. The specific case of parsing with a product of syntactic (PCFG) and semantic (lexical dependency) components is then considered in detail. We show that this factorization gives a modular lexicalized parser which is simpler than comparably accurate non-factored models, and which allows efficient exact inference with large treebank grammars.

1 Introduction

The primary challenge when using A* search is to find heuristic functions that simultaneously are admissible, close to actual completion costs, and efficient to calculate. In this paper, we describe a family of tree and sequence models in which path costs are either defined as or bounded by a combination of simpler component models, each of which scores some projection of the full structure. In such models, we can exploit the decoupled behavior over each projection to give sharp heuristics for the combined space. While we focus on models of trees and sequences within NLP applications, the approach can be applied more generally (and already has been, in the case of biological sequence models). All the concrete cases we consider here involve search over spaces which are equivalent to dynamic programming lattices, though this detail, too, is somewhat peripheral to the basic ideas.

2 Projection Models for Graphs

The core idea of factored A* search can apply to any graph search. Assume that $G = (N, A)$ is a very large graph, with a single source node s and a single goal node g , and that we wish to use A* search to efficiently find a best path from s to g . For concreteness, assume also that the score of a path is the

sum of the scores of the arcs along the path, and that lower scores are better.¹ The particular assumption in this paper is that the arc scoring function has a special factored form. Specifically, there exists a set of projections $\{\pi_1, \dots, \pi_k\}$ of nodes (and therefore also arcs and graphs) such that for any arc (x, y) , its score σ is given by:

$$\sigma_G(x, y) = \sum_{i=1}^k \sigma_{\pi_i}(\pi_i(x), \pi_i(y))$$

Whenever the scoring function factors in this way, we have an immediate recipe for a *factored A* bound*, which we denote by h . Specifically, we can bound the shortest path in G from a node n to the goal g by the sum of the shortest paths inside each projection $\pi(G)$. Formally, if $\alpha_G(n, g)$ is the length of a shortest path from n to g in a graph G , then:

$$\alpha_G(n, g) \geq h(n, g) = \sum_{i=1}^k \alpha_{\pi_i(G)}(\pi_i(n), \pi_i(g))$$

This follows immediately from the optimality of the projected paths and the structure of the scoring function. These projections need not be mutually compatible, and therefore the bound may not be tight. Broadly speaking, the greater the degree to which each projection prefers similar paths, the better the bound will be, and the more efficient our search will be.

3 Projection Models for Sequences

For intuition, we first consider applications to sequence models before extending to the more complex case of tree models.

3.1 Example: Multiple Sequence Alignment

A situation which fits this framework perfectly is the alignment of multiple genome sequences in bioinformatics, where such multiple sequence alignments (MSAs) are standardly evaluated by sum-of-pairs scoring [Durbin *et al.*, 1998]. MSA is a generalization of the longest-common-subsequence problem, in which one is given sequences like those in figure 1a, and asked to produce pointwise alignments. Alignments of d sequences $\{s_1, \dots, s_d\}$ consist of t vertical timeslices which specify, for each sequence, either a successive element of the sequence or a gap (—), and are such that if the gaps are removed, the rows contain the original sequences. The score of a timeslice is the sum of the scores of each of the pairs

¹We will talk about minimum sums, but other semirings work as well, specifically maximum products.

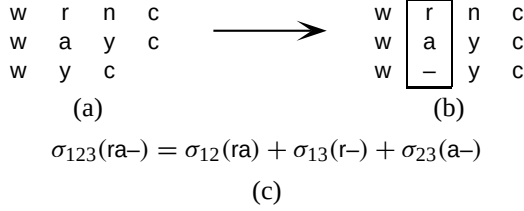


Figure 1: An example of a multiple sequence alignment. (a) The original sequences. (b) A multiple alignment with one timeslice distinguished. (c) The sum-of-pairs scoring function for that timeslice.

in that slice (where each pair of symbols is assigned some experimental goodness value).

The well-known dynamic program for calculating optimal multiple alignments involves a lattice of position nodes $n = [i_1, \dots, i_k]$ which specify an index along each sequence [Durbin *et al.*, 1998]. When each node is visited, each of its $2^k - 1$ successors p' (where each position is either incremented or not) are relaxed with the best score through p combined with the score of the timeslice that the change from p to p' represents. These nodes form a lattice of size $O(n^k)$, where n is the maximum length of the sequences. This becomes extremely inefficient as k grows.

The specific following idea has been used before ([Ikeda and Imai, 1994] is the earliest reference we could find) and it has been worked on recently ([Yoshizumi *et al.*, 2000] *inter alia*), though perhaps it has not received the attention it deserves in the bioinformatics literature. We present it here because (1) it is a good example that can be cast into our framework and (2) it gives a good starting intuition for the novel cases below. Since the score of an arc (timeslice) is a sum of pairwise goodness scores, we can define a set $\{\pi_{ab}\}$ of projections, one onto each pair of indices (i_a, i_b) . Under π_{ab} , a node $[i_1, \dots, i_k]$ will project to its a and b indices, $[i_a, i_b]$. It is easy to see that the optimal path in this projection is just the optimal 2-way alignment of the portions of sequences a and b which are to the right of the indices i_a and i_b , respectively. We can therefore bound the total completion cost of the k -way alignment from n onward with the sum of the pairwise completion costs of the 2-way alignments inside each projection.

Figure 2 shows some experimental speed-ups given by this method, compared to exhaustive search. We took several protein sequence groups from [McClure *et al.*, 1994], and, for each set, we aligned as large a subset of each group as was possible using uniform-cost search with 1GB of memory. The left four runs show the cost (in nodes visited) of aligning these subsets with both uniform-cost search and A* search. In the right four runs, we added another sequence to the subsets and solved the multiple alignment using only the A* search. The A* savings are substantial, usually providing several orders of magnitude over uniform-cost search, and many orders of magnitude over the exhaustive dynamic programming approach.² This verifies previous findings, and

²Note that the DP was never run – it would not have fit in memory – but it is easy to calculate the size of the lattice. There are also subtleties in running the uniform-cost search since the score of a timeslice can be negative (we add in a worst possible negative score).

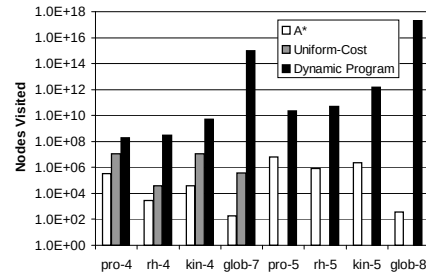


Figure 2: Effectiveness of the factored A* bound for the multiple alignment of several sequence sets. Numbers in the data set names give the number of sequences being aligned. Note the log scale: for example, on glob-7 the A* search is over a trillion times more efficient than exhaustive search.

shows that factored A* bounds can be highly effective.

One potential worry with A* search is that the cost of computing heuristics can be substantial. In this case, the $O(k^2)$ pairwise alignments could be calculated very efficiently; in our experiments this pre-search phase took up much less than 1% of the total time.

3.2 Example: Finite State Transducers

We briefly describe another potential application of factored A* search for sequence models: the intersection of weighted finite-state transducers (WFSTs).³ WFSTs are probabilistic mappings between sequences in one alphabet to sequences in another, for example a transducer might map an input of written text to an output of that text's pronunciation as a phoneme sequence. Intersections of WFSTs have been applied to various tasks in speech and NLP [Mohri, 1997], such as text-to-speech, and, most famously in the NLP literature, modeling morphophonology [Kaplan and Kay, 1994; Albro, 2000]. In these cases, each transducer constrains some small portion of the overall output sequence. The case of finding the most likely intersected output of a set of WFSTs $\{M_i\}$ for an input sequence $\mathbf{w} = {}_0w_n$ involves the following:

1. For each M_i , create the projection π_i of the full output space O onto M_i 's output space (note that this can be the identity projection).⁴
2. For each index j along $\mathbf{w}$ and each output in $\pi_i(O)$, compute optimal completion costs $\alpha_i(j, \pi_i(O))$ for M_i .
3. Use $h(i, O) = \sum_i \alpha_i(j, \pi_i(O))$ as an A* heuristic.

While transduction intersection fits cleanly into the factored framework, the primary utility of transducers lies in their composition, not their intersection [Mohri, 1997]. In this case, transducers are chained together, with the output of one serving as the input to the next. In this case, it is worth switching from talk of summed distances to talk of multiplied probabilities. Say we have two transducers, M_{IX} which gives a distribution $P(X|I)$ from sequences I to sequences X , and M_{XO} , which gives $P(O|X)$ from X to O .

³WFSTs are equivalent to HMMs which have emission weights assigned to their transitions (not states) and which may have epsilon transitions.

⁴For simplicity, we assume all history relevant to any transducer is encoded into the state space O .

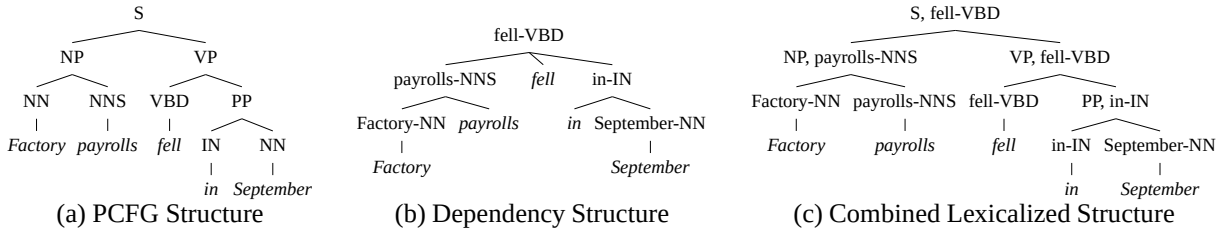


Figure 3: Three kinds of parse structures.

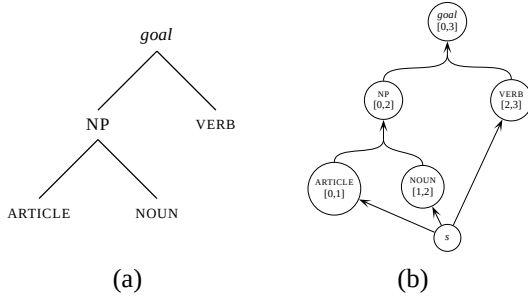


Figure 4: Two representations of a parse: (a) a tree, (b) a path in the edge hypergraph.

We then wish to answer questions about their composed behavior. For example, we might want to find the output o which maximizes $P(o|i)$ according to this model. The common Viterbi approximation is to settle for the o from the pair (o, x) which maximizes $P(o, x|i)$. This problem would fit neatly into the factored framework if the (usually false) conditional independence $P(o, x|i) = P(o|i)P(x|i)$ where true – in fact it would then be WFST intersection. However, something close to this does trivially hold: $P(o, x|i) = P(o|x, i)P(x|i)$. Given this, we can define another model $R(o|i) = \max_x P(o|x, i)$. R is not a proper probabilistic model – it might well assign probability one to every transduction – but its intersection with $P(x|i)$ does upper-bound the actual composed model. Hence these two projections provide a factored bound for a non-factored model, with the practical utility of this bound depending on how tightly $R(o|i)$ typically bounds $P(o|x, i)$.

4 Projection Models for Trees

Search in the case of trees is not over standard directed graphs, but over a certain kind of directed hypergraph in which arcs can have multiple sources (but have a single target). This is because multiple sub-trees are needed to form a larger tree.⁵ Figure 4 shows a fragment of such a hypergraph for a small parse tree (note that all the lines going to one arrowhead represent a *single* hyperarc). We don't give the full definitions of hypergraph search here (see [Gallo *et al.*, 1993] for details), but the basic idea is that one cannot traverse an arc until all its source nodes have been visited. In the parse case, for example, we cannot build a sentence node until we build both a noun phrase node and an (adjacent) verb phrase node. The nodes in this graph are identified by a grammar

⁵These directed B-hypergraphs model what has been explored as AND/OR trees in AI.

symbol, along with the region of the input it spans. The goal node is then a parse of the root symbol over the entire input. (Hyper)paths embody trees, and the score of a path is the combination of the scores of the arcs in the tree. One fine point is that, while a standard path from a source s to a goal g through a node n breaks up into two smaller paths (s to n and n to g), in the tree case there will be an inside path and an outside path, as shown in the right of figure 5. In general, then, the completion structures that represent paths to the goal (marked by α in the figure) are specified not only by a node n and goal g , but also by the original source s .

With this modification, the recipe for the factored A* bound is now:

$$\alpha_G(s, n, g) \geq h(s, n, g) = \sum_i \alpha_{\pi_i(G)}(\pi_i(s), \pi_i(n), \pi_i(g))$$

Next, we present a concrete projection model for scoring lexicalized trees, and construct an A* parser using the associated factored A* bound.

Generative models for parsing natural language typically model one of the kinds of structures shown in figure 3. While word-free syntactic configurations like those embodied by phrase structure trees (figure 3a) are good at capturing the broad linear syntax of a language [Charniak, 1996], word-to-word affinities like those embodied by lexical dependency trees (figure 3b) have been shown to be important in resolving difficult ambiguities [Hindle and Rooth, 1993]. Since both kinds of information are relevant to parsing, the trend has been to model lexicalized phrase structure trees like figure 3c.

In our current framework, it is natural to think of a lexicalized tree as a pair $L = (T, D)$ of a phrase structure tree T and a dependency tree D . In this view, generative models over lexicalized trees, of the sort standard in lexicalized PCFG parsing [Collins, 1999; Charniak, 2000], can be regarded as assigning mass $P(T, D)$ to such pairs. In the standard approach, one builds a joint model over $P(T, D)$, and, for a given word sequence ${}_0w_n$, one searches for the maximum posterior parse:

$$L^* = \max_{L=(T,D)} P(T, D|w)$$

Since $P(w)$ is a constant, one operationally searches instead for the maximizer of $P(T, D, w)$.

The naive way to do this is an $O(n^5)$ dynamic program (often called a tabular parser or chart parser) that works as follows. The core declarative object is an *edge*, such as $e = [X, i, j, h]$ which encapsulates all parses of the span ${}_i w_j$ which are labeled with grammar symbol X and are headed by word w_h ($i \leq h < j$). Edges correspond to the nodes in the

1. Extract the PCFG projection and set up the PCFG parser.
2. Use the PCFG parser to find projection scores $\alpha_{\text{PCFG}}(s, e, g)$ for each edge.
3. Extract the dependency projection and set up the dependency parser.
4. Use the dependency parser to find projection scores $\alpha_{\text{DEP}}(s, e, g)$ for each edge.
5. Combine PCFG and dependency projections into the full model.
6. Form the factored A* estimate $h(s, e, g) = \alpha_{\text{PCFG}}(s, e, g) + \alpha_{\text{DEP}}(s, e, g)$
7. Use the combined parser, with $h(s, e, g)$ as an A* estimate of $\alpha(s, e, g)$

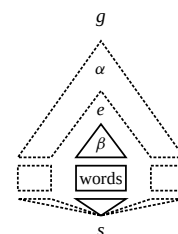


Figure 5: The top-level algorithm (left) and an illustration of how paths decompose in the parsing hypergraph (right).

parsing hypergraph. Two edges $[X, i, j, h]$ and $[Y, j, k, h']$ can be combined whenever they are contiguous (the right one starts where the left one ends) and the grammar permits the combination. For example, if there were a rewrite $Z[w_h] \rightarrow X[w_h]Y[w_{h'}]$, those two edges would combine to form $[Z, i, k, h]$, and that combination would be scored by some joint model over the word and symbol configuration: $P(XY, h'|Z, h)$.⁶ These weighted combinations are the arcs in the hypergraph.

A natural projection of a lexicalized tree L is onto its components T and D (though, to our knowledge, this projection has not been exploited previously). In this case, the score for the combination above would be $P(XY, h'|Z, h) = P(XY|Z)P(h'|h)$.⁷

This kind of projected model offers two primary benefits. First, since we are building component models over much simpler projections, they can be designed, engineered, and tested modularly, and easily. To underscore this point, we built three PCFG models of $P(T)$ and two lexical dependency models of $P(T)$. In section 4.2, we discuss the accuracy of these models, both alone and in combination.

Second, our A* heuristic will be loose only to the degree that the two models prefer different structures. Therefore, the combined search only needs to figure out how to optimally reconcile these differences, not explore the entire space of legal structures. Figure 6 shows the amount of work done in the uniform-cost case versus the A* case. Clearly, the uniform-cost version of the parser is dramatically less efficient; by sentence length 15 it extracts over 800K edges, while even at length 40 the A* heuristics are so effective that only around 2K edges are extracted. At length 10, the average number is less than 80, and the fraction of edges not suppressed is better than 1/10K (and it improves as sentence

length increases).⁸ The A* estimates were so effective that even with our object-heavy Java implementation of the combined parser, total parse time was dominated by the initial, array-based PCFG phase (see figure 6b).⁹

4.1 Specific Projection Models for Parsing

To test our factored parser, we built several component models, which were intended to show the modularity of the approach. We merely sketch the individual models here; more details can be found in [Klein and Manning, 2003]. For $P(T)$, we built successively more accurate PCFGs. The simplest, PCFG-BASIC, used the raw treebank grammar, with nonterminals and rewrites taken directly from the training trees [Charniak, 1996]. In this model, nodes rewrite atomically, in a top-down manner, in only the ways observed in the training data. For improved models of $P(T)$, tree nodes' labels were annotated with various contextual markers. In PCFG-PA, each node was marked with its parent's label as in [Johnson, 1998]. It is now well known that such annotation improves the accuracy of PCFG parsing by weakening the PCFG independence assumptions. For example, the NP in figure 3a would actually have been labeled NP'S. Since the counts were not fragmented by head word or head tag, we were able to directly use the MLE parameters, without smoothing.¹⁰ The best PCFG model, PCFG-LING, involved selective parent splitting, order-2 rule markovization (similar to [Collins, 1999; Charniak, 2000]), and linguistically-derived feature splits.

⁸Note that the uniform-cost parser does enough work to exploit the shared structure of the dynamic program, and therefore edge counts appear to grow polynomially. However, the A* parser does so little work that there is minimal structure-sharing. Its edge counts therefore appear to grow *exponentially* over these sentence lengths, just like a non-dynamic-programming parser's would. With much longer sentences, or a less efficient estimate, the polynomial behavior would reappear.

⁹There are other ways of speeding up lexicalized parsing without sacrificing search optimality. Eisner and Satta [Eisner and Satta, 1999] propose a clever $O(n^4)$ modification which separates this process into two steps by introducing an intermediate object. However, even the $O(n^4)$ formulation is impractical for exhaustive parsing with broad-coverage, lexicalized treebank grammars. The essential reason is that the non-terminal set is just too large. We did implement a version of this parser using their $O(n^4)$ formulation, but, because of the effectiveness of the A* estimate, it was only marginally faster; as figure 6b shows, the combined search time is very small.

¹⁰This is not to say that smoothing would not improve performance, but to underscore how the factored model encounters less sparsity problems than a joint model.

⁶Most models, including ours, will also mention distance; we ignore this for now.

⁷As a probabilistic model, this formulation is mass deficient, assigning mass to pairs which are incompatible, either because they do not generate the same terminal string or do not embody compatible bracketings. Therefore, the total mass assigned to valid structures will be less than one. We could imagine fixing this by renormalizing. In particular, this situation fits into the product-of-experts framework [Hinton, 2000], with one semantic expert and one syntactic expert that must agree on a single structure. However, since we are presently only interested in finding most-likely parses, no global renormalization constants need to be calculated. In any case, the question of mass deficiency impacts only parameter estimation, not inference, which is our focus here.

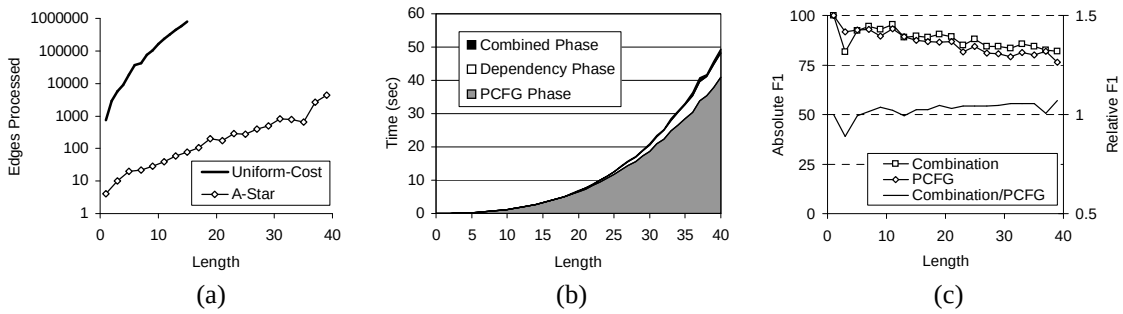


Figure 6: (a) A* effectiveness measured by edges expanded, (b) time spent on each phase, and (c) relative F_1 , all as sentence length increases.

PCFG Model	Precision	Recall	F_1	Exact Match
PCFG-BASIC	75.3	70.2	72.7	11.0
PCFG-PA	78.4	76.9	77.7	18.5
PCFG-LING	83.7	82.1	82.9	25.7

(a) PCFG Models Alone

Dependency Model	Dependency Acc
DEP-BASIC	76.3
DEP-VAL	85.0

(b) Dependency Models Alone

Figure 7: Performance of the projection models alone.

PCFG Model	Dependency Model	Prec	Rec	F_1	Exact	DepAcc
PCFG-BASIC	DEP-BASIC	80.1	78.2	79.1	16.7	87.2
PCFG-BASIC	DEP-VAL	82.5	81.5	82.0	17.7	89.2
PCFG-PA	DEP-BASIC	82.1	82.2	82.1	23.7	88.0
PCFG-PA	DEP-VAL	84.0	85.0	84.5	24.8	89.7
PCFG-LING	DEP-BASIC	85.4	84.8	85.1	30.4	90.3
PCFG-LING	DEP-VAL	86.6	86.8	86.7	32.1	91.0

Figure 8: The combined model, with various projection models.

Models of $P(D)$ were lexical dependency models, which deal with part-of-speech tagged words: pairs $\langle w, t \rangle$. First the head $\langle w_h, t_h \rangle$ of a constituent is generated, then successive right dependents $\langle w_d, t_d \rangle$ until a STOP token $\diamond$ is generated, then successive left dependents until $\diamond$ is generated again. For example, in figure 3, first we choose *fell*-VBD as the head of the sentence. Then, we generate *in*-IN to the right, which then generates *September*-NN to the right, which generates $\diamond$ on both sides. We then return to *in*-IN, generate $\diamond$ to the right, and so on. The dependency models required smoothing, as the word-word dependency data is very sparse. In our basic model, DEP-BASIC, we generate a dependent conditioned on the head and direction, requiring a model of $P(w_d, t_d | w_h, t_h, dir)$. This was estimated using a back-off model which interpolated the sparse bilexical counts with the denser but less specific counts given by ignoring the head word or by first generating the dependent tag and then generating the dependent word given only the dependent tag. The interpolation parameters were estimated on held-out data. The resulting model can thus capture classical bilexical selection, such as the affinity between *payrolls* and *fell*, as well as monolexical preferences, such as the tendency for *of* to modify nouns. In the enhanced dependency model, DEP-VAL, we condition not only on direction, but also on distance and valence. Note that this is (intentionally) very similar to the generative model of [Collins, 1999] in broad structure, but substantially less complex.

4.2 Parsing Performance

In this section, we describe our various projection models and test their empirical performance. There are two ways to measure the accuracy of the parses produced by our system. First, the phrase structure of the PCFG and the phrase structure projection of the combination parsers can be compared to the treebank parses. The parsing measures standardly used for this task are labeled precision and recall.¹¹ We also report F_1 , the harmonic mean of these two quantities. Second, for the dependency and combination parsers, we can score the dependency structures. A dependency structure D is viewed as a set of head-dependent pairs $\langle h, d \rangle$, with an extra dependency $\langle root, x \rangle$ where *root* is a special symbol and *x* is the head of the sentence. Although the dependency model generates part-of-speech tags as well, these are ignored for dependency accuracy. Punctuation is not scored. Since all dependency structures over n non-punctuation terminals contain n dependencies ($n - 1$ plus the root dependency), we report only accuracy, which is identical to both precision and recall. It should be stressed that the “correct” dependency structures, though generally correct, are generated from the PCFG structures by linguistically motivated, but automatic, and only heuristic rules.

Figure 7 shows the relevant scores for the various PCFG and dependency parsers alone. The valence model increases the dependency model’s accuracy from 76.3% to 85.0%, and each successive enhancement improves the F_1 of the PCFG models, from 72.7% to 77.7% to 82.9%. The combination parser’s performance is given in figure 8. As each individual model is improved, the combination F_1 is also improved, from 79.1% with the pair of basic models to 86.7% with the pair of top models. The dependency accuracy also goes up: from 87.2% to 91.0%. Note, however, that even the pair of basic models has a combined dependency accuracy higher than the enhanced dependency model alone, and the top three have combined F_1 better than the best PCFG model alone. For the top pair, figure 6c illustrates the relative F_1 of the combination parser to the PCFG component alone, showing the unsurprising trend that the addition of the dependency model helps

¹¹A tree T is viewed as a set of constituents $c(T)$. Constituents in the correct and the proposed tree must have the same start, end, and label to be considered identical. For this measure, the lexical heads of nodes are irrelevant. The actual measures used are detailed in [Magerman, 1995], and involve minor normalizations like the removal of punctuation in the comparison.

more for longer sentences, which, on average, contain more attachment ambiguity. The top F_1 of 86.7% is greater than that of the lexicalized parsers presented in [Magerman, 1995; Collins, 1996], but less than that of the newer, more complex, parsers presented in [Charniak, 2000; Collins, 1999], which reach as high as 90.1% F_1 . However, it is worth pointing out that these higher-accuracy parsers incorporate many finely wrought enhancements which could presumably be applied to benefit our individual models.¹²

4.3 Factored Bounds for Non-Projection Models

Arbitrary tree models will not be factored projection models. For example, while our parsing model was expressly designed so that $P(T, D) = P(T)P(D)$, to our knowledge no other model over lexicalized trees with this decomposition has been proposed. Nonetheless, non-factored models can still have factored bounds. Given any model $P(A, B)$, we can imagine bounds $R(A)$ and $R(B)$ that obey:

$$\forall A, B : P(A, B) < R(A)R(B)$$

Trivially, $R(A) = R(B) = 1$ will do. To get a non-trivial bound, consider a joint (local) model $P(XY, h'|Z, h)$ of lexicalized tree rewrites. Early lexicalized parsing work [Charniak, 1997] used models of exactly this form. We can use the chain rule to write $P(XY, h'|Z, h) = P(XY|Z, h)P(h'|XY, Z, h)$. Then, we can form $R(XY|Z) = \max_h P(XY|Z, h)$ and $R(h'|h) = \max_{XY, Z} P(h'|XY, Z, h)$. This technique allows one to use factored A* search for non-factored models, though one might reasonably expect such bounds to be much less sharp for non-factored models than for factored models. A particular application of this method for future work would be the exact parsing of the models in [Charniak, 1996; Collins, 1999; Charniak, 2000], as the details of their estimation suggest that their word dependency and phrase structure aspects would be approximately factorizable.

5 Conclusion

Not all models will factor, nor will all models which factor necessarily have tight factored bounds (for example MSA with many sequences or parsing if the component models do not prefer similar structures). However, when we can design factored models or find good factored bounds, the method of factored A* search has proven very effective. For the MSA problem, A* methods allow exact alignment of up to 9 protein sequences (though 5–6 is more typical) of length 100–300, when even three-way exhaustive alignment can easily exhaust memory. For the parsing problem, we have presented here the first optimal lexicalized parser which can exactly parse sentences of reasonable length using large real-world Penn Treebank grammars. The projected models can be designed and improved modularly, with improvements to each model raising the combined accuracy. Finally, we hope that this framework can be profitably used on the other sequence

models we outlined, and on any large space which can naturally be viewed as a composition of (possibly overlapping) projections.

Acknowledgements

We would like to thank Lillian Lee, Fernando Pereira, and Dan Melamed for advice and discussion about this work.

References

- [Albro, 2000] Daniel M. Albro. Taking primitive optimality theory beyond the finite state. In *Proceedings of the Special Interest Group in Computational Phonology*, 2000.
- [Charniak, 1996] Eugene Charniak. Tree-bank grammars. In *AAAI 13*, pages 1031–1036, 1996.
- [Charniak, 1997] Eugene Charniak. Statistical parsing with a context-free grammar and word statistics. In *AAAI 14*, pages 598–603, 1997.
- [Charniak, 2000] Eugene Charniak. A maximum-entropy-inspired parser. In *NAACL 1*, pages 132–139, 2000.
- [Collins, 1996] Michael John Collins. A new statistical parser based on bigram lexical dependencies. In *ACL 34*, pages 184–191, 1996.
- [Collins, 1999] Michael Collins. *Head-Driven Statistical Models for Natural Language Parsing*. PhD thesis, Univ. of Pennsylvania, 1999.
- [Durbin et al., 1998] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, 1998.
- [Eisner and Satta, 1999] Jason Eisner and Giorgio Satta. Efficient parsing for bilexical context-free grammars and head-automaton grammars. In *ACL 37*, pages 457–464, 1999.
- [Gallo et al., 1993] G. Gallo, G. Longo, S. Pallottino, and Sang Nguyen. Directed hypergraphs and applications. *Discrete Applied Mathematics*, 42:177–201, 1993.
- [Hindle and Rooth, 1993] Donald Hindle and Mats Rooth. Structural ambiguity and lexical relations. *Computational Linguistics*, 19(1):103–120, 1993.
- [Hinton, 2000] Geoffrey E. Hinton. Training products of experts by minimizing contrastive divergence. Technical Report GCNU TR 2000-004, GCNU, University College London, 2000.
- [Ikeda and Imai, 1994] T. Ikeda and T. Imai. Fast A* algorithms for multiple sequence alignment. In *Genome Informatics Workshop V*, pages 90–99, 1994.
- [Johnson, 1998] Mark Johnson. PCFG models of linguistic tree representations. *Computational Linguistics*, 24:613–632, 1998.
- [Kaplan and Kay, 1994] Ron Kaplan and Martin Kay. Regular model of phonological rule systems. *Computational Linguistics*, 20:331–378, 1994.
- [Klein and Manning, 2003] Dan Klein and Christopher D. Manning. Fast exact inference with a factored model for natural language parsing. In *NIPS*, volume 15. MIT Press, 2003.
- [Magerman, 1995] David M. Magerman. Statistical decision-tree models for parsing. In *ACL 33*, pages 276–283, 1995.
- [McClure et al., 1994] M.A. McClure, T.K. Vasi, and W.M. Fitch. Comparative analysis of multiple protein-sequence alignment methods. *Molecular Biology and Evolution*, 11:571–592, 1994.
- [Mohri, 1997] Mehryar Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(4):269–311, 1997.
- [Yoshizumi et al., 2000] Takayuki Yoshizumi, Teruhisa Miura, and Toru Ishida. A* with partial expansion for large branching factor problems. In *AAAI/IAAI*, pages 923–929, 2000.

¹²For example, the dependency distance function of [Collins, 1999] registers punctuation and verb counts, and both smooth the PCFG production probabilities.

An Improved Algorithm for Optimal Bin Packing

Richard E. Korf

Computer Science Department
University of California, Los Angeles
Los Angeles, CA 90095
korf@cs.ucla.edu

Abstract

Given a set of numbers, and a set of bins of fixed capacity, the NP-complete problem of bin packing is to find the minimum number of bins needed to contain the numbers, such that the sum of the numbers assigned to each bin does not exceed the bin capacity. We present two improvements to our previous bin-completion algorithm. The first speeds up the constant factor per node generation, and the second prunes redundant parts of the search tree. The resulting algorithm appears to be asymptotically faster than our original algorithm. On problems with 90 elements, it runs over 14 times faster. Furthermore, the ratios of node generations and running times both increase with increasing problem size.

1 Introduction and Overview

Given a set of numbers, and a fixed bin capacity, the bin-packing problem is to assign each number to a bin so that the sum of the numbers assigned to each bin does not exceed the bin capacity. An optimal solution uses the fewest number of bins. For example, given the set of numbers 6, 12, 15, 40, 43, 82, and a bin capacity of 100, we can assign 6, 12, and 82 to one bin, and 15, 40, and 43 to another, for a total of two bins. This is an optimal solution to this instance, since the sum of all the numbers, 198, is greater than 100, and hence at least two bins are required. An example application is given a set of orders for wire of varying lengths, and a standard length in which it is manufactured, how to cut up the minimum number of standard lengths to fill the orders.

Bin packing was one of the earliest problems shown to be NP-complete[Garey & Johnson, 1979]. The vast majority of the literature on this problem concerns polynomial-time approximation algorithms, such as first-fit decreasing (FFD) and best-fit decreasing (BFD), and the quality of the solutions they compute. First-fit decreasing sorts the numbers in decreasing order, orders the bins, and assigns each number in turn to the *first* bin in which it fits. Best-fit decreasing sorts the numbers in decreasing order and then assigns each number in

turn to the *fullest bin* in which it fits. First-fit decreasing requires three bins to pack the set of numbers above, while best-fit decreasing packs them into two bins. Both algorithms run in $O(n \log n)$ time.

In this paper we are concerned with finding optimal solutions, for several reasons. In applications with small numbers of bins, even one extra bin is relatively expensive. In addition, being able to find optimal solutions to problem instances allows us to more accurately gauge the quality of approximation algorithms. Furthermore, an anytime algorithm for finding optimal solutions, such as that presented in this paper, can make use of any additional time available to find better solutions than those returned by polynomial algorithms. Finally, optimal bin packing is computationally challenging, and may lead to insights applicable to other problems.

First we review previous algorithms for optimal bin packing, including our bin-completion algorithm[Korf, 2002]. We then describe two improvements to bin completion. The first is an algorithm for generating undominated bin completions more efficiently, reducing the constant time per node generation. Next we describe a method to reduce the branching factor of the search, by eliminating branch points that are dominated by previous branches. In experiments on millions of problem instances with uniformly distributed random numbers, our new algorithm appears to be asymptotically faster than our original bin-completion algorithm. This is based on the observation that both the ratio of nodes generated by the two algorithms, and also their running times, grow with increasing problem size. On problems of size 90, our new algorithm runs over 14 times faster than our original bin-completion algorithm. We also report mixed results on a common set of benchmark problems.

2 Previous Work

2.1 Martello and Toth

A well-known algorithm for optimal bin packing [Martello & Toth, 1990a; 1990b] is based on depth-first branch-and-bound. The numbers are first sorted, and are considered from largest to smallest. It first computes an approximate solution as an initial upper bound, using the best solution among first-fit, best-fit, and worst-fit

decreasing. Then, for each number, the algorithm places it in each partially-filled bin that it fits into, or in an empty bin. Thus, the algorithm branches on the different bins that a number can be placed in. It also uses a lower-bound function to prune the search.

2.2 More Recent OR Approaches

An anonymous reviewer pointed out some more recent references on optimal bin packing from the operations research literature, based on integer programming formulations, with linear-programming relaxations. [Valerio de Carvalho, 1999; Scholl, Klein, & Jurgens, 1997; Vanderbeck, 1999; DeGraeve & Schrage, 1999; Vance *et al.*, 1994; Vance, 1998] While we have not yet fully digested this work, we report below some comparison results on a standard set of benchmarks.

2.3 Bin Completion

In [Korf, 2002], we described our *bin completion* algorithm, which uses a branching structure different from that of Martello and Toth. A *feasible set* is a set of numbers whose sum does not exceed the bin capacity. Initially, we sort the numbers in decreasing order of size. We then generate feasible sets that include the largest number. If there is more than one such set, the search may branch at that point. Each node of the search tree, except the root node, represents a complete assignment of numbers to a particular bin. The children of the root represent different ways of completing the bin containing the largest number. The nodes at the next level represent different feasible sets that include the largest remaining number, etc. The depth of any branch of the tree is the number of bins in the corresponding solution.

Bin completion is also a branch-and-bound algorithm. It starts with the best-fit decreasing solution as an upper bound, and applies a lower-bound heuristic function to prune the search. Rather than assigning numbers one at a time to bins, it branches on the different feasible sets that can be used to complete each bin. Bin completion appears asymptotically faster than the Martello and Toth algorithm, and outperforms it by a factor of a thousand on problems of size 60. The key property that makes it more efficient is a dominance condition on the feasible completions of a bin that allows us to only consider a small subset of them.

2.4 Set Dominance

Some sets of elements assigned to a bin cannot lead to solutions that are any better than those achievable by assigning other sets of elements to the same bin. We begin with some simple examples of these dominance relations, and then consider the general formulation.

First, consider two elements x and y whose sum is exactly the bin capacity c . Assume that in one solution, x and y are in different bins. In that case, we can swap y with all other elements in the bin containing x , without increasing the number of bins. This gives us an equally good solution with x and y in the same bin. Thus, given a problem with two values x and y such that $x + y = c$,

we can always put x and y in the same bin, resulting in a smaller problem [Gent, 1998]. Unfortunately, this does not extend to three or more elements that sum to exactly the bin capacity.

As another example, consider an element x such that any two remaining elements added to x will exceed c . In other words, at most one additional element can be added to the bin containing x . Let y be the largest remaining element such that $x + y \leq c$. Then, we can place y in the same bin as x without sacrificing solution quality. The reason is that if we placed any other single element z with x , then we could swap y with z , since $z \leq y$ and $x + y \leq c$.

As a final example, again assume that y is the largest remaining element that can be added to x such that $x + y \leq c$, and that y equals or exceeds the sum of *any* set of remaining elements that can be added to x without exceeding c . In that case, we can again put x and y in the same bin, without sacrificing solution quality. The reason is that any other set of elements that were placed in the same bin as x could be swapped with y without increasing the number of bins.

To illustrate the general form of this dominance relation, let A and B be two feasible sets. If the elements in B can be partitioned into subsets, and the subsets can be matched to the elements of A such that the sum of the elements in each subset doesn't exceed the corresponding element of A , then set A *dominates* set B . In other words, if the elements of B can be packed into bins whose capacities are the elements of A , then set A dominates set B . For example, let $A = \{20, 30, 40\}$ and let $B = \{5, 10, 10, 15, 15, 25\}$. Partition B into the subsets $\{5, 10\}$, $\{25\}$, and $\{10, 15, 15\}$. Since $5 + 10 \leq 20$, $25 \leq 30$, and $10 + 15 + 15 \leq 40$, set A dominates set B .

Given all the feasible sets that contain a common element x , only the undominated sets need be considered for assignment to the bin containing x . The reason is that if we complete the bin containing x with a dominated set, then we could swap each subset of numbers in the dominated set with the corresponding element of the dominating set, and get another solution without increasing the total number of bins.

Martello and Toth use this dominance relation to some extent. In particular, they take each element x , starting with the largest element, and check if there is a *single* completion of one or two more elements that dominates all feasible sets containing x . If so, they place x with those elements in the same bin, and apply the reduction to the remaining subproblem. They also use dominance relations to prune some element placements as well.

Our bin-completion algorithm, however, makes much greater use of this dominance condition. In particular, when branching on the completion of any bin, it only considers undominated completions.

3 Finding Undominated Completions

The first contribution of this paper is a faster algorithm to generate all the undominated completions of a bin.

3.1 Previous Algorithm

The simplest algorithm is to generate all feasible sets that include a particular element, and then test each pair of sets for dominance. Our original implementation improved on this by only generating a subset of all feasible completions, and testing for pairwise dominance only among those. In particular, if only one additional number can be added to a bin, only the largest such number is added. If only two additional numbers can be added to a bin, the set of two-element undominated completions can be found in linear time, as follows.

Each undominated two-element completion must have a sum greater than the largest feasible single number, and less than or equal to the remaining capacity of the bin. The remaining values are kept in sorted order, with two pointers, one initially assigned to the largest value and the other to the smallest value. If the sum of these two numbers exceeds the remaining capacity, the pointer to the larger number is moved to the next smaller number. If the sum of the two numbers is less than or equal to the largest single feasible number, the pointer to the smaller number is moved to the next larger number. If the sum of the two numbers is within this range, the pointer to the smaller number is increased to the largest number for which the sum of the two is still in range, and these two values form an undominated two-element completion. Then the pointer to the larger number is moved to the next smaller number, and the pointer to the smaller number is moved to the next larger number. This process continues until the two pointers meet.

If the sum of the numbers in two feasible sets are unequal, only the one with the larger sum can dominate the other. If two sets have the same sum, only the one with the smaller cardinality can dominate the other. Once a subset of the feasible completions is found, each pair was tested to see if either dominates the other. This was done by trying to pack the numbers of the potential dominated set into bins whose capacities were the numbers of the potential dominating set. These small bin-packing problems were solved by brute-force search.

There are two drawbacks to this approach. The first is the time to generate and then test the dominated feasible sets. The second is the memory needed to store the dominated feasible sets before they are pruned.

3.2 Generating Undominated Sets Faster

Ideally, we would like to generate all and only undominated bin completions, without pairwise testing of feasible sets for dominance. We describe such an algorithm in three stages: 1) how to generate all subsets of a universe, 2) how to generate only feasible subsets, and 3) how to generate only undominated feasible subsets.

The easiest way to generate all 2^n subsets of n elements is to recursively traverse a binary tree, where each node represents a collection of subsets. The root node represents the entire power set, and the leaf nodes represent individual subsets. Each interior level of the tree corresponds to a different element. At each node, the left branch includes the corresponding element in all

subsets below it, and the right branch excludes the same element from all subsets below it. The tree is traversed depth-first using only linear memory.

To generate feasible completions of a bin containing a number x , we add an upper bound on the sum to each recursive call, which is initially the residual capacity, or the bin capacity minus x . We sort the remaining unassigned numbers in decreasing order of size. We only include numbers that are less than or equal to the upper bound. When we include a number by taking the left branch from a node, we subtract it from the upper bound of all recursive calls below that branch. When the upper bound drops to zero, we prune the tree below that node, since no further numbers can be added. This generates all feasible completions of a bin containing x .

Generating undominated feasible completions requires a little more work. When we exclude a number that exceeds the upper bound, by taking the right branch from the corresponding node, nothing additional is required, since it can't be a member of any feasible set below that node. If we exclude a number that equals the upper bound, we can terminate that branch of the binary tree immediately, because any feasible subset of included elements below that node cannot sum to more than the excluded element, and any subset with a smaller sum would be dominated by the excluded element.

What happens when we exclude a number that is less than the upper bound? To prevent the excluded element from dominating any included subset below it, the sum of the numbers in any such subset must exceed the excluded element. This generates a lower bound on the sum of the elements in any included subset below this node, which is equal to the excluded element plus one, assuming that the numbers are integers. As with the upper bound, the lower bound is reduced by any subsequently included elements in the recursive calls below the corresponding nodes. The lower bound generated by an excluded element is only used if it exceeds the current lower bound on that node.

Thus, we perform a depth-first traversal of the binary tree representing all possible subsets of numbers remaining to be assigned. This traversal is pruned by the upper and lower bounds propagated down the tree as described above, and generates complete subsets at the leaf nodes. These will include all undominated feasible sets, but may include some dominated feasible sets as well.

To eliminate these, we perform an additional test on the feasible bin completions generated. The residual capacity r of a bin is the bin capacity c minus the largest number in the bin, which is common to all feasible completions of a given bin. Let t be the sum of all the numbers in a feasible set A , excluding the common largest number. The excluded numbers are all remaining numbers less than or equal to r that are not included in A . Set A will be dominated if and only if it contains any subset whose sum s is less than or equal to an excluded number x , such that replacing the subset with x will not exceed the bin capacity. This will be the case if and only if $x - s \leq r - t$. Thus, to guarantee that a feasible set

A is undominated, we check each possible subset sum s , and each excluded number x , to verify that $x - s > r - t$.

This algorithm generates feasible sets and immediately tests them for dominance, so it never stores multiple dominated sets. It tests for dominance by comparing subset sums of included elements to excluded elements, rather than comparing pairs of sets for dominance.

Pseudo-code for this algorithm is given below. The *feasible* function takes a set I of included elements, a set E of excluded elements, a set R of remaining elements, a lower bound l and an upper bound u . It generates all feasible sets of remaining elements whose sum is within the two bounds, and calls the *test* function on each. In the initial call, I and E are empty, R contains all remaining elements less than or equal to the residual capacity r of the bin, u is set to r , and l is set to the largest single element that can feasibly be added to the bin, plus one. *Test* is the test described above, and is a function of the included elements I , the excluded elements E , and the residual capacity r .

```
feasible( $I, E, R, l, u$ )
  if  $R$  is empty or  $u = 0$ , test( $I, E, r$ )
  else
    let  $x$  be the largest element of  $R$ 
    if  $x > u$ , feasible( $I, E, R \setminus \{x\}, l, u$ )
    else if  $x = u$ , feasible( $I \cup \{x\}, E, R \setminus \{x\}, l - x, u - x$ )
    else
      feasible( $I \cup \{x\}, E, R \setminus \{x\}, l - x, u - x$ )
      feasible( $I, E \cup \{x\}, R \setminus \{x\}, \max(l, x + 1), u$ )
```

To improve this algorithm, we use the same optimizations used in our original algorithm to generate feasible sets. Namely, if only one more number can be added to a bin, we only add the largest such number, and if only two more numbers can be added, we generate all undominated two-element completions in linear time. This algorithm speeds up the generation of all undominated sets, without affecting the number of bin completions considered. For that, we turn to our next contribution.

4 Pruning the Search Space

Consider a number w in a bin, with a capacity of c . Assume that two undominated feasible completions of the bin are $\{w, x, y\}$ and $\{w, z\}$. For these completions to be undominated, it must be the case that $z > x$, $z > y$, $x + y > z$, $w + x + z > c$, and $w + y + z > c$. Assume that our search explores bin completions in decreasing order of subset sum, so in this case we consider $\{w, x, y\}$ before $\{w, z\}$. Furthermore, assume that after exhausting the subproblem below the assignment $\{w, x, y\}$, and while exploring the subproblem below the assignment $\{w, z\}$, we find a solution that assigns x and y to the same bin, say $\{v, x, y\}$. Since $w + x + y \leq c$, $v + x + y \leq c$, and $z < x + y$, we could swap z with x and y , resulting in a solution with the same number of bins, but including the bin assignments $\{w, x, y\}$ and $\{v, z\}$. However, all possible solutions below the node representing the bin assignment $\{w, x, y\}$ have already been explored. Thus,

this solution is redundant, and doesn't need to be considered again. In particular, below the $\{w, z\}$ branch of the search tree, any solution that assigns x and y to the same bin will be redundant and can be pruned.

In general, given a node with more than one child, when searching the subtree of any child but the first, we don't need to consider bin assignments that assign to the same bin all the numbers used to complete the current bin in a previously-explored child node. More precisely, let $\{N_1, N_2, \dots, N_m\}$ be a set of brother nodes in the search tree, and let $\{S_1, S_2, \dots, S_m\}$ be the sets of numbers used to complete the bin in each node, excluding the first number assigned to the bin, which is common to all the brother nodes. When searching the subtree below node N_i for $i > 1$, we exclude any bin assignments that put all the numbers in S_j in the same bin, for $j < i$. Thus, no bin completion below node N_i can have as a subset the numbers in S_j , for $j < i$. By rejecting these bin assignments as redundant, the number of node generations is reduced.

4.1 Current Implementation

Our current implementation of this pruning rule propagates a list of *nogood sets* along the tree. After generating the undominated completions for a given bin, we check each one to see if it contains any current nogood sets as a subset. If it does, we ignore that bin completion.

To keep the list of nogood sets from getting too long, occupying memory to store them and time to test them against bin completions, we prune the list as follows. Whenever there is a non-empty intersection between a bin completion and a nogood set, but the nogood set is not a subset of the bin completion, we remove that nogood set from the list that is passed down to the children of that bin completion. The reason is that by including at least one but not all the numbers in the nogood set in a bin completion, we've split up the nogood set, guaranteeing that it can't be a subset of any bin completion below that node in the search tree.

This implementation could probably be improved with more sophisticated data structures for representing and manipulating sets of elements.

5 Experimental Results

We tested our algorithm on large sets of problems with uniformly-distributed high-precision numbers, and on a set of benchmark problems of relatively low precision.

5.1 Uniform High Precision Numbers

We compared our original algorithm to our new bin-completion algorithm on the same problem instances and on the same machine. Since high-precision numbers are often more difficult to pack than low-precision numbers, we used a bin capacity of one million, and random numbers uniformly distributed from one to one million. Given the enormous variation in the difficulty of individual problem instances, we ran one million instances of each problem size, which ranged from 5 to 95 numbers, in increments of 5. Table 1 shows the results.

N	Optimal	Original	Without Pruning		With Nogood Pruning		Ratios	
	Bins	Time	Nodes	Time	Nodes	Time	Nodes	Time
5	3.215	6	.064	6	.064	5	1.000	1.200
10	5.966	13	.119	13	.119	12	1.000	1.083
15	8.659	19	.362	20	.360	19	1.006	1.000
20	11.321	27	.727	28	.716	26	1.015	1.038
25	13.966	36	1.249	37	1.204	35	1.037	1.029
30	16.593	44	2.046	46	1.878	43	1.099	1.023
35	19.212	55	3.376	57	2.827	52	1.194	1.058
40	21.823	73	6.325	71	4.452	65	1.421	1.123
45	24.427	103	13.346	94	7.338	81	1.819	1.272
50	27.026	189	29.414	136	12.364	104	2.379	1.817
55	29.620	609	124.476	367	28.931	174	4.303	3.500
60	32.210	2,059	391.847	1,097	108.527	518	3.611	3.975
65	34.796	28,216	7,984.196	15,694	649.553	2,658	12.292	10.616
70	37.378	41,560	9,408.125	22,628	786.126	3,549	11.968	11.710
75	39.957	194,851	57,529.770	119,928	5,308.159	21,739	10.838	8.963
80	42.534	408,580	113,746.144	233,367	7,560.130	30,972	15.046	13.192
85	45.108	412,576	129,618.988	282,851	8,697.441	36,098	14.903	11.429
90	47.680	2,522,993			38,176.160	171,778		14.688
95	50.253				324,811.294	1,343,092		

Table 1: Experimental Results for Uniformly Distributed, High-Precision Numbers

The first column gives the problem size, which is the number of values being packed. The second column shows the average number of bins needed in the optimal solution. Since the numbers range uniformly from zero to the bin capacity, the expected value of any number is half the bin capacity, and the expected value of the sum of the numbers is the half the bin capacity times the number of values. As expected, the average minimum number of bins is slightly more than half the number of values, due to the inevitable wasted space in the bins.

The third column gives the average running time of our original bin-completion algorithm [Korf, 2002], in microseconds. This is also the total time in seconds to solve all one million problem instances. All implementations are on a 440 Megahertz Sun Ultra 10 workstation.

The next two columns, labelled “Without Pruning”, give the average node generations and running times in microseconds for our implementation of bin completion with our new method of generating undominated feasible sets, but without pruning nogood sets. While this program considers the same number of candidate solutions as our original one, the node generations differ from those reported in our earlier paper [Korf, 2002]. The reason is that we define a node as a recursive call to the search routine, and our current implementation checks terminating conditions before making a recursive call, rather than at the beginning of the search function. Our new program outperforms our original one by a factor of up to 1.84 in running time. We didn’t run it on problems of size 90 or 95 due to the time that would be required.

The next two columns, labelled “With Nogood Pruning”, give the average number of nodes generated and average running time in microseconds for our full algorithm, including nogood pruning. Comparing the num-

ber of node generations to the corresponding column without pruning shows the effect of nogood pruning.

The last two columns give performance ratios of our best program, including nogood pruning. The node ratio is the number of nodes generated without nogood pruning, divided by those generated with nogood pruning. The time ratio is the running time of our original program, divided by our current best program.

As problem size increases, nogood pruning generates increasingly fewer nodes than without pruning. On the largest problems we ran both algorithms on, the ratio of node generations is about a factor of 15. The fact that the node generation ratio increases with increasing problem size suggests that nogood pruning reduces the asymptotic time complexity of bin completion.

The ratios of the running times displays a similar trend, although the values are less than the ratios of node generations. This is due to the increased overhead of nogood pruning. On the larger problems, our new algorithm is over an order of magnitude faster than our original algorithm. Problems of size 95 take an average of only 1.343 seconds per problem to solve optimally.

Variation in Individual Problem Difficulty

There is tremendous variation in the difficulty of individual problems. For example, in 68.56% percent of the one million problems with 95 numbers, the best-fit decreasing solution uses the same number of bins as the lower bound, solving the problem without any search. Among the same problems, however, twenty instances generated more than a billion nodes, three of those generated more than ten billion nodes, and one of those generated more than a hundred billion nodes. Our program solved over 125,000 problems of size 100 in about a day, and then failed to solve the next problem in over 43 days. What

distinguishes the hard problems from the easy ones?

Among all problems of size 95, the average number of bins in the optimal solution is 50.253. Among the twenty hardest problems, however, the average optimal number of bins is only 39.85. Intuitively, this makes sense, since fewer bins means more items per bin, and hence more undominated feasible sets to consider.

On the other hand, problems with a relatively small number of bins in the optimal solution are not necessarily difficult. The reason is that with smaller numbers, approximation algorithms like best-fit decreasing are more accurate. For example, eleven problems of size 95 required 36 or fewer bins. Seven of those required no search, and the average number of nodes generated to solve all eleven was only 12,704, compared to an average of 324,811 nodes for all one million problem instances.

Thus, the hard problems tend to use fewer bins, but problems that use fewer bins are not necessarily hard, since often the lower bounds agree with the solutions returned by approximation algorithms.

5.2 Benchmark Problems

We also ran our best algorithm on eight sets of twenty benchmark problems each, from the operations research library maintained by J.E. Beasley at Imperial College, London. These problem instances were originally generated by Falkenauer [Falkenauer, 1996], and have been used by a number of other researchers [Valerio de Carvalho, 1999; Vanderbeck, 1999]. We compare our results to [Valerio de Carvalho, 1999], since he reports the most detailed results. He ran his experiments on a 120 MHz Pentium, compared to our 440 MHz Sun workstation.

Uniform Problems

The first four problem sets, called *uniform* problems, consist of numbers chosen uniformly from 20 to 100, with a bin capacity of 150. Each set contains 20 problems, which are of size 120, 250, 500, and 1000 numbers each.

On the uniform problems of size 120, our best algorithm took 2 seconds on problem 0, 3 seconds on problem 3, and solved the rest instantly, meaning in less than a second. In eleven of these problems, either the best-fit decreasing solution, or the first solution found by bin completion, matched our lower bound, requiring no search. [Valerio de Carvalho, 1999] reports an average of 4.22 seconds for these problem instances.

For the uniform problems of size 250, our algorithm solved all but three instantaneously, solved problem 15 in 318 seconds, but failed to solve problems 7 and 13 in over ten minutes each. In twelve of these problems, the first solution found by bin completion matched our lower bound. [Valerio de Carvalho, 1999] reports an average of 5.98 seconds for these problems.

On uniform problems of size 500, our algorithm solved 14 problems instantly, took 4 and 160 seconds on problems 3 and 5, respectively, but failed to solve problems 0, 6, 7, and 8 in ten minutes each. In eleven of these problems, the first solution found by bin completion matched

our lower bound. [Valerio de Carvalho, 1999] reports an average of 6.93 seconds on these instances.

For uniform problems of size 1000, our algorithm solved problems 0, 1, 5, 6, 8, 10, 11, 13 and 18 instantly, and solved problems 14 and 16 in 69 and 10 seconds, respectively. In seven of these problems, the first solution found by bin completion matched our lower bound. It failed, however, to solve the remaining 9 problems in ten CPU minutes each. [Valerio de Carvalho, 1999] reports an average of 7.45 seconds for these problems.

Triplet Problems

The other four problem sets are called *triplets*, since each bin contains exactly three elements in the optimal solution. The bin capacity is 1000, with numbers in the range 250 to 500. The first number in each bin was chosen uniformly from 380 to 490, the second was chosen from 250 to one-half the size of the first number, and the third element was chosen so that the sum of the three is exactly 1000. Thus, no space is wasted in the optimal solution.

Our algorithm solved all 20 triplets of size 60 instantly, while [Valerio de Carvalho, 1999] reports an average of 4.14 seconds on these problems.

On triplets of size 120, our algorithm solved all but four problems instantly, and required 26, 1, 3, and 2 seconds on problems 0, 4, 13, and 19, respectively, for an average time of 1.6 seconds. [Valerio de Carvalho, 1999] reports an average of 26.95 seconds on these instances.

On triplets of size 249, our algorithm solved instances 3, 4, 9, 11, and 19 instantly, required 2, 19, 206, 157, 1, 75 and 126 seconds on problems 0, 6, 7, 8, 10, 13, and 15, respectively, but failed to solve the remaining 8 problems after 10 minutes each. [Valerio de Carvalho, 1999] reports an average of 122.85 seconds on these problems.

We were unable to solve any of the triplets of size 501, with ten CPU minutes each. [Valerio de Carvalho, 1999] reports an average of 360.69 seconds on these instances.

Discussion of Benchmark Results

On the uniform problems of size 120, and on the triplets of size 60 and 120, our algorithm outperformed that of [Valerio de Carvalho, 1999] in average running time, taking into account the different clock speeds of our machines. On the remaining sets of problems, our algorithm would require a longer average running time, since it failed to solve some instances in ten minutes. On uniform problems of size 250 and 500, however, our algorithm took less time than that of [Valerio de Carvalho, 1999] on most problem instances, 17 out of 20 for size 250, and 14 out of 20 for size 500. On uniform problems of size 1000, our algorithm took less time on 9 out of 20 problems, and on triplets of size 249 our algorithm was faster on 8 out of 20 problems. Overall, our algorithm performed worse than that of [Valerio de Carvalho, 1999] on the largest problem sets, however.

One possible reason for this difference in performance is that we designed our algorithm with high-precision numbers in mind, ranging from one to a million in our experiments. With these values, there are no duplicate numbers in the same problem instance, and no pairs of

numbers that sum to exactly the bin capacity. Thus, we didn't consider optimizations that are only possible with duplicate numbers. In the uniform benchmark datasets, however, the values range from 20 to 100, and in the triplets datasets they range from 250 to 500. As a result, these problem sets contain many identical numbers.

While high-precision values are usually more difficult to pack, this is not the case with the triplet datasets considered here. In particular, if they were generated in the same way, but using high-precision values instead, the first solution found by bin completion would be optimal, since there would be only one way to fill each bin completely, and that completion would be considered first. Furthermore, it would be immediately recognized as optimal, since there is no extra space in any of the bins.

6 Conclusions

We presented two improvements to our original bin-completion algorithm. The first is an algorithm for generating all undominated bin completions directly, without testing pairs of completions for dominance. More importantly, we presented an algorithm to identify and eliminate redundant bin completions, which prunes the search space. Combining these two improvements yields an algorithm that appears to be asymptotically faster than our original, and runs over 14 times faster on problems of size 90. For numbers uniformly distributed from zero to the bin capacity, we can solve a million problems of size 95 optimally in an average of 1.343 seconds per problem instance.

There is enormous variation in individual problem difficulty, with most problems being solved instantly, but some running for days or weeks. One problem of size 100 ran for 43 days without verifying an optimal solution.

On a set of standard benchmark problems, the results were mixed. Our algorithm outperformed that of [Valerio de Carvalho, 1999] on the smaller problem instances, but did worse on the largest problem instances. These benchmarks problems contain relatively low-precision values, which may provide further opportunities for improving the performance of bin completion.

It is important to note that most of the algorithms described in this paper are anytime algorithms. In other words, they produce an approximate solution immediately, and as they continue to run they produce better solutions, until they find and eventually verify the optimal solution. Furthermore, the gap between the approximate solution and the lower bound is often only a single bin, and always known throughout the computation, allowing the user to decide how much effort to expend in trying to achieve this improvement.

7 Acknowledgments

Thanks to the anonymous reviewers, particularly for pointing out the recent work on this problem in the operations research community. This work was supported by NSF under grant No. EIA-0113313, by NASA and

JPL under contract No. 1229784, and by the State of California MICRO grant No. 01-044.

References

- [DeGraeve & Schrage, 1999] DeGraeve, Z., and Schrage, L. 1999. Optimal integer solutions to industrial cutting stock problems. *INFORMS Journal on Computing* 11(4):406–419.
- [Falkenauer, 1996] Falkenauer, E. 1996. A hybrid grouping genetic algorithm for bin packing. *Journal of Heuristics* 2:5–30.
- [Garey & Johnson, 1979] Garey, M., and Johnson, D. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco: W.H. Freeman.
- [Gent, 1998] Gent, I. 1998. Heuristic solution of open bin packing problems. *Journal of Heuristics* 3:299–304.
- [Korf, 2002] Korf, R. 2002. A new algorithm for optimal bin packing. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-02)*, 731–736. Edmonton, Alberta, Canada: AAAI Press.
- [Martello & Toth, 1990a] Martello, S., and Toth, P. 1990a. Bin-packing problem. In *Knapsack Problems: Algorithms and Computer Implementations*. Wiley. chapter 8, 221–245.
- [Martello & Toth, 1990b] Martello, S., and Toth, P. 1990b. Lower bounds and reduction procedures for the bin packing problem. *Discrete Applied Mathematics* 28:59–70.
- [Scholl, Klein, & Jurgens, 1997] Scholl, A.; Klein, R.; and Jurgens, C. 1997. Bison: A fast hybrid procedure for exactly solving the one-dimensional bin packing problem. *Computers and Operations Research* 24(7):627–645.
- [Valerio de Carvalho, 1999] Valerio de Carvalho, J. 1999. Exact solution of bin-packing problems using column generation and branch-and-bound. *Annals of Operations Research* 86:629–659.
- [Vance et al., 1994] Vance, P. H.; Barnhart, C.; Johnson, E. L.; and Nemhauser, G. L. 1994. Solving binary cutting stock problems by column generation and branch-and-bound. *Computational Optimization and Applications* 3:111–130.
- [Vance, 1998] Vance, P. H. 1998. Branch-and-price algorithms for the one-dimensional cutting stock problem. *Computational Optimization and Applications* 9:211–228.
- [Vanderbeck, 1999] Vanderbeck, F. 1999. Computational study of a column generation algorithm for bin packing and cutting stock problems. *Mathematical Programming Series A* 86:565–594.

Sparse-Memory Graph Search

Rong Zhou and Eric A. Hansen

Department of Computer Science and Engineering
Mississippi State University, Mississippi State, MS 39762
{rzhou,hansen}@cse.msstate.edu

Abstract

We describe a framework for reducing the space complexity of graph search algorithms such as A* that use Open and Closed lists to keep track of the frontier and interior nodes of the search space. We propose a sparse representation of the Closed list in which only a fraction of already expanded nodes need to be stored to perform the two functions of the Closed List – preventing duplicate search effort and allowing solution extraction. Our proposal is related to earlier work on search algorithms that do not use a Closed list at all [Korf and Zhang, 2000]. However, the approach we describe has several advantages that make it effective for a wider variety of problems.

1 Introduction

Graph search algorithms such as Dijkstra’s algorithm and A* use an Open list to store nodes on the search frontier, and a Closed list to keep track of already expanded nodes. The Closed list performs two important functions.

1. It allows the optimal solution path to be reconstructed after completion of the search by tracing pointers backwards from the goal node to the start node.
2. In graph search problems, it allows nodes that have already been reached along one path to be recognized if they are reached along another path, in order to prevent duplicate search effort.

Although both functions are important, storing all expanded nodes in a Closed list can quickly use all available memory.

If one is willing to give up the ability to recognize when the same node is reached along different paths, IDA* and RBFS can solve search problems in linear space [Korf, 1985; 1993]. Related search algorithms provide a limited ability to prevent duplicate search effort by storing a limited number of expanded nodes [Reinefeld and Marsland, 1994; Miura and Ishida, 1998]. In complex graph search problems with many duplicate paths, however, an inability to prevent *all* duplicate search effort often leads to poor performance.

Fortunately, it is not necessary to store all expanded nodes in a Closed list in order to eliminate all duplicate search effort. It is only necessary to store enough nodes to form

a “boundary” in the search graph that prevents previously closed nodes from being revisited. This is the strategy adopted by a pair of recent search algorithms [Korf, 1999; Korf and Zhang, 2000], which are related to earlier work on reducing the memory requirements of dynamic programming algorithms for sequence comparison [Hirschberg, 1975; Myers and Miller, 1988]. If not all expanded nodes are stored in a Closed list, an alternative method of solution extraction is needed – since closed nodes along an optimal solution path may no longer be in memory when the search ends. So, these algorithms preserve information about nodes that are in the middle of a solution path. After the search ends, they use this information to divide the original problem into sub-problems – employing a divide-and-conquer strategy to recursively reconstruct an optimal solution path. This approach to reducing memory requirements results in an algorithm with two distinct phases; the first phase searches from the start to the goal node, and the second phase uses a divide-and-conquer method to reconstruct an optimal solution path.

In this paper, we describe some improvements of this search strategy. Earlier search algorithms that use this strategy do not store a Closed list at all [Korf, 1999; Korf and Zhang, 2000]. Instead they include extra information in open nodes, and even insert additional nodes into the Open list, to ensure that it forms a “boundary” that prevents “leaks” back into the closed region. Instead of adding information to the Open list, our algorithm preserves some already expanded nodes in a Closed list – but only enough to create a boundary and allow efficient solution reconstruction. Much of the closed region of the search can be removed from memory. Thus, we call this a sparse representation of the Closed list, and a sparse-memory approach to graph search. We show that it results in a more flexible approach to reducing memory requirements that offers several advantages.

2 Background

The most closely related algorithms to the search algorithm we introduce in this paper are Divide-and-Conquer Bidirectional Search (DCBS) [Korf, 1999] and Divide-and-Conquer Frontier Search (DCFS) [Korf and Zhang, 2000]. Although both are general graph-search algorithms, they were developed as an approach to performing multiple sequence alignment, and the memory-saving technique they use was first used by related dynamic programming algorithms for se-

quence comparison [Hirschberg, 1975; Myers and Miller, 1988]. To provide some perspective, we begin with a brief description of the multiple sequence alignment problem. For a detailed description of this problem, we refer to our references.

2.1 Multiple sequence alignment

Alignment of multiple DNA or protein sequences is an important problem in computational biology. It provides a way to measure the similarity between sequences and detect related segments. Alignment involves inserting gaps into sequences in order to maximize the number of matching “characters.” It is well-known that this problem can be formalized as a shortest-path problem in a n -dimensional lattice, where n is the number of sequences to be aligned. The number of nodes in the lattice is l^n , where l is the average length of sequences, and thus grows polynomially with the length of sequences, and exponentially with the number of sequences. Although dynamic programming is the traditional method for solving this problem, A* has been shown to outperform it by using an admissible heuristic to limit the number of nodes in the lattice that need to be examined to find an optimal alignment [Ikeda and Imai, 1999; Lermen and Reinert, 2000]. But given the large number of nodes that usually must be examined, memory is a bottleneck, and techniques for solving this problem in reduced memory can be very helpful.

2.2 Divide-and-conquer frontier search

Both Divide-and-Conquer Bidirectional Search (DCBS) [Korf, 1999] and Divide-and-Conquer Frontier Search (DCFS) [Korf and Zhang, 2000] use the same strategy for reducing memory requirements. The difference is that DCBS uses bidirectional search and DCFS uses unidirectional search. Korf and Zhang [2000] find that unidirectional search results in better performance, and so we refer to DCFS in the rest of this paper.

DCFS reduces memory requirements by storing only the frontier nodes of the search, not the interior nodes – that is, by using an Open list but not a Closed list. Because it does not use a Closed list, DCFS must use some other method to avoid duplicate node expansions and to extract an optimal solution path at the end of the search. The problem of avoiding duplicate node expansions is particularly important in the case of multiple sequence alignment because there are combinatorially many paths from the start node to any other node. For this reason, linear-space search algorithms that do not test for duplicates, such as IDA*, have “pitiful results” (in the words of Korf [1999]), when used for multiple sequence alignment. Even bounded-memory search algorithms that detect some but not all duplicates have been reported to perform poorly on this problem [Yoshizumi *et al.*, 2000].

To avoid duplicate node expansions, DCFS uses the following techniques to prevent the search from “leaking” back into the closed region, that is, to prevent closed nodes from being re-generated. First, it stores in each node a list of *forbidden operators*. This list includes one operator for each neighbor (predecessor or successor) of this node that has already been generated. When the node is later selected for

expansion, only operators that are not among the forbidden operators are used to generate its successor nodes. This prevents the search from re-generating nodes that may have been already closed and removed from memory. However, a complication arises in the case of directed graphs – in particular, directed graphs in which a node can have predecessors that are not also potential successors. If a node is expanded and removed from memory before all of its predecessors have been generated and inserted into the Open list, it can be re-generated if one of these predecessor nodes is later expanded. To prevent this, DCFS modifies the Open list in a second way. When a node is expanded, it generates not only its successor nodes, but all of its predecessor nodes – even if the search has not yet found a path to these predecessor nodes. If there is not yet a legal path to these predecessor nodes, they are assigned an infinite f -cost to prevent them from being expanded until a legal path is found.

To make it possible to reconstruct the solution path at the end of the search, DCFS modifies nodes in a further way. In each node past the midpoint of the search, it stores all information about a node on its path that is about halfway between the start and goal node. Then, when the search is completed, DCFS knows a middle node on the optimal solution path, and can reconstruct the solution path by a divide-and-conquer approach. Using the same search algorithm, it recursively solves the subproblems of finding an optimal path from the start node to the middle node, and then from the middle node to the goal node. This recursive method of solution reconstruction is essentially the divide-and-conquer idea first proposed by Hirschberg [1975].

2.3 Limitations

Korf and Zhang [2000] show that DCFS can be very effective in “problem spaces that grow polynomially with problem size, but contain large numbers of short cycles,” including path-planning in two-dimensional grids and multiple sequence alignment, assuming a small number of sequences. They acknowledge that their search algorithm is not as effective in an “exponential problem space with a branching factor of two or more.” In that case, they explain, the Open list is much larger than the Closed list and “not storing the Closed list doesn’t save much.” In fact, the multiple sequence alignment problem is an exponential problem space (and NP-complete [Just, 2001]), when the number of sequences to be aligned is not bounded. Its branching factor is $2^n - 1$ (where n is the number of sequences), and the size of the Open list can dwarf the size of the Closed list when aligning as few as five or six sequences. This problem is well-known and has motivated development of techniques for reducing the size of the Open list when using A* to align multiple sequences. These include use of an upper bound to prune open nodes that cannot lead to an optimal solution [Ikeda and Imai, 1999], and use of partial node expansions [Yoshizumi *et al.*, 2000].

As it turns out, it is difficult to combine DCFS with techniques for reducing the size of the Open list without creating inefficiencies, or even “leaks” back into the closed region. It is possible to combine DCFS with Partial Expansion A*, a technique for reducing the size of the Open list by allowing partially expanded nodes [Yoshizumi *et al.*, 2000]. This can

be beneficial in some cases. But allowing nodes to be partially expanded has the effect of reducing the number of nodes that are closed. In turn, this reduces the memory-saving effect of DCFS by reducing the number of nodes eligible to be removed from memory, as our experimental results will show.

Another approach to reducing the size of the Open list is to prune open nodes when their f -cost is greater than an upper bound on the optimal f -cost, as in Enhanced A* [Ikeda and Imai, 1999]. Because this technique does not reduce the number of closed nodes, the idea of combining it with DCFS appears more promising. However, it introduces other difficulties. Recall that DCFS generates all predecessors of a node when it is expanded, even if it has not yet found a path to these predecessors. If nodes are pruned from the Open list, no path may ever be found to these extra nodes, and the Open list may become cluttered with useless nodes that can never be removed. In directed graphs with the special property that the set of successors of each node is disjoint from the set of predecessors, such as the search graph of the multiple sequence alignment problem, this inefficiency is the only negative effect of pruning the Open list. But in directed graphs that do not share this property, and in all undirected graphs, pruning nodes from the Open list can also result in “leaks” back into the closed region, as follows. The first time a node is generated, there is no guarantee that the best path to it has been found. So, if it is pruned using an upper bound, it may later be re-generated if a better path to it is found. But since the forbidden operators associated with the node when it was first generated were lost when it was pruned, a node that was previously closed could be re-generated.

The difficulty of combining DCFS with techniques for reducing the size of the Open list is a significant limitation. Another potential limitation worth mentioning is the overhead for storing a list of forbidden operators in each node. For problems with a small branching-factor, this overhead is slight. But for the multiple sequence alignment problem, this overhead grows as the number of sequences grows. Recall that the number of operators (i.e., the branching factor) of multiple sequence alignment is $2^n - 1$, and DCFS stores incoming as well as outgoing edges in the list of forbidden operators. Thus, the total number of edges incident to a node is $(2^n - 1) \times 2 = 2^{n+1} - 2$. Although Korf and Zhang [2000] claim that the space complexity of DCFS for multiple sequence alignment is $O(l^{n-1})$, compared to the $O(l^n)$ space complexity of A*, this claim rests on the assumption that a node takes constant storage. As n increases, the storage required for the list of forbidden operators increases exponentially. In fact, the space complexity of DCFS for multiple sequence alignment is $O(2^n l^{n-1})$, and its advantage over the $O(l^n)$ space complexity of A* disappears when $n > \log_2 l$.

3 Sparse-Memory Search Algorithm

The algorithm we describe in the rest of this paper adopts a strategy for reduced-memory search that is similar to the strategy used by DCFS. However, it implements this strategy in a simpler way that is more compatible with techniques for reducing the size of the Open list, and does not require storing lists of forbidden operators in nodes.

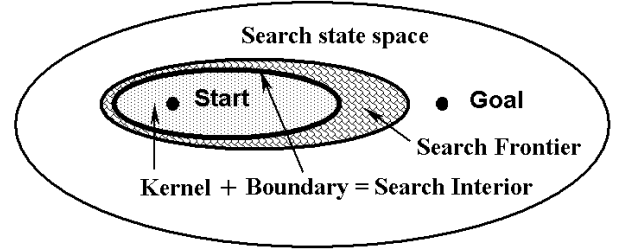


Figure 1: An illustration of the relationships among the kernel and the boundary of the search interior, the search frontier, and the entire state space.

A key difference from DCFS is that our algorithm does not entirely eliminate the Closed list. Instead, we propose a sparse representation of the Closed list that allows many (but not all) closed nodes to be removed from memory. To explain our approach, we note that the search interior (the set of closed nodes) can be partitioned into two disjoint subsets that we call the set of *boundary nodes* and the *kernel* of the search interior.

Definition 1 Let I be the set of search interior nodes, i.e., nodes whose lowest-cost paths have been found. The kernel of I , denoted $K(I)$, is defined as follows:

$$K(I) = \{k \mid k \in I, \forall p \in \text{Pred}(k) \Rightarrow p \in I\},$$

where $\text{Pred}(k)$ denotes the set of predecessor nodes of k , that is, the set of nodes that can make a transition into node k in the underlying graph, which may be directed or undirected.

Basically, the kernel is the set of nodes whose predecessor nodes are all interior (i.e., closed) nodes.

Definition 2 The set of boundary nodes of search interior I , denoted $B(I)$, is defined as the non-kernel nodes of I , that is:

$$B(I) = I \setminus K(I).$$

Another way of describing a boundary node is to say that at least one of its predecessor nodes is not an interior node, or mathematically,

$$B(I) = \{b \mid b \in I, \exists p \in \text{Pred}(b), p \notin I\}.$$

Figure 1 illustrates the relationship between the kernel and boundary of the search interior. Note that nodes in the boundary can enter the kernel, but once a node is in the kernel it remains there. The nodes in the kernel are eligible for removal from memory because they are not needed to prevent duplicate search effort.

The intuitive meaning of “boundary” can be explained as follows. Because every closed node is reachable from the start node, the set of closed nodes can be considered a “volume” in the underlying graph that encompasses the starting node. Nodes outside this volume cannot reach nodes inside the volume without passing through some node in the boundary. Thus, storing only the boundary nodes in the Closed list is as effective as storing the entire “volume,” with respect to

preventing the search from “leaking” back into the closed region. (From this perspective, one could say that DCFS adds nodes to the Open list – i.e., the search frontier – in order to ensure that the frontier forms a boundary, and it treats the Closed list as the kernel that can be removed from memory.)

In addition to keeping boundary nodes in the Closed list, our sparse-memory approach keeps some other, non-boundary nodes that allow solution reconstruction. The method of solution reconstruction is different from the one used by DCFS. Recall that DCFS stores in each node all information about a “middle” node on the path through this node to the goal. In our approach, each node in the search graph maintains a pointer to its predecessor node along the best path to this node, *or to some earlier, ancestral node along this path*.¹ In the latter case, we have a *sparse solution path*.

Definition 3 A sparse solution path is an ordered list of nodes $P = \{P[0], P[1], \dots, P[d]\}$, such that $d \geq 1$, $P[0]$ is a start node, $P[d]$ is a goal node, and $\sum_{i=0}^{d-1} \delta(P[i], P[i+1]) = \delta(\text{start}, \text{goal})$, where $\delta(u, v)$ denotes the cost of the lowest-cost path between two nodes u and v .

If a node in a sparse solution path has a pointer to an ancestral node instead of a pointer to its predecessor, we call the ancestral node a *relay node* to indicate that it skips over some nodes in an original, “dense” solution path. We also consider the start node (which has no backward pointer) a relay node. How relay nodes are created and used in solution reconstruction is explained below. Here, we simply note that relay nodes are never removed from the Closed list because they are needed for solution reconstruction. Thus, a sparse representation of the Closed list includes all boundary nodes and all relay nodes. Nodes that do not fall in either category may be removed from memory.

It is well-known that A* can be viewed as Dijkstra’s algorithm applied to a transformed graph with the same set of nodes and edges, but modified edge costs given by the equation $\hat{c}(u, v) = c(u, v) - h(u) + h(v)$, where $c(u, v)$ (or $\hat{c}(u, v)$) is the cost of edge (u, v) in the untransformed (or transformed) graph and $h(u)$ (or $h(v)$) is the cost estimate of the lowest-cost path from node u (or v) to the goal node. Therefore, we present our sparse-memory algorithm as a modification of Dijkstra’s algorithm, and note that our description applies to A* also. Our sparse-memory approach requires two assumptions. First, the transformed (or untransformed) graph cannot contain any negative-cost edges. This means the heuristic used by A* must be consistent. Without this assumption, it is impossible for our algorithm (or any heuristic search algorithm) to accurately identify the interior of the search, and DCFS makes the same assumption. Second, we assume that our algorithm knows the in-degree (in the underlying graph) of every node it visits. For comparison, DCFS assumes that it knows every predecessor (in the underlying graph) of a node it visits. Thus, our second assumption is weaker than that made by DCFS.

¹Because pointers require less memory than state information about a midpoint node, the memory required to store all pointers plus relay nodes could be less than the memory required to copy state information about the same midpoint node into many other nodes.

```

Procedure ExpandNode (Node  $u$ )
1  for each  $v \in \text{Successors}(u)$  do
2    if  $v \in \text{Open}$  then
3       $\rho(v) \leftarrow \rho(v) - 1$ 
4      if  $g(u) + c(u, v) < g(v)$  then
5         $g(v) \leftarrow g(u) + c(u, v)$ 
6         $\text{ancestor}(v) \leftarrow u$ 
7      else if  $v \in \text{Closed}$  then
8         $\rho(v) \leftarrow \rho(v) - 1$ 
9      else /* generate new node */
10        $\rho(v) \leftarrow \text{in-degree}(v) - 1$ 
11        $g(v) \leftarrow g(u) + c(u, v)$ 
12        $\text{ancestor}(v) \leftarrow u$ 
13        $\text{Open} \leftarrow \text{Open} \cup \{v\}$ 
14       if memory is full then PruneClosedList ()

```

```

Procedure PruneClosedList ()
1  for each  $v \in \text{Open} \cup \text{Closed}$  do
2    if  $\text{ancestor}(v) \in \text{Pred}(v)$  then
3       $\alpha \leftarrow \text{ancestor}(v)$ 
4      while  $\alpha \in \text{Closed}$  and  $\rho(\alpha) = 0$  do
5         $\alpha \leftarrow \text{ancestor}(\alpha)$ 
6      if  $\alpha \neq \text{ancestor}(v)$  then
7         $\text{ancestor}(v) \leftarrow \alpha$ 
8         $\rho(\alpha) \leftarrow \infty$  /* never prune relay node */
9  for each  $v \in \text{Closed}$  do
10   if  $\rho(v) = 0$  then
11      $\text{Closed} \leftarrow \text{Closed} \setminus \{v\}$ 
12   Delete  $v$ 

```

```

Algorithm SMGS (Node  $\text{start}$ , Node  $\text{goal}$ )
1   $g(\text{start}) \leftarrow 0$ 
2   $\rho(\text{start}) \leftarrow \infty$  /* never prune start node */
3   $\text{ancestor}(\text{start}) \leftarrow \text{null}$ 
4   $\text{Open} \leftarrow \{\text{start}\}$ 
5   $\text{Closed} \leftarrow \emptyset$ 
6  while  $\text{Open} \neq \emptyset$  do
7     $u \leftarrow \arg \min_u \{g(u) \mid u \in \text{Open}\}$ 
8     $\text{Open} \leftarrow \text{Open} \setminus \{u\}$ 
9     $\text{Closed} \leftarrow \text{Closed} \cup \{u\}$ 
10   if  $u$  is goal then
11      $\text{SSP} \leftarrow \text{ExtractSparseSolutionPath}(u)$ 
12      $\text{DSP} \leftarrow \emptyset$ 
13     for  $i = 1$  to  $\text{length}(\text{SSP})$  do
14       if  $\text{SSP}[i-1] \in \text{Pred}(\text{SSP}[i])$  then
15          $\text{DSP.addTail}(\text{SSP}[i])$ 
16       else
17          $\text{DSP.addTail}(\text{SMGS}(\text{SSP}[i-1], \text{SSP}[i]))$ 
18     return  $\text{DSP}$ 
19   else
20     ExpandNode ( $u$ )

```

Figure 2: Pseudocode for SMGS (Sparse-Memory Graph Search), in which procedure *ExpandNode* assumes a directed graph. See Figure 3 for the pseudocode of *ExpandNode* in undirected graphs.

3.1 Pruning the Closed list

Our search algorithm must be able to efficiently distinguish between the kernel and boundary of the search interior, in order to prune nodes from the kernel. Recall that a node is in the kernel if all of its predecessors are closed. To identify such nodes, we introduce a technique for keeping track of the number of unexpanded predecessors for each generated-and-stored node. We call this number the ρ -value of a node. It is initially set to the number of predecessors (the in-degree) of the node in the underlying graph minus one to account for the predecessor that generated it. The ρ -value is updated during node expansion. As each successor of a node is considered (some of which may already be in the Open or Closed lists), its ρ -value is decremented. (See lines 3, 8 and 10 of procedure *ExpandNode* in Figure 2.) This requires negligible time and space overhead, and, given ρ -values, kernel-membership for a node can be determined in constant time by checking whether it is a closed node with a ρ -value of zero.

An advantage of the sparse-memory approach is that it does not immediately remove closed nodes from memory, unlike DCFS. A sparse-memory version of Dijkstra's algorithm (or A*) acts exactly like Dijkstra's algorithm (or A*) until it senses that memory is about to be exhausted. Only then does it invoke procedure *PruneClosedList* in Figure 2 to recover memory. This procedure prunes nodes from the Closed list in two steps. First it updates the ancestral pointer of any boundary node whose predecessor is about to be pruned (lines 1-8). This is necessary to allow solution reconstruction and requires finding the *relay node* that is the closest boundary node along its solution path (lines 3-5), and updating its ancestral pointer accordingly (line 7). This makes this boundary node a relay node, and to prevent it from being pruned in the future, its ρ -value is set to ∞ (line 8). After this step, kernel nodes are pruned unless they are a start node or relay node created in a previous pruning step (lines 9-12). Updating the ancestral pointers of nodes (followed by pruning) creates a sparse solution path, from which a complete or "dense" solution path can be reconstructed after the search terminates.

3.2 Solution Path Reconstruction

The fact that the Closed list is not pruned unless memory is close to being exhausted means that the overhead of solution reconstruction can be avoided if memory resources are adequate. In that case, the sparse-memory algorithm acts exactly like Dijkstra's algorithm (or A*) and an optimal solution path is extracted in the conventional way.

If the Closed list has been pruned, an optimal solution path is reconstructed by invoking the Sparse-Memory Graph Search algorithm (SMGS) in Figure 2 recursively. First the sparse solution path (SSP) is extracted (line 11) in the conventional way by tracing pointers backward from the goal. Then the corresponding *dense solution path* (DSP) is reconstructed as follows. For each pair of consecutive nodes ($SSP[i-1], SSP[i]$) in the SSP (starting from the start node), the algorithm checks to see if $SSP[i-1]$ is a predecessor of $SSP[i]$ (line 14). If so, node $SSP[i]$ is added to the tail of the DSP (line 15); otherwise, the search algorithm calls itself recursively with $SSP[i-1]$ and $SSP[i]$ as the

Procedure *ExpandNode* (Node u)

```

1   $\rho(u) \leftarrow \text{degree}(u)$ 
2  for each  $v \in \text{Successors}(u)$  do
3    if  $v \in \text{Open}$  then
4      if  $g(u) + c(u, v) < g(v)$  then
5         $g(v) \leftarrow g(u) + c(u, v)$ 
6         $\text{ancestor}(v) \leftarrow u$ 
7      else if  $v \in \text{Closed}$  then
8         $\rho(v) \leftarrow \rho(v) - 1$ 
9         $\rho(u) \leftarrow \rho(u) - 1$ 
10     else /* generate new node */
11        $g(v) \leftarrow g(u) + c(u, v)$ 
12        $\text{ancestor}(v) \leftarrow u$ 
13        $\text{Open} \leftarrow \text{Open} \cup \{v\}$ 
14     if memory is full then PruneClosedList ()
```

Figure 3: Pseudocode for *ExpandNode* in undirected graphs.

new start and goal nodes, in order to get a dense solution path between the two which is added to the tail of DSP (line 17).

Another difference from DCFS that is worth noting is that DCFS divides a problem into two subproblems at each level of the recursion. The sparse-memory approach can divide a problem into two or more subproblems. The extra flexibility is possible because the sparse-memory approach uses relay nodes with ancestral pointers, whereas DCFS stores all information about a middle node in each node. When a problem is divided into more than two subproblems, the subproblems are smaller and easier to solve and solution reconstruction can be faster. In fact, relay nodes can be spaced at half intervals, one-third intervals, or any other interval, and allow a tradeoff between the sparseness of the search interior and the speed of solution reconstruction.

3.3 Pruning the Open list

We now consider how easily the sparse-memory approach to pruning the Closed list can be combined with techniques for pruning the Open list. Pruning nodes on the Open list that have an f -cost greater than an upper bound on the optimal f -cost creates difficulties for DCFS, as discussed earlier. An advantage of the sparse-memory approach is that it does not create difficulties, at least in solving problems for which the set of predecessors of a node is disjoint from the set of successors, such as the multiple sequence alignment problem. This advantage will be illustrated in our experimental results.

For other directed graphs, that is, for directed graphs in which the same node can be both predecessor and successor of another node, DCFS allows leaks back into the closed region when the Open list is pruned. The sparse-memory approach does not, and this is another advantage. Unfortunately, the sparse-memory approach has a problem in such graphs. If the best path to a node has not been found when it is first generated, it could be pruned and re-generated later, when a better path is found. Since ρ -values are erased when the node is pruned, the ρ -value of such nodes will not be decremented to zero. As a result, the sparse-memory algorithm may not recover as much memory from the Closed list. However, our experimental results suggest that this inefficiency is minor.

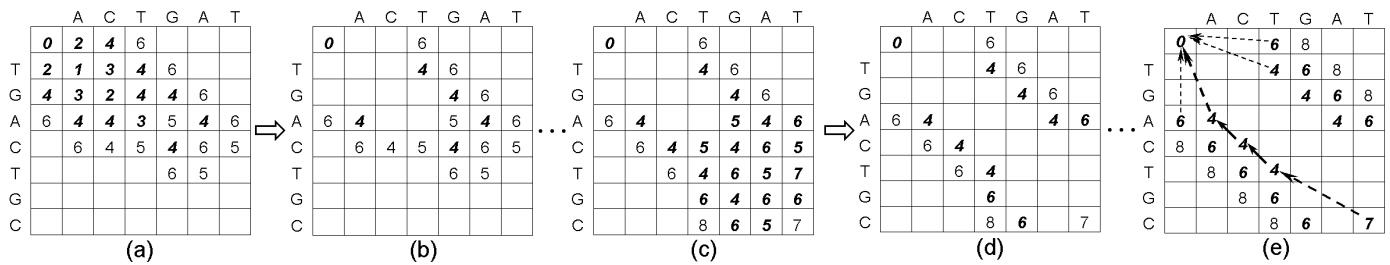


Figure 4: Example of Sparse-memory Dijkstra searching for an optimal alignment of two sequences. Panels (a) and (b) show the search space just before and after the first pruning of the Closed list. Panels (c) and (d) show the search space just before and after the second pruning. Panel (e) shows the sparse solution path at the end of the search.

In undirected graphs, we noted earlier that DCFS also allows “leaks” back into the closed region when the Open list is pruned. The sparse-memory algorithm does not, if we make a minor modification, as explained in the following.

3.4 Undirected graphs

The pseudocode in Figure 2 shows the *ExpandNode* procedure for directed graphs. Figure 3 gives the pseudocode of the *ExpandNode* procedure in undirected graphs. The difference is that in undirected graphs, ρ -values are only set and decremented for closed nodes, not open nodes. Thus, pruning open nodes cannot cause any problems by distorting ρ -values. In fact, the pseudocode in Figure 3 could be used for directed graphs also. But in directed graphs, it would require considering all predecessors of a node as well as successors during node expansion, in order to set ρ -values correct. The overhead for this would be significant. This is not a problem in undirected graphs because the set of potential successors of a node is equal to the set of potential predecessors.

3.5 Hybrid algorithms

The sparse-memory algorithm we have described is based on more than one idea. It uses nodes on the Closed list to create a “boundary” that prevents re-generation of closed nodes, instead of using the Open list as a boundary. In addition, it uses relay nodes for divide-and-conquer solution reconstruction, instead of storing in each node state information about a middle node along the search path.

Because these ideas can be considered separately, it is possible to create search algorithms that are hybrids of DCFS and the sparse-memory approach. For example, the DCFS technique of using the Open list as a boundary could be combined with relay nodes. This would create a version of DCFS that is able to divide a solution path into more than two pieces, allowing flexible (and potentially faster) solution reconstruction; use of relay nodes would also allow DCFS to delay pruning of the Closed list until memory is full. Similarly, some of the techniques used by DCFS can be considered independently. For example, forbidden operators can be used in combination with the sparse-memory approach, instead of ρ -values. One advantage of using forbidden operators is that they sometimes speed up search, since it is faster not to generate a node than to generate it and check for a duplicate on the Open or Closed list. We consider the performance of these hybrid algorithms in the computational results that follow.

4 Computational Results

We first illustrate how a sparse-memory version of Dijkstra’s algorithm works by showing how it solves a small pairwise sequence alignment problem. Then we consider the performance of sparse-memory A* on more challenging problems.

4.1 Example

Consider the problem of aligning two sequences, ACTGAT and TGAATC, using a very simple cost function: zero for a match, one unit for a substitution, and two units for a gap. The state space of the problem can be represented by a two-dimensional grid in which the columns correspond to one sequence and the rows to the other. The problem of finding an optimal alignment of the two sequences corresponds to the problem of finding an optimal path from the start node in the upper-left corner to the goal node in the lower-right corner, where horizontal and vertical moves correspond to gap insertions in one or the other sequence, and diagonal moves correspond to a substitution or match of characters.

Figure 4 shows the behavior of sparse-memory Dijkstra’s algorithm at the critical points when memory becomes full and the Closed list is pruned. It assumes that memory capacity is 30 nodes. Figure 4(a) shows the explored state space when memory is full for the first time. The number in each cell is the g -value of the corresponding state (or node). For closed nodes, the g -value is highlighted in bold italics. Among the 17 closed nodes, 11 are identified as kernel nodes and pruned. Figure 4(b) shows the result of pruning. When memory is full the second time, as shown in Figure 4(c), the Closed list is pruned again and 12 kernel nodes are removed from memory, as shown in Figure 4(d), freeing enough memory for continued search. Figure 4(e) shows the state space in memory when the goal node (the cell in the lower-right corner) is expanded. The SSP solution is shown as a chain of thick dashed arrows. The arrows drawn in thin dashed lines represent ancestor pointers that are created during pruning.

In this example, it is also possible to see how the Open list can be pruned using the sparse-memory approach. Suppose we know that 8 is an upper bound on the cost of an optimal alignment. Then all nodes with g -values greater than or equal to 8 can be pruned from the Open list, and the search progresses in the same way except for not storing the seven nodes whose g -values equal 8. Note that pruning nodes in the Open list will *not* change the boundary of the search interior.

4.2 Multiple sequence alignment

We tested Sparse-memory A* (Sparse-A*) on a series of challenging multiple sequence alignment problems, in order to compare its performance to DCFA* (the DCFS version of A*). We used a 300Mhz Sun UltraSparc II workstation with two gigabytes of RAM. Results are displayed in Table 1.

First, we considered the identical test domain used by Korf and Zhang [2000]: alignment of three random sequences of length 4000 using their simple cost function, with results averaged over 100 trials. Both DCFA* and Sparse-A* are effective in this domain, whereas A* could not solve any problem instance due to memory limitations. Versions of A* that use special techniques to recover memory by pruning the Open list [Ikeda and Imai, 1999; Yoshizumi *et al.*, 2000] were also unable to solve any of these problem instances. This is because the Closed list, not the Open list, fills most of memory in this case. One reason there are so many closed nodes is that the accuracy of the pairwise heuristic used for multiple sequence alignment depends on the similarity of the sequences. Because random sequences have only random similarities, the heuristic is weak, resulting in many node expansions and closed nodes.

For this problem set, DCFA* is more memory-efficient than Sparse-A*. However, Sparse-A* runs faster. It runs faster for long sequences like these because the solution path is correspondingly long, and solution reconstruction can be faster using relay nodes. At each recursion level, relay nodes make it possible to divide a solution path into several smaller, easier-to-solve subproblems, instead of always dividing it in half. Faster solution reconstruction comes at the expense of a small increase in memory for extra relay nodes, as can be seen by comparing the performance of DCFA* using relay nodes (a hybrid algorithm) to standard DCFA*.

Why can DCFA* be more memory-efficient than Sparse-A*? One factor is that using relay nodes to divide a solution path into more than two pieces requires extra memory for the extra relay nodes – a classic space-time tradeoff. Another factor is that use of forbidden operators allows a closed node to be pruned as soon as all its predecessors are *generated*. By contrast, use of ρ -values requires waiting until all predecessors are *closed*, before pruning the node. As a result, a hybrid algorithm that combines the sparse-memory approach with forbidden operators outperforms the sparse-memory approach alone, for this problem set. These two factors may not account for all of the difference in memory efficiency, but we currently have no other explanation.

We next compared Sparse-A* and DCFA* on real protein sequences using the PAM250 cost matrix, which is widely-used by biologists. In aligning five sequences randomly selected from a pool of low-similarity protein sequences of length 300 used in previous experiments [McNaughton *et al.*, 2002], Sparse-A* and DCFA* were again effective in solving all instances. By contrast, A* ran out of memory on most instances. For this problem set, the Enhanced A* algorithm [Ikeda and Imai, 1999], which uses an upper bound to prune nodes from the Open list, performs very well. This is because the number of open nodes is very large for this set of problems. Unlike DCFA*, the sparse-memory approach can be safely combined with this technique of pruning the

Algorithm	Statistics	3 seqs. (4,000)	5 seqs. (300)	7 seqs. (450)
DCFA*	Secs.	1,435	2,930	172
	Nodes(K)	1,953	9,997	1,200
	Mbytes	284	551	112
DCFA* + relay nodes	Secs.	712	2,863	97
	Nodes(K)	2,202	10,910	1,215
	Mbytes	293	561	114
Sparse-A*	Secs.	908	3,073	65
	Nodes(K)	4,510	17,626	1,093
	Mbytes	389	766	72
Sparse-A* + forbidden ops.	Secs.	871	3,218	101
	Nodes(K)	3,388	13,100	1,071
	Mbytes	355	673	104
Enhanced A*	Secs.		1,940	24
	Nodes(K)	can't solve	16,844	76
	Mbytes		720	28
Enhanced Sparse-A*	Secs.	917	2,785	24
	Nodes(K)	4,510	10,600	76
	Mbytes	389	465	28

Table 1: Performance comparison of DCFA*, sparse-memory A*, Enhanced A*, and hybrid algorithms. The node count is the number of stored nodes, measured in thousands, and memory includes storage of the pairwise heuristic.

Open list. This gives it an overall advantage, and Enhanced Sparse-A* is the best-performing algorithm for this problem set. (Although Enhanced Sparse-A* stores slightly more nodes than DCFA*, it uses less memory because forbidden operators makes the DCFA* nodes larger. We also emphasize the following point: although the table shows that the running time of Enhanced A* is less than the running time of Enhanced Sparse-A*, this does not mean that it is a faster algorithm. If Enhanced A* by itself can solve a problem in available memory, the sparse-memory version of Enhanced A* does not need to prune the Closed list at all, and has identical performance. But for these experiments, we adjusted the sparse-memory algorithm to minimize its memory use – since memory is the key factor we are evaluating here.)

A drawback of DCFA* is that its node size increases as the number of sequences being aligned increases. This becomes very apparent when DCFA* is used to align seven protein sequences of length around 450, randomly selected from a set of similar protein sequences used in previous experiments [Yoshizumi *et al.*, 2000]. The high similarity of these sequences gives rise to a very accurate heuristic, and makes these sequences much easier to align. Even A* can solve these problems. Although A*, Sparse-A* and DCFA* always expand the same number of nodes, DCFA* generates and stores an average of 10% more nodes in this domain because it inserts extra nodes in the Open list. A more serious problem is that the nodes created by DCFA* are two times bigger than the nodes created by the other algorithms, because they include lists of forbidden operators. As a result, DCFA* runs slower and uses more memory than any other algorithm, including A*! (In aligning ten sequences, the nodes created by DCFA* are seven times larger than the nodes created by A*,

and their relative size almost doubles with each additional sequence thereafter.) The table shows Sparse-A* is not as effective as Enhanced A*, because pruning the Open list is much more important than pruning the Closed list, for this problem set. Nevertheless, Enhanced Sparse-A* performs no worse than Enhanced A*, and this point is important. The sparse-memory approach improves performance when possible, and complements other techniques for reducing memory.

For perspective, we discuss what happens if we combine DCFA* with Partial Expansion A* [Yoshizumi *et al.*, 2000]. (In our experiments, we set the cutoff value for Partial Expansion A* to zero, to minimize memory use.) For the set of seven similar protein sequences, Partial Expansion DCFA* required an average of 29 Mbytes of memory, and 437 CPU seconds. The much greater running time is due to the overhead of partial expansions. The slightly higher memory requirement is because nodes are partially expanded, and nodes that are not closed cannot be pruned by DCFA*. For the set of three random 4000-length sequences, Partial Expansion DCFA* stored 40% more nodes and ran 42% slower than DCFA*. Again, the reason is that partially expanded nodes are not pruned. Finally, we note that when DCFA* is combined with Enhanced A*, it consistently requires more memory (and CPU time) than DCFA* alone, due to the complications discussed earlier.

4.3 15-Puzzle

The 15-puzzle is not a problem for which duplicate detection is a crucial issue. Nevertheless, it is interesting to consider as both a benchmark and an example of an undirected graph. We used 91 of the 100 problem instances in Korf (1985) as a test set. Sparse-A* by itself used an average of 79% of the memory used by A*; DCFA* used an average of 44% of the memory used by A*; and Enhanced A* used an average of 57% of the memory used by A*. A sparse-memory version of Enhanced A* performed best. It used 32% of the memory used by A*. We also tested DCFA* combined with Enhanced A*. It is unable to prevent leaks back into the closed region, for reasons discussed earlier, and we noticed many node re-expansions. Interestingly, it used only 10% of the memory used by A*, due to aggressive pruning, and ran faster than the other algorithms. But this is only because the 15-puzzle is a domain in which node re-generation has less overhead than managing the Open and Closed lists, since IDA* outperforms all of these algorithms on this problem. In undirected graphs with exponentially many duplicate paths, a sparse-memory version of Enhanced A* is likely to perform best.

5 Conclusion

We have proposed a sparse-memory approach to graph search that builds on ideas in earlier work, but implements them in a way that is often more effective. A key advantage of this approach to reducing the size of the Closed list is that it can be combined with a technique for reducing the size of the Open list by upper-bound pruning. This is especially useful for large branching-factor problems where the size of the Open list would otherwise dramatically exceed the size of the Closed list, such as the multiple sequence alignment problem when there are more than four or five sequences.

The sparse-memory approach has other advantages. It allows more flexible, and potentially faster, solution reconstruction. And last but not least, it can behave exactly like A* (or Enhanced A*, or Dijkstra's algorithm) until it reaches a memory limit, and only then removes nodes from the Closed list. Thus, there is (virtually) no overhead for this technique unless a search problem cannot be solved within a given memory bound. Then the overhead for solution reconstruction is compensated for by the reduced memory requirements.

Acknowledgments We thank the anonymous reviewers for helpful comments. This work was supported in part by NSF CAREER grant IIS-9984952 and NASA grant NAG-2-1463.

References

- [Hirschberg, 1975] D. Hirschberg. A linear space algorithm for computing maximal common subsequences. *Communications of the ACM*, 18(6):341–343, 1975.
- [Ikeda and Imai, 1999] T. Ikeda and H. Imai. Enhanced A* algorithms for multiple alignments: Optimal alignments for several sequences and k-opt approximate alignments for large cases. *Theoretical Computer Science*, 210(2):341–374, 1999.
- [Just, 2001] W. Just. Computational complexity of multiple sequence alignment with SP-score. *Journal of Computational Biology*, 8:615–623, 2001.
- [Korf and Zhang, 2000] R. Korf and W. Zhang. Divide-and-conquer frontier search applied to optimal sequence alignment. In *Proceedings of the 17th National Conference on Artificial Intelligence (AAAI-2000)*, pages 910–916, 2000.
- [Korf, 1985] R. Korf. Depth-first iterative deepening: An optimal admissible tree search. *Artificial Intelligence*, 27:97–109, 1985.
- [Korf, 1993] R. Korf. Linear-space best-first search. *Artificial Intelligence*, 62:41–78, 1993.
- [Korf, 1999] R. Korf. Divide-and-conquer bidirectional search: First results. In *Proc. of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99)*, pages 1184–1189, 1999.
- [Lermen and Reinert, 2000] M. Lermen and K. Reinert. The practical use of the A* algorithm for exact multiple sequence alignment. *Journal of Computational Biology*, 7(5):655–671, 2000.
- [McNaughton *et al.*, 2002] M. McNaughton, P. Lu, J. Schaeffer, and D. Szafron. Memory-efficient A* heuristics for multiple sequence alignment. In *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI-02)*, pages 737–743, 2002.
- [Miura and Ishida, 1998] T. Miura and T. Ishida. Stochastic node caching for memory-bounded search. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98)*, pages 450–456, 1998.
- [Myers and Miller, 1988] E. Myers and W. Miller. Optimal alignments in linear space. *Computer Applications in the Biosciences*, 4:11–17, 1988.
- [Reinefeld and Marsland, 1994] A. Reinefeld and T. Marsland. Enhanced iterative-deepening search. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 16:701–710, 1994.
- [Yoshizumi *et al.*, 2000] T. Yoshizumi, T. Miura, and T. Ishida. A* with partial expansion for large branching factor problems. In *Proceedings of the 17th National Conference on Artificial Intelligence (AAAI-2000)*, pages 923–929, 2000.

SPATIAL REASONING

Layered Mereotopology

Maureen Donnelly

Institute for Formal Ontology and Medical Information Science
University of Leipzig, Germany
maureen.donnelly@ifomis.uni-leipzig.de

Abstract

In fields such as medicine, geography, and mechanics, spatial reasoning involves reasoning about entities—for example cavities and invading particles—that may coincide without overlapping. The purpose of this paper is to develop a mereotopology for domains that include coincident entities, such as material objects, holes, geopolitical entities and spatial regions. In addition, I construct mathematical models of this mereotopology in which nontrivial coincidence relations are defined.

1 Introduction

Two objects *coincide* when they occupy overlapping regions of space.¹ My hand and my body (partially) coincide. The Mississippi River and Minnesota (partially) coincide. Note that the objects in both pairs actually stand in a stronger relation than coincidence. The objects in each pair overlap (i.e. share parts). My hand is a part of both itself and my body. The first ten kilometers of the Mississippi River are part of both the river and the state. Any two overlapping entities are, trivially, coincident—their locations are identical at their common parts.

It is an assumption of this paper that the relation of coincidence is broader than that of overlap. In other words, I assume that there are pairs of coincident objects which do not share parts. The food that is currently being digested in my stomach cavity coincides with, but does not overlap, my stomach cavity. The U.S. Embassy in Paris coincides with, but does not overlap, France. Any object coincides with, but does not overlap, the spatial region at which it is located at a given point of time.

A mereotopology is a formal theory of parthood and connection relations. It has long been recognized that mereotopology forms an essential part of formal ontology. Several different mereotopologies have been proposed in recent literature, including [Asher and Vieu,

1995; Borgo *et al.*, 1996; Gotts *et al.*, 1996; Smith, 1996]. These theories are ultimately intended for reasoning about spatial relations among material objects. However, it is assumed in nearly all of this work that the immediate domains of application are restricted to spatial regions.² When material objects are introduced, as in [Cohn, 2001], mereotopological relations are still restricted to regions. Each object's spatial properties are determined by those of the region at which it is located. Thus, distinctive coincidence relations are not usually introduced into standard mereotopologies: on domains of regions, coincidence is just overlap.

Likewise, mathematical models for these theories typically use domains consisting of subsets of a topological space. See, for example, [Biacino and Gerla, 1991; Asher and Vieu, 1995; Cohn and Varzi, 1998]. On these domains, there is no natural way of defining a coincidence relation that is distinct from overlap.

The goal of this paper is to construct a mereotopology for domains that include coincident but non-overlapping entities. In particular, domains for the theory may include both material objects and the regions at which they are located, in addition to other types of entities, such as holes or geopolitical entities, which may coincide with material objects. My theory allows mereotopological relations to apply directly to all entities within the domain, be they regions, material objects, holes, or what have you. It also extends mereotopology by adding coincidence relations and by making explicit the relation between an object and the region at which it is located. To guide the development of the formal theory, I construct mathematical models in which a coincidence relation distinct from overlap is defined.

The formal theory presented in this paper borrows much from the theory of location of [Casati and Varzi, 1999]. It differs in that it divides the domain into differ-

¹ Note that with this usage, coincident objects need only occupy *overlapping* spatial regions. I will use the term "complete coincidence" for the stronger relation that holds between objects that occupy *identical* spatial regions.

² Here and throughout this paper the term "region" designates any part of an immaterial space in which objects are located. I leave open the question of whether these parts may be of mixed dimensionalities. This contrasts with the usage in, for example [Gotts *et al.*, 1996], where the term "region" does not apply to lower-dimensional entities, such as boundaries or points.

ent layers, each of which is mereotopologically independent of the others, and also in providing mathematical models that allow distinct members of the domain to be located at the same region. Additional work on combining either coincidence or location relations with other spatial relations can be found in [Bennett, 2001; Borgo *et al.*, 1997; Cohn, 2001; Schulz and Hahn, 2001].

2 Layered Models

I introduce a class of mathematical structures in which members of the domain can coincide without overlapping. I call these structures *Layered Models* because their domains are partitioned into equivalence classes, called *layers*. Members of the same layer coincide only when they overlap. Members of different layers never overlap and cannot be parts of the same object, though they may coincide. Examples of types of entities assigned to distinct layers are: 1. spatial regions, 2. material objects, 3. holes within these material objects. All regions are parts of a special layer that covers the entire space.

Layered Models are the target models of Layered Mereotopology, the formal theory developed in §3 – §5.

Layered Models are defined as follows. Let $T = \langle X, cl \rangle$ be a topological space, where X is the set of points and cl is the closure operator. Let I be either, N , the set of all natural numbers or an initial segment of N . The elements of the domain, D , of a Layered Model are ordered pairs, $x_i = \langle x, i \rangle$ where $x \subseteq X$ and $i \in I$. (I will generally use the abbreviation, x_i for $\langle x, i \rangle$. All variables referring to objects in Layered Models appear in *Arial* font to distinguish them from the variables of the formal theory.) The second component of each ordered pair determines the layer to which the pair belongs. All pairs of the form $\langle x, 0 \rangle$ (i.e. x_0) belong to a special layer, called the *region layer*.

I require that the domain, D , of any Layered Model satisfy the following conditions:

1. For any nonempty subset, x ($\emptyset \neq x \subseteq X$), $x_0 \in D$.
2. For any $i \in I$ and any $Y \subseteq \wp(X)$, if $y_i \in D$ for all $y \in Y$, then $(\cup_{y \in Y}(y))_i \in D$.
3. For any $i \in I$ and any $x_i, y_i \in D$, if $\sim(x \subseteq y)$, then there is a $z_i \in D$ such that $z \subseteq x$ and $z \cap y = \emptyset$.
4. If $x_i, y_i \in D$ and $x \cap y \neq \emptyset$, then $(x \cap y)_i \in D$.

It follows from 2 that for any $i \in I$, the sum of all members of D of the form x_i is also in D . More precisely, for any $i \in I$, let $Y_i = \{x : x \subseteq X \text{ and } x_i \in D\}$. Then, by 2, $L_i = (\cup_{y \in Y_i}(y))_i \in D$. I will call L_i the *Layer i* of D . Notice that it follows from condition 1 that Layer 0, the region layer, is X_0 . The other layers need not cover that entire topological space. In other words, for $i > 0$, Layer i may be equal to x_i where $x \subset X$.

The parthood relation, P , is defined on a Layered Model, D , as follows:

$$Px_iy_j =: x \subseteq y \text{ and } i = j.$$

Notice that it follows from this definition of parthood that any $x_i \in D$ is part of Layer i . Also, when overlap, O , is defined in terms of P in the usual way,

$$O(x_i, y_j) =: \exists z(P(z, x_i) \& P(z, y_j)),$$

then it follows that only objects which are parts of the same layer can overlap: $O(x_i, y_j)$ iff $x \cap y \neq \emptyset$ and $i = j$.

If underlap, U , is defined in the usual way,

$$U(x_i, y_j) =: \exists z(P(x_i, z) \& P(y_j, z))$$

then it follows that, since all objects are parts of their layer, objects underlap if and only if they are parts of the same layer: $U(x_i, y_j)$ iff $i = j$.

A connection relation, C , is defined on D as:

$$Cx_iy_j =: cl(x) \cap y \neq \emptyset \text{ or } x \cap cl(y) \neq \emptyset \text{ and } i = j.$$

As with parthood, only objects residing on the same layer can be connected to each other.

Other mereotopological relations, such as external connection and tangential parthood, can be defined in terms of P and C in the usual way. For example, an external connection relation, EC , can be defined as follows:

$$EC(x_i, y_j) =: \sim O(x_i, y_j) \text{ and } C(x_i, y_j).$$

In contrast to parthood and connection relations, coincidence relations can hold between objects from different layers. Three different coincidence relations are defined on D .

$$Coin(x_i, y_j) =: x \cap y \neq \emptyset \quad (\text{coincidence})$$

$$CCoin(x_i, y_j) =: x = y \quad (\text{complete coincidence})$$

$$Cov(x_i, y_j) =: x \subseteq y \quad (\text{covering})$$

Another relation that can hold among objects from different layers is abutment, A .

$$A(x_i, y_j) =: x \cap y = \emptyset \& (cl(x) \cap y \neq \emptyset \text{ or } x \cap cl(y) \neq \emptyset)$$

Finally, I add the function, r , from D to Layer 0 which assigns each member of the domain to its representative on the region layer:

$$r(x_i) = x_0$$

Notice that, when restricted to Layer 0, r is the identity function: for any $x_0 \in D$, $r(x_0) = x_0$. Also notice that two objects abut if and only if their regions are externally connected.

I will give a simple example of a Layered Model designed to illustrate the way in which the models can be used to represent spatial relations among regions, material objects, and immaterial entities such as holes. The background topological space for this model is $\mathbb{R}^3$ with its standard topology. The region layer in this model has as parts the members of the set $\{x_0 : \emptyset \neq x \subseteq \mathbb{R}^3\}$.

Suppose that we wish to represent relations holding among a vase, a portion of water in the vase, a flower standing in the vase, and the interior of the vase. The vase, water, and flower are represented, respectively, by parts, v_1 , w_1 , and f_1 , of Layer 1 where the subsets, v , w , and f , of $\mathbb{R}^3$ are disjoint. The interior of the vase, a hole, is represented by h_2 on Layer 2 where h and v are disjoint, but connected, w is a proper subset of h ($w \subset h$), and h and f have a nonempty intersection ($h \cap f \neq \emptyset$). According to this representation, the water is not part of the hole, but the fact that the water is contained in the interior of the vase is represented in the model by the covering relation, $Cov(w_1, h_2)$, holding between w_1 and h_2 . Similarly, the flower does not overlap the interior of the vase, but the fact that the flower is partially contained in the interior of the vase is represented by the coincidence relation: $Coin(f_1, h_2)$. The vase and its interior do

not coincide. Nor are they connected. But the fact that the vase touches its interior is represented by the abutment relation: $A(v_1, h_2)$. Finally, every object is exactly collocated with its spatial region. This is represented in the model by the complete coincidence of the object and its region. For example, $CCoin(v_1, v_0)$.

3 Layered Mereology

A mereology is a formal theory of the binary parthood relation. My aim in this section is to develop a mereology, called *Layered Mereology*, that is satisfied by the parthood relation, P , defined on Layered Models. Layered Mereology is extended in §4 and §5 to Layered Mereotopology, a theory that also includes coincidence and connection relations.

Layered Mereology (and also its extension, Layered Mereotopology) is formulated in first-order logic. Relations and functions are represented in the formal theory with plain text letters to distinguish them from the relations and functions defined in Layered Models. Layered Mereology assumes one primitive, the binary relation P , which, on the intended interpretation, represents parthood.

The following relations are defined in the formal theory in terms of P :

- (D1) $PPxy =: Pxy \ \& \ \sim Pyx$ (x is a proper part of y)
- (D2) $Oxy =: \exists z (Pzx \ \& \ Pzy)$ (x and y overlap)
- (D3) $Uxy =: \exists z (Pzx \ \& \ Pzy)$ (x and y underlap)

It is trivial to see that, if P is interpreted as P in Layered Models, the defined relations O and U will be interpreted as U and O , respectively.

The axioms of Layered Mereology will be somewhat nonstandard. For example, they cannot require that any pair of objects, x and y , have a mereological sum. My goal is to axiomatize P in such a way that, when restricted to a single layer, it satisfies the axioms (and axiom schema) of General Extensional Mereology (GEM):³

- (P1) Pxx
- (P2) $Pxy \ \& \ Pyx \rightarrow x = y$
- (P3) $Pxy \ \& \ Pyz \rightarrow Pxz$
- (P4) $\sim Pxy \rightarrow \exists z (Pzx \ \& \ \sim Ozy)$
- (GEM5) $\exists x \phi \rightarrow \exists z \forall w (Owz \leftrightarrow \exists x (\phi \ \& \ Owx))$

The ϕ in (GEM5) stands for any first-order formula in which z does not occur free. (GEM5) states that if any member of the domain satisfies the formula ϕ , then there must be a sum of all objects satisfying ϕ in the domain. (GEM5) must be altered for Layered Mereology because Layered Models only include the sums of objects that are parts of the same layer.

I will discuss summation in Layered Models in more detail shortly. For now, notice that the relation P in Layered Models satisfies each of the first four axioms of

GEM. They are therefore used in their original forms as axioms for Layered Mereology.⁴

The first three axioms require that P is a partial ordering. (P1) says that P is reflexive. (P2) says that P is anti-symmetric. (P3) says that P is transitive. That (P1)-(P3) are satisfied by P in Layered Models follows immediately from the fact that P is just the subset relation restricted to the separate layers.

(P4) says that, if x is not a part of y , then there is some part, z , of x that does not overlap y . (P4) is satisfied in Layered Models by virtue of condition 3 of §2.

It follows from (P1) - (P4) that overlap, O , is extensional. This means that any two members of the domain that overlap the same objects are identical.

(PT1) $x = y \leftrightarrow \forall z (Oxz \leftrightarrow Oyz)$

Because O is extensional, for any formula ϕ in which z does not occur free, if we can assign z to a member of the domain that satisfies

$$\forall w (Owz \leftrightarrow \exists x (\phi \ \& \ Owx)) \quad (*)$$

then this object is the unique sum of all ϕ -ers. However, for any such formula ϕ , there need not be an object satisfying (*) even if some member of the domain satisfies ϕ . For example, let ϕ be $x = x$ and let D be the domain of a Layered Model. Then every member of D satisfies ϕ . But if there are $x_i, y_j \in D$ with $i \neq j$, then no member of D satisfies (*). This is because such an object would have to overlap every member of D and there can be no member of D that overlaps both x_i and y_j for $i \neq j$.

Thus, we need a restricted version of (GEM5) that requires sums to exist only if all summands are part of the same layer. Such an axiom schema will be satisfied in all Layered Models by virtue of condition 2 of §2. Given that two objects in a Layered Model are parts of the same layer if and only if they underlap, the restricted summation axiom schema can read as follows:

(P5) $(\exists x \phi \ \& \ \forall x, y (\phi \ \& \ \phi/y \rightarrow Uxy)) \rightarrow \exists z \forall w (Owz \leftrightarrow \exists x (\phi \ \& \ Owx))$

Here, ϕ/y is the formula ϕ with all free instances of x replaced by y (and where variable substitution is performed as necessary so y is free in ϕ/y where x is free in ϕ). (P5) says that if there is some object that satisfies ϕ and any two objects that satisfy ϕ underlap, then there is a sum of all objects satisfying ϕ .

For convenience, when ϕ is any formula in which z does not occur free, I will use the abbreviation $zSMx[\phi]$ for the substitution instance of the formula (*).

We would like to be able to say more things about layers within the mereology. So far, we can only say that two objects are on the same layer. We would like to be able to say that a certain object is a layer or is the layer of a particular object. (D4) defines a relation holding between y and z when z is the sum of all objects that y underlaps (i.e. y 's layer).

(D4) $Lyx =: zSMx[Uxy]$ (z is y 's layer)

³ Throughout this paper, initial universal quantifiers are suppressed unless they are needed for clarity.

⁴ All axioms of Layered Mereology are labeled with a "P". "PT" is used for theorems of Layered Mereology.

It is easy to see that when P is interpreted as $\mathbf{P}$ in Layered Models, $Lx_i y_j$ holds if and only if $y_j = \text{Layer } i$. However, axioms (P1)-(P5) do not allow us to infer that for any object, y , there is some object that is the sum of all objects that underlap y . In other words, our axioms so far do not allow us to infer that every object has a layer. This would follow from (P5) if we knew that any two objects that underlap y must underlap each other (i.e. that U is transitive). Notice that the underlap relation for Layered Models, U , is an equivalence relation (reflexive, symmetric, and transitive) and that the sets consisting of all objects from a single layer are the equivalence classes determined by the U relation. It follows from (P1)-(P5) that U is reflexive and symmetric, but not that U is transitive.

I, therefore, add a final axiom to Layered Mereology:
(P6) $(Uxy \ \& \ Uyz) \rightarrow Uxz$ (underlap is transitive)

It follows immediately that every object has a layer:
(PT2) $\forall y \exists z Lyz$ (every object has a layer)

Thus the relation, L , is a function. I will use the functional term $l(x)$ to stand for the layer of x .

The following theorems can also be derived:

- (PT3) $Pxl(x)$ (every object is part of its layer)
(PT4) $Uxy \leftrightarrow l(x) = l(y)$
(two objects underlap iff they have the same layer)
(PT5) $Uxy \leftrightarrow Pyl(x)$
(x underlaps y iff y is part of x 's layer)

The unary predicate, LY , distinguishes certain members of the domain as layers.

- (D5) $LYz =: \exists x Lxz$ (z is a layer)

It follows easily from (PT3), (PT4), and (D5) that:
(PT6) $LYz \leftrightarrow Lzz$ (z is a layer iff z is its own layer)

(PT6) tells us that layers are those members of the domain which are the mereological sums of all objects that they underlap.

When P is interpreted as $\mathbf{P}$ in Layered Models, LYx_i if and only if $x_i = \text{Layer } i$.

We can use the formula schema $zSMx[\phi]$ to introduce more useful relations. (D6) - (D8) are the standard definitions of the sum, product, and difference relations. (D9) defines a *relative complement* relation.

- (D6) $+(v, y, z) =: zSMx[Pxv \vee Pxy]$
(z is the binary sum of v and y)
(D7) $\times(v, y, z) =: zSMx[Pxv \ \& \ Pxy]$
(z is the binary product of v and y)
(D8) $-(v, y, z) =: zSMx[Pxy \ \& \ \sim O xv]$
(z is the difference of v in y)
(D9) $-(v, z) =: zSMx(Uxv \ \& \ \sim O xv)$
(z is the relative complement of v)

The following theorems concerning the relative complement can be derived:

- (PT7) $\exists z(-x, z) \leftrightarrow \sim LYx$ (x has a relative complement iff x is not a layer)

(PT8) $-(x, z) \leftrightarrow -(x, l(x), z)$ (z is x 's relative complement iff z is the difference of x in x 's layer)

Using (PT4) it is easy to prove that, when they exist, sums, products, and relative complements belong to the same layer as the original object(s).

It is straightforward to show that any layer of a model of Layered Mereology is a model of General Extensional Mereology (GEM). More precisely, we can prove the following:

(Meta-Theorem) Let M be any model of Layered Mereology with domain, D . Note that M need not belong to the class of Layered Models defined in §2. Let $w \in D$ and let $D_w = \{y : y \in D \ \& \ l(y) = l(w)\}$. Let M_w be the structure whose domain is D_w with P interpreted as in M (i.e. Pyz holds in M_w iff $y, z \in D_w$ and Pyz holds in M). Then M_w satisfies axioms (P1) – (P4) and axiom schema (GEM5).

4 The Region Function

In Layered Mereology, we have no way of stating that two objects coincide. Layered Mereology lets us describe the parthood relations between objects. It does not let us describe the (relative) locations of objects. To do this, I extend Layered Mereology by adding the unary function, r . On the intended interpretation r assigns each object, x , to the region, $r(x)$, at which x is exactly located. In Layered Models, r is interpreted as the function $\mathbf{r}$.

Using r , we can define a one-place predicate, R , which distinguishes the sub-domain of regions.

- (D10) $Ry =: \exists x(r(x) = y)$ (y is a region)

When r is interpreted as $\mathbf{r}$ in Layered Models, Rx_i if and only if $i = 0$.

The axioms for r are added to axioms (P1) - (P6).⁵ It is easy to check that they are satisfied in Layered Models.

- (R1) $Ry \ \& \ Rz \rightarrow Uyz$ (all regions are located in the same layer)
(R2) $Ry \ \& \ Uyz \rightarrow r(z) = y$ (every member of the region layer is its own region)

The theorems below are immediate consequences of (R1) and (R2).

- (RT1) $Ry \rightarrow r(y) = y$ (every region is located at itself)
(RT2) $Ry \ \& \ Uyz \rightarrow Rz$ (every member of a region's layer is a region)
(RT3) $\exists x \phi \ \& \ \forall x (\phi \rightarrow Rx) \ \& \ zSMx[\phi] \rightarrow Rz$
(every sum of regions is a region)

Additional axioms relate the region function to parthood.

- (R3) $Pxy \rightarrow Pr(x)r(y)$ (if x is part of y , then x 's region is part of y 's region)

Notice that the converse of (R3) is not generally satisfied in Layered Models. $r(x_i)$ may be part of $r(y_j)$ even though x_i is not part of y_j . This will be the case whenever $x \subseteq y$, but $i \neq j$.

On the other hand, in Layered Models, if $O(r(x_i), r(y_j))$ and $i = j$, then $O(x_i, y_j)$ must also hold.

⁵ All axioms specific to the r function are marked with "R". Theorems specific to the r function are marked with "RT".

(R4) $Uxy \ \& \ O(r(x), r(y)) \rightarrow Oxy$ (if x and y are on the same layer and x 's region overlaps y 's region, then x overlaps y)

The coincidence relations are defined in terms of the region function, r .

(D11) $CCoin(x, y) =: r(x) = r(y)$ (x and y completely coincide)

(D12) $Cov(x, y) =: P(r(x), r(y))$ (y covers x)

(D13) $Coin(x, y) =: O(r(x), r(y))$ (x and y coincide)

It is easy to check that when P is interpreted as $\mathbf{P}$ and r as $\mathbf{r}$ in Layered Models, $CCoin$, Cov , and $Coin$ are, respectively, $\mathbf{CCoin}$, $\mathbf{Cov}$, and $\mathbf{Coin}$.

The following theorems can be derived from (P1)-(P6) and (R1)-(R4).

(RT4) $Cov(x, y) \ \& \ Cov(y, x) \leftrightarrow CCoin(x, y)$ (y covers x and x covers y iff x and y completely coincide)

(RT5) $CCoin(x, y) \ \& \ CCoin(x, z) \ \& \ Uyz \rightarrow y = z$ (any object can completely coincide with at most one object in any layer)

(RT6) $CCoin(x, y) \ \& \ Uxy \rightarrow x = y$ (if x and y completely coincide and are on the same layer, then $x = y$)

(RT7) $Cov(x, y) \ \& \ Uxy \rightarrow Pxy$ (if y covers x and x and y are on the same layer, then x is part of y)

(RT8) $Coin(x, y) \ \& \ Uxy \rightarrow Oxy$ (if x and y coincide and are on the same layer, then x and y overlap)

In addition, the implications illustrated in the diagram below can be derived. The arrow indicates that the atomic formula at the start of the arrow implies the atomic formula at the end of the arrow.

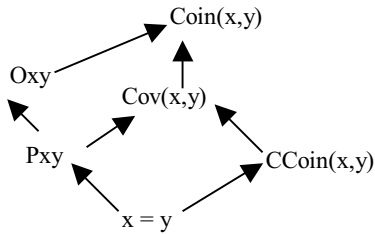


Figure 1: Implication hierarchy

5 Layered Mereotopology

The base theory can be extended to Layered Mereotopology in a straightforward way by adding a connection relation, C , where Cxy means " x is connected to y ". C is interpreted as $\mathbf{C}$ in Layered Models.⁶

(C1) Cxx (connection is reflexive)

(C2) $Cxy \rightarrow Cyx$ (connection is symmetric)

(C3) $Pxy \rightarrow \forall z(Czx \rightarrow Czy)$ (if x is part of y , everything connected to x is connected to y)

(C4) $Cxy \rightarrow Uxy$ (if x and y are connected, then they are parts of the same layer)

(C5) $Cxy \rightarrow C(r(x), r(y))$ (if x and y are connected, their regions are also connected)

⁶ Axioms specific to Layered Mereotopology are marked with a "C". Theorems are marked with "CT"

(C6) $Uxy \ \& \ C(r(x), r(y)) \rightarrow Cxy$ (if x and y are members of the same layer and their regions are connected, then x and y are connected)

External Connection and Abutment are defined as follows.

(D14) $ECxy =: Cxy \ \& \ \sim Oxy$ (x and y are externally connected)

(D15) $Axy =: EC(r(x), r(y))$ (x and y abut)

When C is interpreted as $\mathbf{C}$ in Layered Models, EC is the relation, $\mathbf{EC}$ and A is the relation, $\mathbf{A}$.

The following theorems are easily derived:

(CT1) $ECxy \rightarrow Axy$ (if x and y are externally connected, then x and y abut)

(CT2) $Uxy \rightarrow (Axy \leftrightarrow ECxy)$ (if x and y are on the same layer, then they abut iff they are externally connected)

(CT3) $Axy \rightarrow \sim Coin(x, y)$ (if x and y abut, then they do not coincide)

The tangential part relation, TP , is usually defined in terms of external connection as follows:

$TPxy =: Pxy \ \& \ \exists z(ECzx \ \& \ ECzy)$ (**)

This definition is not appropriate for layered models. To see why, consider the Layered Model whose underlying topological space is $\mathbb{R}$ with its standard topology and whose layers have the members of the following sets as parts:

Layer 0: $\{x_0 : \emptyset \neq x \subseteq \mathbb{R}\}$

Layer 1: $\{x_1 : \emptyset \neq x \subseteq [0, 1]\}$

It would follow from the standard definition of TP that $[0, 1]_1$ has no tangential parts, since it is not externally connected to any member of the domain. For example, it would follow that $[0, 0]_1$ and $[1, 1]_1$ are not tangential parts of $[0, 1]_1$.

I will therefore use the following definition of tangential part:

(D16) $TPxy =: Pxy \ \& \ \exists z(Azx \ \& \ Azy)$ (x is a tangential part of y)

Applying this definition to the previous model, it turns out that any part of $[0, 1]_1$ that contains either $[0, 0]_1$ or $[1, 1]_1$ is a tangential part of $[0, 1]_1$. More generally, it follows from (D16) that, for objects x and y on the same layer, x is a tangential part of y if and only if x 's region is a tangential part of y 's region.

(CT5) $Uxy \rightarrow (TPxy \leftrightarrow TP(r(x), r(y)))$

Interior parthood is then defined as usual.

(D17) $IPxy =: Pxy \ \& \ \sim TPxy$ (x is an interior part of y)

Relational counterparts of topological operators can now be defined in the usual way. For instance, an object's interior can be defined as the sum of its interior parts:

(D18) $INT(y, z) =: zSMx[IPxy]$ (z is the interior of y)

Since each layer of any model of Layered Mereology is a model of GEM, it is easy to show that each layer of any model of Layered Mereotopology is a model of the standard mereotopology which uses (**) as the definition of tangential parthood and includes axioms (C1)-(C3) in addition to those of GEM.

6 Conclusions and Further Work

The goal of this paper was to construct a mereotopology for domains that include coincident but non-overlapping entities. The result is an extension of mereotopology that includes coincidence relations and a region function in addition to mereotopological relations.

These additional elements of the theory make it particularly appropriate for applications that involve reasoning about objects that are located in holes. Reasoning about holes is crucial in a wide variety of domains, including medicine (body cavities and orifices) and mechanics (valves, pathways formed by piping). For more examples, see [Casati and Varzi, 1994].

The particular theory presented in this paper allows the same mereotopological relations to apply directly to all spatial entities, including regions, material objects, and holes. This approach is an alternative to that of [Cohn, 2001] in which spatial relations apply to material objects only indirectly, via the spatial regions at which they are located. One advantage of allowing direct descriptions of the spatial properties of material objects is that this leaves open the possibility of attributing different spatial structures to material objects and the regions at which they are located. For example, we may wish to represent material objects as having only closed, regular, divisible parts, but represent spatial regions as sums of points. Slight changes in the conditions on the domains of Layered Models (specifically, condition 4) and in the axioms of Layered Mereotopology (specifically, (R4)) would allow models in which the parts of different layers are restricted to different granularities.

One other project for further work is to relax the restrictions on the underlap relation so that members of the domain that properly coincide (i.e. coincide without overlapping) can be parts of the same object. An organism has both material parts, such as a heart and a liver, and immaterial parts, such as an abdominal cavity. It has been argued in [Schulz and Hahn, 2001] that material and immaterial parts of the body may properly coincide: the brain is located in the cranial cavity, but it is not part of the cranial cavity. It is possible to define models similar Layered Models in which properly coincident members of the domain may underlap. It remains to develop a formal theory of coincidence and mereotopological relations for these kinds of models.

Acknowledgements

I am grateful for the helpful comments of Jonathan Simon, Barry Smith, Thomas Bittner, and three anonymous reviewers on previous drafts of this paper. This work was supported by the Alexander von Humboldt Foundation under the auspices of its Wolfgang Paul Program.

References

- [Asher and Vieu, 1995] N. Asher and L. Vieu. Towards a Geometry of Commonsense: A Semantics and a Complete Axiomatization of Mereotopology. *Proceedings of IJCAI'95*, 846-852, Morgan Kaufmann, San Mateo, CA, 1995.
- [Bennett, 2001] B. Bennett. Space, Time, Matter, and Things. In C. Welty & B. Smith (Eds.), *Proceedings of FOIS'01*, 105-116, ACM Press, New York, 2001.
- [Biacino and Gerla, 1991] L. Biacino and G. Gerla. Connection Structures. *Notre Dame Journal of Formal Logic*. 32: 242-247, 1991.
- [Borgo et al., 1996] S. Borgo, N. Guarino, and C. Masalo. A Pointless Theory of Space Based on Strong Connection and Congruence. In L. C. Aiello & S. Shapiro (Eds.), *Proceedings of KR'96*, 220-229, Morgan Kaufmann, San Mateo, CA, 1996.
- [Borgo et al., 1997] S. Borgo, N. Guarino, and C. Masalo. An Ontological Theory of Physical Objects. In L. Ironi (Ed.), *Proceedings of QR'97*, 223-231, Cortona, Italy, 1997.
- [Casati and Varzi, 1994] R. Casati and A. C. Varzi. *Holes and Other Superficialities*. MIT Press, Cambridge, MA, 1994.
- [Casati and Varzi, 1999] R. Casati and A. C. Varzi. *Parts and Places. The Structures of Spatial Representation*. MIT Press, Cambridge, MA, 1999.
- [Cohn, 2001] A. G. Cohn. Formalizing Bio-Spatial Knowledge. In C. Welty & B. Smith (Eds.), *Proceedings of FOIS'01*, 198-209, ACM Press, New York, 2001.
- [Cohn and Varzi, 1998] A. G. Cohn and A. C. Varzi. Connection Relations in Mereotopology. In H. Prade (Ed.), *Proceedings of ECAI'98*, 150-154, Wiley 1985.
- [Gotts et al., 1996] N. M. Gotts, J. M. Gooday, and A. G. Cohn. A Connection Based Approach to Commonsense Topological Description and Reasoning. *The Monist*, 79: 51-75, 1996.
- [Schulz and Hahn, 2001] S. Schulz and U. Hahn. Mereotopological Reasoning about Parts and (W)Holes in Bio-Ontologies. In C. Welty & B. Smith (Eds.), *Proceedings of FOIS'01*, 210-221, ACM Press, New York, 2001.
- [Smith, 1996] B. Smith. Mereotopology: A Theory of Parts and Boundaries. *Data & Knowledge Engineering*. 20: 287-303, 1996.

Reasoning about distances

Frank Wolter and Michael Zakharyashev

Department of Computer Science, University of Liverpool,
Liverpool L69 7ZF, U.K.;

Department of Computer Science, King's College,
Strand, London WC2R 2LS, U.K.

(emails: frank@csc.liv.ac.uk, mz@dc.s.kcl.ac.uk)

Abstract

The paper presents a novel expressive logic-based formalism intended for reasoning about numerical distances. We investigate its computational properties (in particular, show that it is EXPTIME-complete) and devise a tableau-based satisfiability-checking algorithm. To be able to express knowledge about implicit or unknown distances, we then extend the language with variables ranging over distances and prove that the resulting logic is decidable as well.

1 Introduction

The numerical notion of *distance* or *metric* as a measure of closeness or similarity (in a great variety of senses) plays a fundamental role in many branches of science—including, naturally, knowledge representation, artificial intelligence and informatics in general. Recent examples are such different fields as spatial KR&R and geographic information systems [Cohn and Hazarika, 2001; Adam and Gangopadhyay, 1997], computational molecular biology [Setubal and Meidanis, 1997; Clote and Backofen, 2000], text processing [Salton, 1989] and data mining [Dunham, 2003]. Unlike classical physics and the corresponding mathematical models, the ‘distance spaces’ arising in these applications may be quite different from the standard Euclidean spaces, e.g., they are often finite and do not necessarily satisfy all of the metric axioms. Another important feature is that properties of *only some* objects in the space may be known, and the task is to use the available information to determine properties of other elements of interest. Traditional methods of dealing with metric data are mostly numerical or ‘sub-symbolic’ (e.g., graph-theoretic or probabilistic) and usually do not involve any kind of (automated) reasoning which is based upon symbolic, explicit knowledge representation. *The main aim of this paper is to present a new expressive logic-based formalism intended for reasoning about distances and to provide it with a tableau decision algorithm.*

The language we propose is—as far as its algorithmic properties and expressiveness are concerned—a compromise between full first-order and propositional languages: it contains object names and unary atomic predicates from which complex predicates are constructed by means of the Boolean oper-

ators and restricted quantifiers like, e.g., $\forall^{<r}$ (‘for all objects within a circle of radius r ’) and $\exists^{<r}$ (‘for some object within a circle of radius r ’). Thus, $\exists^{<r}P$ denotes the set of all objects each of which is located at distance $< r$ from at least one object with property P . To illustrate the expressive power of the language and to convince the reader that it can be useful, we give three concrete examples.

Example 1. Consider a space of proteins (i.e., sequences of amino acids); some of them are well studied, but properties of the majority are not known. Using one of the available techniques, we can define a ‘similarity measure’ between proteins which gives rise to a certain ‘distance’ (perhaps, metric) space. One of the typical problems is to recognise whether a given protein p has some property P , e.g., whether it contains an EF-hand motif. The usual approach of comparing the structure of p with that of ‘closely located’ proteins whose properties are known can be supplemented with a knowledge base composed by an expert and containing information like

- $\exists^{\leq 5}Q \sqcap P \doteq \perp$ — ‘no protein located at distance ≤ 5 from a protein with a property Q has P ,’
- $\exists^{\leq 2}\{\ell\} \sqsubseteq R$ — ‘all proteins located at distance ≤ 2 from a particular protein ℓ have a property R ,’ etc.

Note that, in view of incomplete information about properties of elements of the distance space, to answer the query whether p can have P , we have to run a satisfiability-checking algorithm rather than a model-checking one.

Example 2. Suppose now that we are trying to organise a web site for selling books (or any other goods) on-line. One way of classifying the existing books is to introduce a measure of closeness of the contents of *some* of them. We then obtain a database containing items of the form $\delta(b_1, b_2) = a$, if the distance between books b_1 and b_2 is a , and, say, $\delta(b_1, b_2) > c$, for some number c , if the distance between b_1 and b_2 is only known to be greater than c . Now we can build a sort of terminology by saying:

- $Harry_Potter \in \text{Fiction}$,
- $\text{Fiction} \sqcap \exists^{\leq 3}\text{Science} \sqsubseteq \text{Science_Fiction}$,
- $\text{Science_Fiction} \sqsubseteq \forall^{\leq 2}\neg\text{Science}$,
- $\exists^{<1}AI \sqcap \exists^{\leq 1}\text{Science_Fiction} \not\sqsubseteq \perp$,
- $\forall^{\leq 1}\text{Computer_Science} \sqsubseteq \text{Science}$, etc.,

and deduce, for instance, that AI is not subsumed by Computer Science.

Example 3. Perhaps the most natural application of this kind of formalism is spatial representation and reasoning. We can speak in terms of ‘absolute’ distances:

- $\delta(\text{London}, \text{Leipzig}) > 100$,
- $\text{house} \in \exists^{<1} \forall^{\leq 0.5} (\text{shop} \sqcup \text{restaurant})$ — ‘the house is located within 1 mile distance from a shopping centre of radius 0.5,’
- $\text{doughnut} \doteq \exists^{\leq 10} \{ \text{centre} \} \sqcap \neg \exists^{<6} \{ \text{centre} \}$.

We can also deal with distances given implicitly by saying, for example,

- $\delta(\text{London}, \text{Leipzig}) > 3 \cdot \delta(\text{Liverpool}, \text{Manchester})$,
- $\text{my_house} \in \exists^{<x} \text{college}$, $\delta(\text{Dov's_house}, \text{college}) \doteq x$ (‘I do not want to have a house that is located farther from my college than Dov’s house’)

without referring to concrete distances between the cities and perhaps having no idea where Dov’s house is.

The approach to reasoning about distances we advocate in this paper has grown up from four main sources. First, extensive research has been done by the (spatial/constraint) database community [Revesz, 2002] aiming to deal efficiently with large sets of data which come equipped with some kind of distance (or similarity) measure; consult, e.g., the literature on ‘similarity search’ and ‘ k -nearest neighbour queries’ [Bozkaya and Özsoyoglu, 1999; Hjaltason and Samet, 1999]. Although this paper does not contribute to this field,¹ we were driven by the idea that it may be useful to extend the ‘evaluation-based’ methods by automated reasoning services. As was mentioned above, complete information about models based on distance spaces, and even distances themselves, are often not available, which means that the quality of solutions provided by the database technology can be significantly improved by using explicit knowledge of the field (or even conjectures) supported by appropriate reasoning procedures.

Two approaches to extend the database technology by means of KR&R have been developed by the qualitative spatial reasoning community. One is based upon the representation of spatial data by means of constraint systems [Renz and Nebel, 1999], while the other uses first-order theories, e.g., [Randell *et al.*, 1992]. Constraint systems are usually very efficient (at most NP-complete); however, their expressive power is rather limited (for instance, they do not allow any kind of quantification). On the other hand, the unrestricted quantification of first-order representations is a serious obstacle for their use in practical reasoning systems.

In a sense, our approach is similar to that taken by the description logic community which represents conceptual knowledge by means of restricted quantification (a kind of modal necessity and possibility operators) and uses reasonably effective tableau-based procedures even for expressive and complex (say, EXPTIME-complete) formalisms [Horrocks, 1998]. The chief difference from description logics is

¹Note, however, that our logics can express some of the clustering conditions.

that we have to integrate rather sophisticated reasoning about *quantitative data* into a logic-based framework.

Logics of distance spaces were introduced in [Sturm *et al.*, 2000; Kutz *et al.*, 2003], where our main concern was to find out whether such logics can be decidable. Unfortunately, the logics constructed in these papers were not supported by effective reasoning machinery. The aim of this paper is more practical. To begin with, in Section 2, we present an application oriented variant $\mathcal{MS}_{\leq}^{\leq}[\mathbb{Q}^+]$ of a distance logic (containing location constants, set variables and restricted quantifiers) and show that the satisfiability problem for its formulas in metric (and some weaker distance) spaces is EXPTIME-complete. As distance spaces used in computer science and AI are often finite (e.g., data with a similarity measure, a model of a railway system, etc.), we consider also the problem of satisfiability of $\mathcal{MS}_{\leq}^{\leq}[\mathbb{Q}^+]$ knowledge bases in finite spaces. We show that the language does not ‘feel’ the difference between finite and infinite spaces: all the results presented for satisfiability in arbitrary spaces hold for finite spaces as well. (Unfortunately, if we restrict attention to the two-dimensional Euclidean space or its subspaces, then the satisfiability problem becomes undecidable.) Then, in Section 3, we devise a tableau-based satisfiability checking algorithm for this logic. An interesting and quite unexpected consequence of the proofs of these results is the following observation. The full $\mathcal{MS}_{\leq}^{\leq}[\mathbb{Q}^+]$ contains both operators $\exists^{\leq a}$ and $\exists^{< a}$ which together allow assertions like

- $\text{house} \in \exists^{\leq 1} \text{shop} \sqcap \neg \exists^{< 1} \text{shop}$ — ‘the closest shop is located in precisely 1 mile distance from the house.’

But in everyday life we often do not pay attention to the boundaries of spatial regions or treat them in a sort of vague manner. What happens if we keep only one of the operators $\exists^{\leq a}$ or $\exists^{< a}$ in our language? Do the two resulting fragments behave differently, or to put it another way, does our language feel the difference between regions *with* and *without* boundaries? To our surprise, the answer turns out to be ‘NO’ (for a precise formulation of the result see Theorem 7).

To be able to express knowledge about implicit or unknown distances, in Section 4 we extend $\mathcal{MS}_{\leq}^{\leq}[\mathbb{Q}^+]$ with variables ranging over distances and prove that the satisfiability problem for the resulting language is decidable.

Because of lack of space we had to omit all (rather non-trivial and lengthy) proofs. They are available at <http://www.csc.liv.ac.uk/~frank>. Here we only say that the complexity result as well as the proof of the correctness and completeness of the tableau algorithm are based on a representation of metric spaces in the form of relational structures *à la* Kripke frames. This representation allows us to use advanced techniques from modal and description logics.

2 The logic $\mathcal{MS}_{\leq}^{\leq}[M]$

Let M be a non-empty subset of positive rational numbers (i.e., a subset of $\mathbb{Q}^+$). Denote by $\mathcal{MS}_{\leq}^{\leq}[M]$ the language with the following alphabet and term and formula formation rules. The alphabet consists of

- an infinite list of *set (or region) variables* $X_1, X_2, \dots$;
- an infinite list of *location constants* $\ell_1, \ell_2, \dots$,

- the *set constant* (or *nominal*) $\{\ell_i\}$, for every ℓ_i ,
- binary *distance* (δ), *equality* ($\doteq$) and *membership* ($\in$) *predicates*;
- the *Boolean operators* $\sqcap$, $\neg$ (and their derivatives $\sqcup$, $\top$ and $\perp$);
- two *distance quantifiers* $\exists^{<a}$, $\exists^{\leq a}$ and their duals $\forall^{<a}$, $\forall^{\leq a}$, for every $a \in M$;
- two *universal quantifiers* $\exists$ and $\forall$.

Set terms s of $\mathcal{MS}_{\leq}^{\leq}[M]$ are defined as

$$s ::= X_i \mid \{\ell_i\} \mid \top \mid \perp \mid \neg s \mid s_1 \sqcap s_2 \mid s_1 \sqcup s_2 \mid \exists^{<a} s \mid \exists^{\leq a} s \mid \forall^{<a} s \mid \forall^{\leq a} s \mid \exists s \mid \forall s,$$

and the *formulas* of $\mathcal{MS}_{\leq}^{\leq}[M]$ are

- $\ell \in s$, where ℓ is a location constant and s a set term,
- $s \doteq t$ and $s \neq t$, where s and t are set terms,
- and all ‘distance atoms’ of the form
 $\delta(k, \ell) < a$, $\delta(k, \ell) \leq a$, $\delta(k, \ell) > a$, $\delta(k, \ell) \geq a$,

where k, ℓ are location constants and $a \in M$.

We use $s_1 \sqsubseteq s_2$ as an abbreviation for $s_1 \sqcap s_2 \doteq s_2$.

The intended semantics for this language is defined as follows. An $\mathcal{MS}_{\leq}^{\leq}[M]$ -model is a structure of the form

$$\mathfrak{B} = \langle W, d, X_1^{\mathfrak{B}}, X_2^{\mathfrak{B}}, \dots, \ell_1^{\mathfrak{B}}, \ell_2^{\mathfrak{B}}, \dots \rangle,$$

where $\langle W, d \rangle$ is a *metric space* with a *distance function* d satisfying the axioms

$$d(x, y) = 0 \text{ iff } x = y, \quad (1)$$

$$d(x, z) \leq d(x, y) + d(y, z), \quad (2)$$

$$d(x, y) = d(y, x), \quad (3)$$

for all $x, y, z \in W$, $X_i^{\mathfrak{B}} \subseteq W$, and $\ell_i^{\mathfrak{B}} \in W$. Thus, $\mathfrak{B}$ defines explicitly the values of set variables and location constants. (Although we do not make the unique name assumption—i.e., $\ell_i^{\mathfrak{B}} = \ell_j^{\mathfrak{B}}$, $i \neq j$, is not prohibited—it is easy to force different location constants to denote different points by including $\{\ell_i\} \sqsubseteq \neg\{\ell_j\}$ into the knowledge base.) The *values* $s^{\mathfrak{B}}$ of other $\mathcal{MS}_{\leq}^{\leq}[M]$ -terms s in $\mathfrak{B}$ are computed inductively as follows:

$$\begin{aligned} \top^{\mathfrak{B}} &= W, & \perp^{\mathfrak{B}} &= \emptyset, \\ \{\ell_i\}^{\mathfrak{B}} &= \{\ell_i^{\mathfrak{B}}\}, & (s_1 \sqcap s_2)^{\mathfrak{B}} &= s_1^{\mathfrak{B}} \cap s_2^{\mathfrak{B}}, \\ (s_1 \sqcup s_2)^{\mathfrak{B}} &= s_1^{\mathfrak{B}} \cup s_2^{\mathfrak{B}}, & (\neg s)^{\mathfrak{B}} &= W - s^{\mathfrak{B}}, \end{aligned}$$

$$\begin{aligned} (\exists^{<a} s)^{\mathfrak{B}} &= \{x \in W : \exists y \in W (d(x, y) \leq a \wedge y \in s^{\mathfrak{B}})\}, \\ (\exists^{\leq a} s)^{\mathfrak{B}} &= \{x \in W : \exists y \in W (d(x, y) < a \wedge y \in s^{\mathfrak{B}})\}, \\ (\forall^{\leq a} s)^{\mathfrak{B}} &= \{x \in W : \forall y \in W (d(x, y) \leq a \rightarrow y \in s^{\mathfrak{B}})\}, \\ (\forall^{<a} s)^{\mathfrak{B}} &= \{x \in W : \forall y \in W (d(x, y) < a \rightarrow y \in s^{\mathfrak{B}})\}, \\ (\exists s)^{\mathfrak{B}} &= \{x \in W : \exists y \in s^{\mathfrak{B}}\}, \\ (\forall s)^{\mathfrak{B}} &= \{x \in W : \forall y \in W y \in s^{\mathfrak{B}}\}. \end{aligned}$$

The *truth-relation* $\mathfrak{B} \models \varphi$, where φ is an $\mathcal{MS}_{\leq}^{\leq}[M]$ -formula, is defined in the natural way:

- $\mathfrak{B} \models \ell \in s$ iff $\ell^{\mathfrak{B}} \in s^{\mathfrak{B}}$,
- $\mathfrak{B} \models s_1 \doteq s_2$ iff $s_1^{\mathfrak{B}} = s_2^{\mathfrak{B}}$ (and $\mathfrak{B} \models s_1 \neq s_2$ iff $s_1^{\mathfrak{B}} \neq s_2^{\mathfrak{B}}$),
- $\mathfrak{B} \models \delta(k, \ell) \leq a$ iff $d(k^{\mathfrak{B}}, \ell^{\mathfrak{B}}) \leq a$,
- $\mathfrak{B} \models \delta(k, \ell) < a$ iff $d(k^{\mathfrak{B}}, \ell^{\mathfrak{B}}) < a$, and similar for $\geq$ and $>$.

An $\mathcal{MS}_{\leq}^{\leq}[M]$ -knowledge base is a finite set of $\mathcal{MS}_{\leq}^{\leq}[M]$ -formulas. A knowledge base Σ is called *consistent* (or *satisfiable*) if there exists an $\mathcal{MS}_{\leq}^{\leq}[M]$ -model $\mathfrak{B}$ such that $\mathfrak{B} \models \varphi$ for all $\varphi \in \Sigma$; in this case we write $\mathfrak{B} \models \Sigma$.

The language $\mathcal{MS}_{\leq}^{\leq}[M]$ is a proper fragment of the language $\mathcal{MS}[M]$ introduced in [Kutz *et al.*, 2003]. While $\mathcal{MS}[M]$ has the interesting property of being of the same expressive power as the two-variable fragment of first-order logic over metric spaces, reasoning in $\mathcal{MS}[M]$ is undecidable. [Kutz *et al.*, 2003] investigates in-depth the fragment $\mathcal{MS}^{\#}[M]$ of $\mathcal{MS}[M]$ with distance operators $\exists^{\leq a}$, $\exists^{>a}$ (‘for some object outside a circle of radius a ’) and their duals. In particular, various decidability results are obtained. The expressivity of $\mathcal{MS}_{\leq}^{\leq}[M]$ and $\mathcal{MS}^{\#}[M]$ is incomparable and the mathematics required to analyse these logics differs considerably. The tableau-algorithm presented here shows that $\mathcal{MS}_{\leq}^{\leq}[M]$ is probably more suitable for efficient reasoning than $\mathcal{MS}^{\#}[M]$.

Our first result establishes the decidability and the computational complexity of the satisfiability problem for $\mathcal{MS}_{\leq}^{\leq}[M]$ -knowledge bases in metric spaces:

Theorem 4. *The satisfiability problem for $\mathcal{MS}_{\leq}^{\leq}[M]$ -knowledge bases in arbitrary metric spaces is EXPTIME-complete (even if the parameters from M are coded in binary).*

Actually the same result can be proved for satisfiability in weaker distance spaces, say, the class of spaces which do not necessarily satisfy the triangular inequality (2) or the symmetry axiom (3). The proof of this and some other theorems in this paper are based on a rather sophisticated representation of distance spaces in the form of relational structures *à la* Kripke frames. The EXPTIME satisfiability-checking algorithm is basically an elimination procedure which iteratively deletes from the set of all 1-types over a sufficiently small set of terms all those that contain distance quantifiers without witnesses.

According to the next result, the logic of metric spaces introduced above has the finite model property, and so Theorem 4 holds for the class of finite metric spaces as well (the third equivalence actually follows immediately from the fact that any finite metric space is isometric to a subspace of some L_{∞}^n , $n \in \mathbb{N}^+$ [Matousek, 2002]).

Theorem 5. *The following conditions are equivalent for any $\mathcal{MS}_{\leq}^{\leq}[M]$ -knowledge base Σ :*

- Σ is satisfiable in a metric space;
- Σ is satisfiable in a finite metric space;
- Σ is satisfiable in a finite subspace of some space $L_{\infty}^n =$

$(\mathbb{R}^n, d_\infty^n)$, where

$$d_\infty^n((a_1, \dots, a_n), (b_1, \dots, b_n)) = \max\{|a_i - b_i| : 1 \leq i \leq n\}.$$

It turns out, however, that these positive results do not hold if the intended metric space is the two-dimensional Euclidean space or the class of its subspaces (this result is proved by a reduction of the undecidable $\mathbb{N} \times \mathbb{N}$ -tiling problem):

Theorem 6. (i) *Satisfiability of $\mathcal{MS}_{\leq}^{\leq}[\{1, 2\}]$ -knowledge bases in subspaces of $\mathbb{R} \times \mathbb{R}$ is undecidable.*

(ii) *The satisfiability problem for $\mathcal{MS}_{\leq}^{\leq}[\{1, 2\}]$ -knowledge bases in $\mathbb{R} \times \mathbb{R}$ is undecidable.*

Denote by $\mathcal{MS}^{\leq}[\mathbb{Q}^+]$ ($\mathcal{MS}^{\leq}[\mathbb{Q}^+]$) the fragment of $\mathcal{MS}_{\leq}^{\leq}[\mathbb{Q}^+]$ without the quantifiers $\exists^{\leq a}$ and $\forall^{\leq a}$ ($\exists^{< a}$ and $\forall^{< a}$, respectively).

Theorem 7. *Suppose that Σ is a knowledge base over $\mathcal{MS}^{\leq}[\mathbb{Q}^+]$ and Σ' results from Σ by replacing all occurrences of $\forall^{\leq a}$ with $\forall^{< a}$ and $\exists^{\leq a}$ with $\exists^{< a}$. Then Σ is satisfiable iff Σ' is satisfiable.*

3 The tableau algorithm

Our tableau algorithm is closely related to tableau-based decision procedures from the field of description logic [Baader and Sattler, 2000]. However, in addition to the standard methodology of description logic, we have to take care of the ‘arithmetical meaning’ of the quantifiers.

Suppose that a knowledge base Σ is given. To simplify presentation, we will assume that Σ is formulated in the language $\mathcal{MS}_{\leq}^{\leq}[\mathbb{N}^+]$ without universal quantifiers $\forall$ and $\exists$. (Knowledge bases containing rational numbers can be transformed into knowledge bases over $\mathbb{N}$ without effecting their satisfiability.)

Denote by $\text{trm}(\Sigma)$ the set of set terms occurring in Σ . Without loss of generality we may assume that Σ contains at least one parameter from $\mathbb{N}^+$ and one nominal $\{\ell_i\}$. A set term s is in *negation normal form* (NNF) if $\neg$ occurs in s only in front of set variables and set constants. Each set term can clearly be transformed into an equivalent one in NNF by pushing $\neg$ inwards (e.g., $\neg \exists^{< a} s$ is equivalent to $\forall^{< a} \neg s$). So we may assume all set terms to be in NNF. Distance atoms of $\mathcal{MS}_{\leq}^{\leq}[\mathbb{N}^+]$ can be expressed via nominals and distance quantifiers, e.g.,

$$\begin{aligned} \delta(k, \ell) < a & \text{ is equivalent to } k \in \exists^{< a} \{\ell\}, \\ \delta(k, \ell) \geq a & \text{ is equivalent to } k \in \neg \exists^{< a} \{\ell\}. \end{aligned}$$

So, from now on we will not be considering distance atoms at all. Moreover, without loss of generality we may assume that the language $\mathcal{MS}_{\leq}^{\leq}[\mathbb{N}^+]$ contains no formulas of the form $\ell \in s$. Indeed, every such formula is equivalent to $\{\ell\} \sqcup s \doteq s$. We can also assume that all equalities $s_1 \doteq s_2$ and their negations $s_1 \neq s_2$ are of the form $s \doteq \top$ and $s \neq \perp$, respectively. And by taking for each formula of the form $s \neq \perp$ in Σ a fresh location constant ℓ_s and replacing $s \neq \perp$ with $\ell_s \in s$, we can get rid of all negations of equalities.

To sum up: in what follows (without loss of generality) we confine ourselves to considering those knowledge bases

Σ over $\mathbb{N}^+$ that contain only equalities of the form $s \doteq \top$, with s being in NNF.

Let $\text{loc}(\Sigma)$ be the set of all location constants occurring in Σ . Define the *closure* $\text{cl}(\Sigma)$ of Σ to be the finite set of terms containing $\text{trm}(\Sigma)$ and, for all $a \in \mathbb{N}^+$,

$$\{\forall^{< a} s, \forall^{\leq a} s : \exists b \geq a (\{\forall^{\leq b} s, \forall^{< b} s\} \cap \text{trm}(\Sigma) \neq \emptyset)\}.$$

Denote by γ_Σ the maximal parameter which occurs in Σ and set

$$M[\Sigma] = \{1, 2, \dots, \gamma_\Sigma\}, \quad M[\Sigma]^- = \{a^- : a \in M[\Sigma]\},$$

where the a^- are not in $\mathbb{N}$.

The tableau algorithm we are about to describe operates on *constraint systems* $\mathcal{S} = \langle T, <, L, S \rangle$ for Σ , where $\langle T, < \rangle$ is a forest consisting of the disjoint union of finite intransitive trees $\langle T_\ell, <_\ell \rangle$ with roots $\ell \in \text{loc}(\Sigma)$, and L and S are *labelling functions* such that

- with each pair $x < y$, $x, y \in T$, the function L associates a number $L(\{x, y\})$ from $M[\Sigma] \cup M[\Sigma]^-$;
- with each $x \in T$ the function S associates a subset $S(x) \subseteq \text{cl}(\Sigma)$.

Roughly, the intuition behind this definition is as follows: T is the set of points of a metric space, L defines distances between some points in this space, and S defines values of set terms from $\text{cl}(\Sigma)$.

We start with the *initial constraint system* $\langle T_0, <_0, L_0, S_0 \rangle$ for Σ , where $T_0 = \text{loc}(\Sigma)$, $<_0 = L_0 = \emptyset$, and $S_0(\ell) = \{\{\ell\}\}$, for every $\ell \in \text{loc}(\Sigma)$. Then the initial constraint system is expanded by repeatedly applying (in any order) the *completion rules* in Fig. 1, where blocked nodes are defined as follows.

Denote by $<^+$ the transitive closure of $<$. We say that a node $x \in T$ is *directly blocked* by a node y if $y <^+ x$, $S(x) = S(y)$, but for no distinct $u <^+ x$ and $v <^+ x$ do we have $S(u) = S(v)$. The $<^+$ -successors of directly blocked nodes are called *indirectly blocked*. All directly or indirectly blocked nodes comprise the set of *blocked nodes*.

Constraint systems obtained by applying the completion rules to the initial constraint system for Σ will be called *constraint systems for Σ* . Such a constraint system $\mathcal{S}$ is said to be *complete* if one of the following conditions hold:

1. $\mathcal{S}$ contains a node x such that $X_i, \neg X_i \in S(x)$, for some set variable X_i ;
2. $\mathcal{S}$ contains a node x , such that $\{\ell_i\}, \neg\{\ell_i\} \in S(x)$ for some location constant $\{\ell_i\}$;
3. none of the rules in Fig. 1 is applicable to $\mathcal{S}$ (in the sense that no application of a rule can change the constraint system).

In cases 1 and 2 we say that the constraint system contains a *clash*.

Theorem 8. *The tableau algorithm, having started from any given constraint system Σ , always terminates (i.e., reaches a complete constraint system after finitely many² steps); it is sound (i.e., if there is a complete clash-free constraint system for Σ then Σ is consistent) and complete (i.e., if Σ is satisfiable, then the tableau algorithm returns a clash-free complete system).*

²Actually, at most $|\text{loc}(\Sigma)| \cdot |\text{trm}(\Sigma)|^{2|\text{cl}(\Sigma)|+1}$.

$R_{\sqcap}$	If $s_1 \sqcap s_2 \in S(x)$ and x is not indirectly blocked, then set $S(x) := S(x) \cup \{s_1, s_2\}$.
$R_{\sqcup}$	If $s_1 \sqcup s_2 \in S(x)$ and x is not indirectly blocked, then set either $S(x) := S(x) \cup \{s_1\}$ or $S(x) := S(x) \cup \{s_2\}$.
R_{nom}	If $\{\ell_i\} \in S(x) \cap S(y)$ and neither x nor y is indirectly blocked, then set $S(x) := S(x) \cup S(y)$ and $S(y) := S(x) \cup S(y)$.
$R_{=}$	If $s \doteq \top \in \Sigma$ and x is not indirectly blocked, then set $S(x) := S(x) \cup \{s\}$.
$R_{\forall}$	If $\forall^{<a}s \in S(x)$ or $\forall^{\leq a}s \in S(x)$ and x is not indirectly blocked, then set $S(x) := S(x) \cup \{s\} \cup \{\forall^{\leq b}s : b < a, b \in M[\Sigma]\}$ $\quad \cup \{\forall^{<b}s : b \leq a, b \in M[\Sigma]\}$.
$R_{\forall <}$	If $\forall^{<a}s \in S(x)$, $L(\{y, x\})$ is defined, and x is not indirectly blocked, then: if $L(\{y, x\}) = a^-$, then set $S(y) := \{s\} \cup S(y)$; if $L(\{y, x\}) = b < a$, then set $S(y) := \{\forall^{<a-b}s\} \cup S(y)$; if $L(\{y, x\}) = b^-$ with $b < a$, then set $S(y) := \{\forall^{\leq a-b}s\} \cup S(y)$.
$R_{\forall \leq}$	If $\forall^{\leq a}s \in S(x)$, $L(\{y, x\}) \in \{b, b^-\}$, and x is not indirectly blocked, then: if $b = a$, then set $S(y) := \{s\} \cup S(y)$; if $b < a$, then set $S(y) := \{\forall^{\leq a-b}s\} \cup S(y)$.
$R_{\exists <}$	If $\exists^{<a}s \in S(x)$, x is not blocked, and $L(\{z, x\}) \notin \{b : b < a\} \cup \{b^- : b \leq a\}$ for any z with $s \in S(z)$, then create a new node $y > x$ and set $L(\{x, y\}) := a^-$ and $S(y) := \{s\}$.
$R_{\exists \leq}$	If $\exists^{\leq a}s \in S(x)$, x is not blocked, and $L(\{z, x\}) \notin \{b : b \leq a\} \cup \{b^- : b \leq a\}$ for any z with $s \in S(z)$, then create a new node $y > x$ and set $L(\{x, y\}) := a$ and $S(y) := \{s\}$.

Figure 1: Tableau rules.

It is to be noted that the rule $R_{\forall}$ is rather inefficient as it stands. Actually, there is no need to add those $\forall^{\leq b}s$ and $\forall^{<b}s$, but then some other rules, as well as the blocking strategy, have to be changed in a subtle way. We show here this ‘inefficient variant’ to make the tableaux as transparent as possible. The worst case complexity (even of the more efficient variant) of the tableau algorithm does not match the EXPTIME upper bound of Theorem 4, which is actually a feature of all implemented tableau procedures in description logic.

4 Metric logic with numerical variables

Let us consider now the language $\mathcal{MS}_{\leq}^{\leq}[V]$ which is defined almost precisely as $\mathcal{MS}_{\leq}^{\leq}[M]$, with the only difference be-

ing that instead of parameters (numbers) from M we use *variables* from a countably infinite list $V = \{x_0, x_1, \dots\}$ ranging over $\mathbb{Q}^+$. Given an *assignment* $\alpha : V \rightarrow \mathbb{Q}^+$ and an $\mathcal{MS}_{\leq}^{\leq}[V]$ -knowledge base Σ , we denote by Σ^{α} the $\mathcal{MS}_{\leq}^{\leq}[\mathbb{Q}^+]$ -knowledge base which results from Σ by replacing each x_i with $\alpha(x_i)$.

Let Γ be a set of *constraints* for variables in V (say, a set of rational linear inequalities over V or polynomial equations). A knowledge base Σ over $\mathcal{MS}_{\leq}^{\leq}[V]$ is said to be *satisfiable relative to Γ* if there exists an assignment α such that

- the $\alpha(x_i)$ give a solution to Γ and
- Σ^{α} is satisfiable.

Here are two simple examples:

Example 9. (1) Γ consists of equalities $x_i = a_i$, where $a_i \in \mathbb{Q}^+$. Then a knowledge base Σ over $\mathcal{MS}_{\leq}^{\leq}[V]$ is satisfiable relative to Γ iff the knowledge base Σ^{α} is satisfiable for $\alpha : x_i \mapsto a_i$. Thus, in this case Σ^{α} is a usual $\mathcal{MS}_{\leq}^{\leq}[\mathbb{Q}^+]$ -knowledge base considered in the previous sections.

(2) Γ consists of inequalities $x_i < x_j$, so that no absolute values for distances are fixed, and to satisfy Σ we first have to find a solution (assignment) α to Γ and then check whether Σ^{α} is satisfiable.

To illustrate the expressive power of the new language, we will use the following notation: given set terms s_1 and s_2 , denote by $d(s_1, s_2) = x$ the set of two $\mathcal{MS}_{\leq}^{\leq}[V]$ -formulas $s_1 \sqcap \exists^x s_2 \neq \perp$ and $s_1 \sqcap \exists^{<x} s_2 \doteq \perp$ (saying that the distance between s_1 and s_2 is x). Then we can state that the distance between Russia and the UK is larger than the distance between Russia and Germany:

$$\Sigma = \{d(\text{Russia}, \text{UK}) = x, d(\text{Russia}, \text{Germany}) = y\} \text{ and } \Gamma = \{x > y\}.$$

Incomplete knowledge can also be expressed by using interval constraints $a_1 \leq x \leq a_2$, where x is a variable occurring in the knowledge base.

Of course, as long as algorithmic properties are not relevant, we can consider arbitrarily complex constraints Γ on the variables in V . Our main decidability result, however, holds for the case when Γ consists of a finite set of rational linear inequalities of the form

$$a_1 x_{i_1} + \dots + a_k x_{i_k} \geq b \quad \text{or} \quad a_1 x_{i_1} + \dots + a_k x_{i_k} > b.$$

Namely, we have the following:

Theorem 10. *It is decidable whether a given $\mathcal{MS}_{\leq}^{\leq}[V]$ -knowledge base Σ is satisfiable relative to a set of rational linear inequalities.*

The proof of Theorem 10 is based on a (rather straightforward) combination of a linear programming technique with the tableau algorithm presented above. The resulting decision procedure is extremely complex. We conjecture, however, that more sophisticated combinations with lower complexity can be developed. Here is one example:

Theorem 11. *Suppose that Γ consists of the ‘pure’ strict inequalities*

$$\{0 < x_1, x_1 < x_2, x_2 < x_3, \dots, x_{n-1} < x_n\}$$

and Σ is an $\mathcal{MS}_{\leq}^{\leq}[\{x_1, \dots, x_n\}]$ -knowledge base. Then the satisfiability problem for Σ relative to Γ is EXPTIME-complete.

(In fact Σ is satisfiable relative to Γ iff Σ^a is satisfiable, where $a(x_1) = 1$ and $a(x_i) = 1 + \frac{1}{n+2-i}$, for $1 < i \leq n$.)

5 Conclusion

We have introduced and investigated a ‘practical’ variant of the distance logics from [Sturm *et al.*, 2000; Kutz *et al.*, 2003]. Our main achievement is a tableau-based decision procedure similar to those implemented in working description logic systems like FACT and RACER, see [Baader *et al.*, 2003]. This similarity gives us some grounds to believe that a system based on this tableau procedure (which is currently being implemented) will be sufficiently fast for ‘realistic’ applications. As in spatial databases, however, a very important problem is to integrate the distance logic with other KR&R formalisms. Although in the intended applications the extensions of relevant predicates may strongly depend on the distance measure, usually *non-metric* attributes and relations also play a rather significant role (e.g., classifications of proteins, their topological properties). Without an *integration* of formalisms taking care of these different aspects of the problem domain one cannot imagine a system powerful enough for real-world applications. Fortunately, a framework for such an integration is already available. The terms of the language $\mathcal{MS}_{\leq}^{\leq}[\mathbb{Q}^+]$ are constructed from atomic predicates by means of distance and Boolean operators. This makes it possible to treat $\mathcal{MS}_{\leq}^{\leq}[\mathbb{Q}^+]$ as an abstract description system in the sense of [Baader *et al.*, 2002] and combine it in an algorithmically robust way with other effective KR&R formalisms, say, description logics or spatial RCC-8 via E-connections [Kutz *et al.*, 2002] or fusions [Baader *et al.*, 2002].

Acknowledgements

The work on this paper was supported by the DFG-grant Wo 583/3-3 and by U.K. EPSRC grants no. GR/R45369/01 and GR/R42474/01. Thanks are due to Anatoli Degtyarev, Ian Hodkinson, Carsten Lutz, and Oliver Kutz for stimulating discussions and remarks on the preliminary versions of the paper.

References

- [Adam and Gangopadhyay, 1997] N. Adam and A. Gangopadhyay. *Database Issues in Geographic Information Systems*. Kluwer, 1997.
- [Baader and Sattler, 2000] F. Baader and U. Sattler. Tableau algorithms for description logics. In R. Dyckhoff, editor, *Proceedings of Tableaux 2000*, volume 1847 of *Lecture Notes in Artificial Intelligence*, pages 1–18. Springer, 2000.
- [Baader *et al.*, 2002] F. Baader, C. Lutz, H. Sturm, and F. Wolter. Fusions of description logics and abstract description systems. *Journal of Artificial Intelligence Research*, 16:1–58, 2002.
- [Baader *et al.*, 2003] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.
- [Bozkaya and Özsoyoglu, 1999] T. Bozkaya and Z. Özsoyoglu. Indexing large metric spaces for similarity search queries. *TODS*, 24(3):361–404, 1999.
- [Clote and Backofen, 2000] P. Clote and R. Backofen. *Computational Molecular Biology*. John Wiley & Sons, 2000.
- [Cohn and Hazarika, 2001] A. Cohn and S. Hazarika. Qualitative spatial representation and reasoning: an overview. *Fundamenta Informaticae*, 43:2–32, 2001.
- [Dunham, 2003] M. Dunham. *Data Mining. Introductory and Advanced Topics*. Pearson Education, 2003.
- [Hjaltason and Samet, 1999] G. Hjaltason and H. Samet. Distance browsing in spatial databases. *ACM Transactions on Database Systems*, 24(2):265–318, 1999.
- [Horrocks, 1998] I. Horrocks. Using an expressive description logic: FACT or fiction? In A. Cohn, L. Schubert, and S. Shapiro, editors, *Proceedings of the 6th International Conference on Principles of Knowledge Representation and Reasoning*, pages 636–647, 1998.
- [Kutz *et al.*, 2002] O. Kutz, F. Wolter, and M. Zakharyashev. Connecting abstract description systems. In D. Fensel, F. Giunchiglia, D. McGuinness, and A. Williams, editors, *Proceedings of the 8th International Conference on Principles of Knowledge Representation and Reasoning*, pages 215–226. Morgan Kaufmann, 2002.
- [Kutz *et al.*, 2003] O. Kutz, H. Sturm, N.-Y. Suzuki, F. Wolter, and M. Zakharyashev. Logics of metric spaces. *ACM Transactions on Computational Logic*, 2003. In print.
- [Matousek, 2002] J. Matousek. *Lectures on Discrete Geometry*. Springer, 2002.
- [Randell *et al.*, 1992] D. Randell, Z. Cui, and A. Cohn. A spatial logic based on regions and connection. In *Proceedings of the 3rd International Conference on Knowledge Representation and Reasoning*, pages 165–176. Morgan Kaufmann, 1992.
- [Renz and Nebel, 1999] J. Renz and B. Nebel. On the complexity of qualitative spatial reasoning. *Artificial Intelligence*, 108:69–123, 1999.
- [Revesz, 2002] P. Revesz. *Introduction to Constraint Databases*. Springer Verlag, 2002.
- [Salton, 1989] G. Salton. *Automated Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*. Addison-Wesley, 1989.
- [Setubal and Meidanis, 1997] J. Setubal and J. Meidanis. *Introduction to Computational Molecular Biology*. PWS Publishing Company, 1997.
- [Sturm *et al.*, 2000] H. Sturm, N.-Y. Suzuki, F. Wolter, and M. Zakharyashev. Semi-qualitative reasoning about distances: a preliminary report. In *Logics in Artificial Intelligence. Proceedings of JELIA 2000, Malaga, Spain*, number 1919 in LNAI, pages 37–56. Springer-Verlag, 2000.

TEMPORAL REASONING

Incremental Tractable Reasoning about Qualitative Temporal Constraints

Alfonso Gerevini

DEA – Università degli studi di Brescia, via Branze 38, I-25123 Brescia, Italy
gerevini@ing.unibs.it

Abstract

In many applications of temporal reasoning we are interested in reasoning incrementally. In particular, given a CSP of temporal constraints and a new constraint, we want to *maintain* certain properties in the extended CSP (e.g., a solution), rather than recomputing them from scratch. The Point Algebra (PA) and the Interval Algebra (IA) are two well-known frameworks for qualitative temporal reasoning. Most of the existing algorithms for PA and the known tractable fragments of IA, such as ORD-Horn, has been designed for “static” reasoning. In this paper we study the incremental version of some fundamental problems of temporal reasoning, proposing new algorithms that amortize their complexity when processing a sequence of input constraints. After analyzing the role of path-consistency for incremental satisfiability, we propose algorithms for maintaining a solution of a CSP over either PA or ORD-Horn, and the minimal labels of a CSP over PA. Our algorithms improve the complexity of using existing techniques by a factor of $O(n)$ or $O(n^2)$, where n is the number of variables involved in the CSP.

1 Introduction

Constraint-based qualitative temporal reasoning is a widely studied area of AI in which the most prominent approaches are Allen’s Interval Algebra (IA) [1983] and Vilain and Kautz’s Point Algebra (PA) [1986]. Given a CSP of temporal constraints, fundamental reasoning problems are deciding satisfiability of the CSP, finding a solution, and computing its “minimal labels”. These problems are NP-hard for the full IA, while they are polynomial for PA and for several fragments of IA, such as Nebel & Bürckert [1995] ORD-Horn subalgebra, which is the unique maximal tractable subclass of IA containing all the basic relations.

In many applications of temporal reasoning we are interested in “dynamic” or *on-line* reasoning. For instance, given an initial (possibly empty) CSP of temporal constraints, whenever a new constraint is asserted, we want to *maintain* a solution of the CSP, rather than recomputing it from scratch using a “static” algorithm.

The design of algorithms for dynamic polynomial problems is an important research field in operation research and theoretical computer science (e.g., [Ausiello *et al.*, 1991]). A dynamic problem can be either *semi-dynamic* or *fully-dynamic*. In the semi-dynamic version we deal with information that is incrementally given (or decrementally retracted).

In the fully-dynamic version, we deal with both assertions and retractions. Typically, the worst-case performance of an algorithm for a dynamic problem is specified in terms of its *amortized complexity* for a single operation (assertion or retraction), that can be defined as the average runtime per operation over a worst-case *sequence* of operations [Tarjan, 1985].

The large majority of the algorithms that have been developed for reasoning within tractable classes of qualitative temporal constraints are static. In this paper we investigate the incremental version of some reasoning problems for PA and ORD-Horn, discussing the behavior of existing techniques and proposing new ones. Given a sequence s of $O(n^2)$ input constraints involving n variables, our incremental algorithms improve the total runtime required for processing s using existing techniques by a factor of $O(n)$ or $O(n^2)$.

In Section 2 we give the necessary background on PA and IA. In Section 3, after analyzing the use of a path-consistency algorithm for incremental satisfiability checking, we propose new incremental algorithms for finding a solution of a CSP over either PA or ORD-Horn, and for computing the minimal labels of a CSP over PA. Finally, in Section 4 we briefly discuss the fully-dynamic version of the problems considered, and we mention further results and future work.

2 Background on PA and IA

Vilain and Kautz’s Point Algebra [1986] consists of three basic relations between time points ($<$, $>$, $=$), all possible unions of them ($\leq$, $\geq$, $\neq$, and $?$, where $?$ is the universal relation), and of the empty relations ($\emptyset$). Allen’s Interval Algebra [1983] consists of thirteen basic relations between temporal intervals, all possible unions of these relations, and the empty relation. PA and IA are closed under the operations union ($\cup$), intersection ($\cap$), difference ($\setminus$), converse ($\smile$), and composition ($\circ$). The first four operations are defined in the standard way. The composition for two relations can be derived from the *transitivity tables* of the basic relations in PA and IA.

A *qualitative temporal constraint satisfaction problem* (or briefly *temporal CSP*) in our context is a set of constraints of the kind xRy , where x and y are either *point variables* or *interval variables*, and R is either a PA-relation (if x and y are point variables) or an IA-relation (otherwise).

Given a temporal CSP Θ over either PA or IA, a fundamental reasoning problem is deciding the *satisfiability* of Θ . Θ is satisfiable if and only if there exists a *solution* for Θ , i.e., an assignment of temporal values to the variables of Θ (rational numbers for point variables, pairs of rational numbers for interval variables) such that all the constraints in Θ are satis-

fied. The problem of deciding the satisfiability of Θ will be called PSAT, if Θ is over PA, and ISAT if Θ is over IA. The problem of finding a solution for Θ will be called PSOL, if Θ is over PA, and ISOL otherwise.

A related reasoning problem is finding a *scenario* that refines a given temporal CSP. A scenario of a CSP Θ is a satisfiable *refinement* of Θ , where the constraints between all pairs of variables are basic relations. A CSP Θ' is a refinement of Θ if and only if Θ' and Θ involve the same variables, and for every pair of variables (x, y) such that $xR'y \in \Theta'$ and $xRy \in \Theta$, $R' \subseteq R$.¹ Any scenario of a satisfiable temporal CSP over PA (IA) identifies a total order σ of the point (interval endpoint) variables of the CSP that is consistent with the constraints of the scenario, and in which point (interval endpoint) variables that must have the same interpretation are mapped to the same position. Clearly, from σ we can derive a solution for the CSP by just assigning to the points (endpoints) of σ numbers consistent with their relative order.²

Any temporal CSP Θ involving n variables can be processed using a cubic time algorithm that refines Θ to an equivalent *path consistent* CSP [Montanari, 1974]. A CSP is path consistent if for every subset of constraints involving three variables i, j , and k , the relation R_{ik} between i and k is stronger or equal than (i.e., is a subset of) the composition of R_{ij} and R_{jk} .

A temporal CSP is *minimal* if, for every pair of variables i, j , the relation R_{ik} between i and k is the strongest relations between i and k that is entailed by the CSP, i.e., for each basic r_{ij} relation in R_{ij} there exists a scenario for the CSP in which R_{ij} is refined to r_{ij} . We call the problem of computing the minimal CSP PMIN for PA, and IMIN for IA.

From a computational point of view, PA and IA have different properties. The reasoning problems introduced above are polynomial for PA, while they are NP-hard for IA [Vilain and Kautz, 1986]. In particular, PSAT and PSOL can be solved in $O(n^2)$ time using van Beek's method [1992], while PMIN requires $O(n^4)$ time [van Beek, 1992; Gerevini and Schubert, 1995]. These techniques use a graph-based representation of the CSP that in [Gerevini and Schubert, 1995] is called *temporally labeled graph* (TL-graph). A TL-graph is a graph with a set of labeled edges, where each edge connects a pair of distinct vertices v, w representing the point variables of the temporal CSP. The edges are either directed and labeled $\leq$ or $<$, or undirected and labeled $\neq$ ($=$ -constraints are represented by a pair of $\leq$ -constraints). van Beek's method for finding a scenario for a satisfiable CSP of PA-constraints is based on first identifying the strongly connected components (SCC) of the TL-graph representing the CSP [Cormen *et al.*, 1990] using only its $\leq$ -edges. Each SCC is then collapsed into a single vertex representing an equivalent class of variables that must be interpreted with the same temporal value. If we omit the $\neq$ -edges from the resultant graph, we obtain a directed acyclic graph that we call the induced *precedence*

graph of the CSP. From a topological sort of the vertices of this graph we can easily derive a scenario for the CSP.³

Enforcing path-consistency to a temporal CSP requires cubic time and is sufficient to decide PSAT [Ladkin and Maddux, 1988], but it does not solve PMIN. In order to solve PMIN for a CSP over PA, in addition to enforce path-consistency to it, we need to identify particular 4-variable constraint subsets of the path consistent CSP containing a $\leq$ -constraint that should be refined to " $<$ ". Such subsets are called *forbidden subgraphs* in [van Beek, 1992] and *$\neq$ -diamonds* in [Gerevini and Schubert, 1995].

Regarding IA, several tractable fragments have been identified. The Simple Interval Algebra (SIA) [van Beek, 1992; Ladkin and Maddux, 1988] is formed by the IA-relations that can be translated into a conjunction of PA-constraints between interval endpoints. All reasoning problems for a CSP Ω over SIA can be solved by applying the corresponding algorithms to the PA-translation of Ω . The most interesting tractable fragment of IA is Nebel and Bürckert's ORD-Horn class [1995]. ORD-Horn subsumes SIA and is the unique maximal tractable sub-algebra of IA containing all the basic relations. Each constraint C over ORD-Horn can be translated into a set of disjunctions of PA-constraints of the form $p = q$, $p \leq q$ or $p \neq q$, where p and q are endpoints of intervals in C , and at most one literal is of type " $=$ " or " $\leq$ ". Given a temporal CSP Ω over ORD-Horn, $\pi_1(\Omega)$ will denote the CSP of PA-constraints formed by the *unary* disjunctions (PA-constraints) in the point translation of all interval constraints in Ω .

Like PSAT, ISAT for ORD-Horn can be decided in cubic time by using a path-consistency algorithm. ISOL for ORD-Horn can be solved in square time, if the input CSP is known to be path consistent [Gerevini and Cristani, 1997], in cubic time otherwise [Ligozat, 1996].

In the rest of the paper, the incremental version of the reasoning problems for a temporal CSP is indicated by adding the prefix "I" to the name of the corresponding static problem, e.g., the incremental version of PSOL is I-PSOL. Without loss of generality, we assume that the initial temporal CSP already involves all variables, possibly constrained only by the universal relation. Moreover, our amortized complexity analysis is based on input sequences of length quadratic in the number of the variables. These assumptions are discussed in an extended version of the paper [Gerevini, 2003].

3 Incremental Tractable Reasoning

3.1 Path-consistency for I-PSAT and I-ISAT

Although van Beek's method for solving PSAT requires only quadratic time, when we consider the incremental version of this problem (I-PSAT), enforcing path-consistency turns out to be more efficient than a simple iterative application of van Beek's (static) algorithm. In fact, we can use a path-consistency algorithm like the one given in Figure 1 (PC), which was proposed in [Allen, 1983] and slightly reformulated and improved in [Vilain and Kautz, 1986; Bessière, 1996], to incrementally process a sequence of $O(n^2)$ PA-

¹Without loss of generality we assume that, if no information between x and y is provided, R is the *universal relation*, and that for every pair of variables (x, y) such that $xRy \in \Theta$, $yR^{\sim}x \in \Theta$.

²E.g., we assign an integer i to the points in the first position of σ , $i + 1$ to the points in the second position, etc. If we have a solution, we can derive a total order (and hence a scenario) in a trivial way.

³A topological sort for a DAG is a linear order σ of its vertices such that if v is a successor of w in the graph, then v precedes w in σ .

Algorithm: PC

Input: the matrix representation M of a temporal CSP over either PA or IA.

Output: **fail**, if enforcing path-consistency to the input CSP generates the empty relation; the matrix representation of a path consistent CSP equivalent to the input CSP, otherwise.

```

1.  $Q := \{(i, j) \mid i < j\}$ 
2. while  $Q \neq \emptyset$  do
3.   select and delete an element from  $Q$ ;
4.   for  $k \neq i, k \neq j$  do
5.     if  $\text{REVISE}(i, j, k)$  then
6.       if  $M[i, k] = \emptyset$  then return fail
7.       else add  $(i, k)$  to the end of  $Q$ ;
8.     if  $\text{REVISE}(k, i, j)$  then
9.       if  $M[k, j] = \emptyset$  then return fail
10.      else add  $(k, j)$  to the end of  $Q$ ;
11. return  $M$ .
```

Subroutine: $\text{REVISE}(i, k, j)$

```

1. if  $M[i, k]$  or  $M[k, j]$  is the universal relation then return false;
2.  $S := M[i, k] \circ M[k, j]$ ;
3. if  $M[i, j] \subseteq S$  then return false
4.  $M[i, j] := M[i, j] \cap S$ ;  $M[j, i] := M[j, i] \sim$ ;
5. return true;
```

Figure 1: A path-consistency algorithm for qualitative temporal constraints as formulated in [Bessière, 1996].

constraints in $O(n^3)$ total time (i.e., the amortized time complexity per input constraint is linear). This is better than reapplying van Beek’s algorithm after each constraint assertion, which requires $O(n^4)$ total time. Similarly, for any fragment of IA for which enforcing path-consistency is sufficient to decide satisfiability, we can use PC to solve I-ISAT in $O(n^3)$ total time, instead of $O(n^5)$. More precisely, let $\langle C_1, C_2, \dots, C_k \rangle$ be the sequence of the input constraints, Θ_0 an initial (possibly empty) satisfiable temporal CSP, and $\Theta_i = \Theta_0 \cup \{C_j \mid j \leq i\}$ ($i > 0$).⁴ To decide the satisfiability of each Θ_i , we consider three cases:

- (1) if the relation R of $C_i = xRy$ is weaker than the relation R' of the corresponding constraint in Θ_{i-1} , then clearly Θ_i is satisfiable, and no further processing is needed;
- (2) if $R \cap R'$ is the empty relation, then Θ_i is unsatisfiable and we return **fail**;
- (3) if none of the previous cases holds, then, first we revise the relation between x and y to $R' \cap R$ (and between y and x to $R' \sim \cap R \sim$), and then we run PC on the resultant revised CSP with step 1 modified so that Q initially contains only the item (x, y) .

We call this simple method *incremental path-consistency* (IPC). This technique has the following important property, that will be exploited by the algorithms in the next sections.⁵

Theorem 1 *IPC solves I-PSAT and I-ISAT for ORD-Horn in $O(n^3)$ total time, where n is the number of temporal variables.*

⁴ Θ_0 may already contain up to $O(n^2)$ constraints different from the universal relation. For CSPs over PA $k \leq 6 \cdot (n^2 - n)/2$, because we assume that there is no duplication in the input sequence, and there are at most 6 different input PA-constraints between any pair of variables (the empty and the universal relations are not valid input relations). Similarly, for CSPs over IA $k \leq (2^{13} - 2) \cdot (n^2 - n)/2$.

⁵For lack of space we omit proof details, which are available in [Gerevini, 2003].

Proof. Since a pair (x, y) enters into Q only when the relations between x and y is revised, and each relation R can be revised at most twice, if $R \in \text{PA}$, or at most 12 times, if $R \in \text{IA}$, it follows that the total time complexity is $O(n^3)$. $\square$

It is worth noting that, as pointed out by Bessière [Personal Communication, 1997], the $O(n^3)$ time complexity of Ligozat’s algorithm [1996] for finding a scenario of a path consistent CSP over ORD-Horn follows from Theorem 1.

3.2 An Algorithm for I-PSOL

For the incremental version of PSOL (I-PSOL) the simple application from scratch of van Beek’s static algorithm [1992] after each constraint assertion requires $O(n^4)$ total time.⁶

In this section we propose an algorithm for solving I-PSOL in $O(n)$ amortized time, i.e., the total time complexity for processing a sequence on $O(n^2)$ constraint assertions is $O(n^3)$. The algorithm is based on maintaining a topological sort $\mathcal{S}$ for the precedence graph of the input CSP in which the constraints are incrementally given. Let $\mathcal{S}_{i-1}$ be a topological sort for Σ_{i-1} (Σ_{i-1} is satisfiable). When we process a new constraint C_i we perform two main operations:

- (i) we check whether $\Sigma_i = \Sigma_{i-1} \cup \{C_i\}$ is satisfiable;
- (ii) if Σ_i is satisfiable, we update $\mathcal{S}_{i-1}$ to derive $\mathcal{S}_i$.

As we have seen, (i) can be solved in $O(n^3)$ total time by using IPC. Regarding (ii), there are two related difficulties to address: C_i induces an edge (or precedence constraint) in the precedence graph that invalidates $\mathcal{S}_{i-1}$; if the relation of C_i is “=” or “ $\leq$ ”, new equalities may be generated (i.e., in the TL-graph representation a new SCC may be generated, and the precedence graph should be revised accordingly). The algorithm INCREMENTAL-PA-SOL given in Figure 2 accomplishes (i) and (ii) taking these cases into account. The rest of this section is devoted to the description of the algorithm.

The current path consistent CSP is represented by a matrix M , while the current topological sort is represented by an array T such that $T[j] = v$ if and only if v is the j -th element of the sort. $\text{Ord}(v)$ denotes the position of v in the topological sort (i.e., the index of v in T). INCREMENTAL-PA-SOL uses a modification of IPC, called IPC-1, in which each variable has a flag (EQ) which is set to **false** before running INCREMENTAL-PA-SOL. The EQ -flag of a variable v is revised to **true** if the relation of the new input constraint is “=” and v is one of its variables, or if during the propagation of the new input constraint, the relation between v and *any* other variable is revised to “=”.

The second case is handled in IPC-1 by using an extension of the subroutine REVISE in which, if $M[i, j]$ is revised to “=”, then $EQ[i]$ and $EQ[j]$ are set to **true**. Clearly, this modification does not increase the complexity of the incremental path-consistency method.

If the relation of the new PA-constraint xRy is in $\{<, \leq\}$, then xRy may induce a precedence constraint $x \prec y$ for $\mathcal{S}_i$ that is not satisfied in $\mathcal{S}_{i-1}$. (Similarly for $R \in \{>, \geq\}$ inducing $y \prec x$.) This is the case when Σ_i is satisfiable, $\text{Ord}(x) > \text{Ord}(y)$ (or $\text{Ord}(y) > \text{Ord}(x)$), and xRy induces no new equality in Σ_i (i.e., the set E of step 3 is empty).

⁶A temporal CSP over PA may involve $O(n^2)$ constraints with relation different from “?”, and each of these can be revised at most twice by additional input constraints.

Algorithm: INCREMENTAL-PA-SOL

Input: the matrix representation M of the path consistent CSP of Σ_{i-1} ; a topological order T for the precedence graph G of Σ_{i-1} ; a new PA-constraint xRy .

Output: A solution for $\Sigma_i = \Sigma_{i-1} \cup \{xRy\}$, if Σ_i is satisfiable, fail otherwise.

1. $M := \text{IPC-1}(M, xRy)$;
2. **if** $M = \text{fail}$ **then return fail**;
3. $E := \text{set of the point variables } x \text{ such that } EQ[x] = \text{true}$
(E is a global variable used also in UPDATE-TSORT);
4. **if** E is empty **then**
5. **if** $R \in \{<, \leq\}$ and $\text{Ord}(x) > \text{Ord}(y)$ **then**
6. UPDATE-TSORT(T, x, y);
7. **if** $R \in \{>, \geq\}$ and $\text{Ord}(y) > \text{Ord}(x)$ **then**
8. UPDATE-TSORT(T, y, x);
9. **else** collapse the vertices of G in E to x ;
10. $S := \{v \mid v \notin E - \{x\} \wedge \text{Ord}(y) \leq \text{Ord}(v) \leq \text{Ord}(x)\}$;
11. $L := \text{list of the variables in } S \text{ topologically sorted using } G$;
12. **for** $j := \text{Ord}(y)$ **to** $\text{Ord}(x) - |S| + 1$ **do**
13. $T[j] := \text{Pop}(L)$; $\text{Ord}(T[j]) := j$;
14. **for** $j := \text{Ord}(x) - |E| + 2$ **to** $V(G)$ **do**
15. $T[j] := T[j + |E| - 1]$; $\text{Ord}[T[j]] := j$;
16. **return** an assignment to the variables in Σ_i consistent with T .

Subroutine: UPDATE-TSORT(T, x, y)

(The topological sort T for Σ_{i-1} is updated using the new precedence constraint $x \prec y$, and it becomes a topological sort for Σ_i .)

1. $\text{shift} := 1$;
2. $L := \text{a list containing only } y$;
3. **for** $i := \text{Ord}(y) + 1$ **to** $\text{Ord}(x)$ **do**
4. **if** $M[y, T[i]] \in \{<, \leq\}$ **then**
5. add $T[i]$ to the end of L ;
6. $\text{shift} := \text{shift} + 1$;
7. **else** $\text{Ord}(T[i]) := i - \text{shift}$;
8. $T[i - \text{shift}] := T[i]$;
9. **for** $i := \text{Ord}(x) - \text{shift} + 1$ **to** $\text{Ord}(x)$ **do**
10. $z := \text{Pop}(L)$; $T[i] := z$; $\text{Ord}(z) := i$;

Figure 2: Algorithm for I-PSOL. $V(G)$ denotes the current number of vertices in the precedence graph G . $\text{Pop}(L)$ returns the first element of L and removes it from L .

Under these conditions, the new constraint requires to update S_{i-1} , which is accomplished by steps 4–8. Specifically, if the induced precedence constraint is $x \prec y$, we run the subroutine UPDATE-TSORT(T, x, y), which revises the current topological sort (T) in the following way.⁷

UPDATE-TSORT considers each temporal variable v such that $\text{Ord}(y) \leq \text{Ord}(v) \leq \text{Ord}(x)$. (Notice that the only part of S_{i-1} that can be invalidated by $x \prec y$ regards the variables from $T[\text{Ord}(y)]$ to $T[\text{Ord}(x)]$.) If Σ_i entails $y \leq v$, then in S_i y (and hence also x) must precede v , because in the precedence graph there is a path from y (x) to v . Since by step 1 of the main algorithm M is the path consistent CSP of Σ_i , these entailment can be checked in constant time by simply looking up at the appropriate entries of M (step 4 of the subroutine). The v -variables that must follow x are added to the end of a list L that initially contains only y . Once all variables in the relevant part of S_{i-1} have been considered, those that are in L should be postponed, while the others can

⁷UPDATE-TSORT is similar to a technique used by Marchetti et al. [1993] in the context of an algorithm for detecting cycles in a DAG.

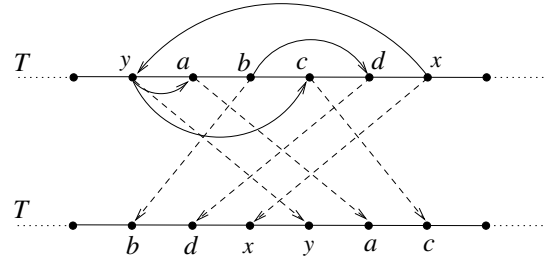


Figure 3: Example of an update of the topological order T with $x \prec y$ using UPDATE-TSORT(x, y, T). Directed solid arcs represent edges in the precedence graph of Σ_i . The variables preceding x in the current topological sort, and that in the revised sort must follow it, are y, a and c . When step 10 is executed the values of shift and L are 3 and (y, a, c) , respectively.

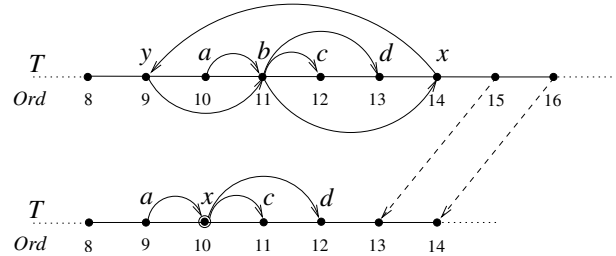


Figure 4: Example of revision of the topological sort T when a set E of variables become equal by adding $x \leq y$ to Σ_{i-1} ($E = \{x, y, b\}$). Directed solid arcs represent edges in the precedence graph of Σ_i before and after collapsing the variables in E to x . Dashed arrows indicate the shift performed by steps 14–15 of INCREMENTAL-PA-SOL.

be anticipated. In particular, if a variable is not added to L , we can shift it backward a number of positions equal to the number of items currently in L (which is given by the value of shift). This is done by steps 3–8 of the subroutine, while steps 9–10 move forward the variables in L to the positions of T that were occupied by the variables shifted backward. Notice that when UPDATE-TSORT(T, x, y) terminates, the relative order of the variables that entered into L , as well as of those that were shifted backward, are not changed in the revised topological sort. Figure 3 shows a simple example.

If xRy induces new equalities (E is not empty), then the variables in E , together with those possibly represented by them, form an equivalence class. Step 9 collapses the vertices in the precedence graph G of Σ_{i-1} corresponding to the variables in E into a single representative vertex x , transferring to x all incoming and outgoing edges of the collapsed vertices. In order to update T , we need now to reorder the variables from $T[\text{Ord}(y)]$ to $T[\text{Ord}(x)]$ that have not been collapsed. These variables are topologically ordered using G by step 11, and assigned to T in positions from $\text{Ord}(y)$ to $\text{Ord}(x) - |E| + 1$. Then, steps 13 and 14 *shrinks* T by shifting backward the variables appearing after x in T , starting from position $\text{Ord}(x) - |E| + 2$ and with a shift equal to the number of variables (vertices) eliminated from G (i.e., $|E| - 1$). Figure 4 gives an example illustrating steps 9–15.

Theorem 2 INCREMENTAL-PA-SOL solves I-PSOL in $O(n^3)$ total time, where n is the number of temporal variables.

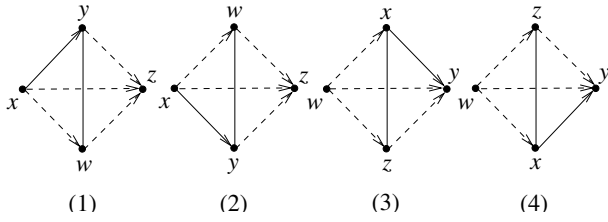


Figure 5: Types $\neq$ -diamonds that can be generated in $\hat{\Sigma}_i$ by the new constraint $x \leq y$. Directed (undirected) edges represent $\leq$ -constraints ($\neq$ -constraints). In the minimal CSP of Σ_i , for cases (1)-(2) R_{xz} should be $<$, for cases (3)-(4) R_{wy} should be $<$.

Proof (sketch). Correctness of the algorithm follows from Theorem 1 and the discussion in the description of the algorithm. By Theorem 1 step 1 requires $O(n^3)$ total time. The other most costly steps are 9, 11 requiring $O(n^2)$ time. They are performed at most $O(n)$ times for any input sequence. $\square$

3.3 An Algorithm for I-PMIN

The simple use of van Beek’s (static) algorithm for PMIN after each constraint assertion to solve I-PMIN requires $O(n^6)$ total time. In this section we propose an algorithm for solving I-PMIN, INCREMENTAL-PA-MIN, requiring $O(n^4)$ total time. This is the best possible worst-case complexity bound.

In order to maintain a CSP over PA minimal when a new PA-constraint xRy is given, in addition to enforce path-consistency, we need to identify and remove all possibly $\neq$ -diamonds that are generated (see Figure 5). Since $\neq$ -diamonds consist only of constraints of type $\leq$ and $\neq$, clearly a new constraint can generate additional $\neq$ -diamonds only if $R \cap R' \in \{=, \leq, \geq, \neq\}$, where R' is the relation between x and y in the minimal CSP of Σ_{i-1} .

Let $\bar{\Sigma}_{i-1}$ be minimal CSP of Σ_{i-1} , and $\hat{\Sigma}_i$ of the path consistent CSP of $\bar{\Sigma}_{i-1} \cup \{xRy\}$. If $R \cap R' \in \{=, \leq, \geq\}$ (and $R \subset R'$), then the new constraint and its propagation can generate new $\leq$ -constraints in $\hat{\Sigma}_i$. It is easy to see that any new possible $\neq$ -diamonds in $\hat{\Sigma}_i$ must involve one of these new $\leq$ -constraints. Hence, in order to make $\hat{\Sigma}_i$ minimal, it suffices to identify and eliminate all diamonds involving one of these constraints. Figure 5 shows the only four possible types of $\neq$ -diamonds that can be generated in $\hat{\Sigma}_i$ by the new constraint $x \leq y$. Note that cases (1) and (2) are symmetric and can be reduced to the same case (analogously for (3) and (4)).

If $R \cap R' = \neq$ (and $R \subset R'$), then the propagation of xRy cannot give rise to new $\neq$ -constraints, and hence in order to make $\hat{\Sigma}_i$ minimal, it suffices to identify and eliminate all new diamonds involving $x \neq y$.

Our algorithm for solving I-PMIN is given in Figure 6. It uses a modification of IPC, called IPC-2, which like IPC-1 is a revision of IPC to keep track of certain constraints generated by the propagation of xRy . IPC-2 adds all pairs of variables whose relation has been refined to $\leq$ to a list L . L is a global variable that is initialized by steps 1–2 of INCREMENTAL-PA-MIN to the list containing either (x, y) , (y, x) or no pair. A pair of variables is added to L by the subroutine REVISE-2 of IPC-2, a modification of REVISE in which after step 4 we perform the following additional steps to update L :

Algorithm: INCREMENTAL-PA-MIN

Input: the matrix representation M of the minimal CSP of Σ_{i-1} ; a new PA-constraint xRy .

Output: The matrix representation of the minimal CSP for $\Sigma_i = \Sigma_{i-1} \cup \{xRy\}$, if Σ_i is satisfiable, fail otherwise.

```

1. if  $R \cap M[x, y] = \leq \subset M[x, y]$  then  $L := ((x, y))$  else  $L := ()$ ;
2. if  $R \cap M[x, y] = \geq \subset M[x, y]$  then  $L := ((y, x))$  else  $L := ()$ ;
3. if  $R \cap M[x, y] = \neq \subset M[x, y]$  then  $N := \text{true}$  else  $N := \text{false}$ ;
4.  $M := \text{IPC-2}(M, xRy)$ ;
5. if  $M = \text{fail}$  then return fail;
6. for each  $(u, v) \in L$  do
7.   for each  $w, z$  such that  $w \neq z$  and  $w, z \neq u, v$  do
8.     if  $M[u, w] = M[v, z] = M[w, z] = \leq \wedge M[v, w] = \neq$ 
9.       then  $M[u, z] := <$ ;  $M[z, u] := >$ ;
10.    if  $M[w, u] = M[w, z] = M[z, v] = \leq \wedge M[u, z] = \neq$ 
11.      then  $M[w, v] := <$ ;  $M[v, w] := >$ ;
12. if  $N$  then
13.   for each  $w, z$  such that  $w \neq z$  and  $w, z \neq x, y$  do
14.     if  $M[w, x] = M[w, y] = M[x, z] = M[y, z] = \leq$ 
15.       then  $M[w, z] := <$ ;  $M[z, w] := >$ ;
16. return  $M$ .
```

Figure 6: Algorithm for solving I-PMIN. L is a global variable that is modified by IPC-2 as described in the text.

5. if $M[i, j] = \leq$ then add (i, j) to L
6. else if $M[j, i] = \leq$ then add (j, i) to L

Step 4 of INCREMENTAL-PA-MIN runs IPC-2 to (a) decide PSAT for Σ_i , (b) enforce path-consistency to M , and (c) update L with the new implied $\leq$ -constraints. If Σ_i is satisfiable, then steps 6–11 identify all new $\neq$ -diamonds that are generated in M by the new constraint $x \leq y$. These diamonds are eliminated by revising to $<$ or $>$ the relation of the “non-minimal” constraints (steps 9 & 11). Finally, steps 13–15 eliminate all new $\neq$ -diamonds involving $x \neq y$.

Theorem 3 INCREMENTAL-PA-MIN solves I-PMIN in $O(n^4)$ total time, where n is the number of temporal variables.

Proof (sketch). Since the algorithm eliminates all new $\neq$ -diamonds, correctness follows from Theorem 1 and fact that any path consistent CSP over PA with no $\neq$ -diamond is minimal [Gerevini and Schubert, 1995]. Complexity follows from Theorem 1 and the fact that the total number of pairs that can enter into L is $O(n^2)$. $\square$

3.4 An Algorithm for I-ISOL over ORD-Horn

Gerevini and Cristani [1997] showed that ISOL for a path consistent CSP Ω over ORD-Horn can be solved in $O(n^2)$ time by using a solution for the π_1 -translation of Ω . Figure 7 gives an algorithm solving I-ISOL for Ω which exploits this property and uses INCREMENTAL-PA-SOL. Our algorithm requires $O(n^3)$ total time, while the application from sketch of a known static algorithm each time a new constraint is asserted requires $O(n^5)$ total time. This cost can be reduced to $O(n^4)$ total time by using IPC combined with Gerevini & Cristani’s algorithm, which is still one order worse than the complexity of the algorithm presented here.

Our algorithm uses a modification of IPC, called IPC-3, which again is a revision of IPC to keep track of certain constraints generated by the propagation of the new input constraint xRy . Given the path consistent CSP $\hat{\Omega}$ of Ω_{i-1} , the

Algorithm: INCREMENTAL-ORDHORN-SOL

Input: the matrices M and M' representing the path consistent CSP $\hat{\Omega}$ of Ω_{i-1} , and the path consistent CSP of $\pi_1(\hat{\Omega})$, respectively; a topological order T for the precedence graph of $\pi_1(\hat{\Omega})$; a new interval constraint xRy .

Output: A solution for Ω_i , if Ω_i is satisfiable, fail otherwise.

1. **if** $(R \cap M[x, y]) \subset M[x, y]$ **then**
2. $P :=$ list formed by the constraints in $\pi_1(\{xRy\})$ **else** $P := \emptyset$;
3. $M := \text{IPC-3}(M, xRy)$;
4. **if** $M = \text{fail}$ **then return fail**;
5. **for each** $pSq \in P$ **do**
6. **if** $(S \cap M'[p, q]) \subset M'[p, q]$ **then** initialize the EQ -flags and
7. run steps 1–15 of INCREMENTAL-PA-SOL(M', T, pSq);
8. **return** an assignment to the interval endpoints in Ω_i consistent with T .

Figure 7: An algorithm solving I-ISOL for ORD-Horn.

path consistent CSP of $\pi_1(\hat{\Omega})$, a topological order T for the precedence graph of $\pi_1(\hat{\Omega})$, and a new interval constraint xRy , IPC-3 accomplishes the following tasks: (a) it decides ISAT for Ω_i ; (b) it updates $\hat{\Omega}$ to derive the path consistent CSP $\hat{\Omega}_i$ of $\Omega_i = \Omega_{i-1} \cup \{xRy\}$; (c) it derives a list P containing the PA-constraints in $\pi_1(\hat{\Omega}_i)$ that are *not* in $\pi_1(\hat{\Omega})$. P is a global variable that is initialized to the list formed by $\pi_1(\{xRy\})$, if xRy refines the current relation between x and y in $\hat{\Omega}$, and to the empty list otherwise (step 1). P is updated by the subroutine REVISE-3 of IPC-3, which is the same as REVISE in IPC, but with the important addition that, if $M[i, j]$ is revised, then the PA-constraints in $\pi_1(\{M[i, j]\})$ are added to P . Steps 5–7 incrementally update the path consistent CSP of $\pi_1(\hat{\Omega})$ (M') with the constraints in P , as well as the topological sort T of its precedence graph.

Theorem 4 INCREMENTAL-ORDHORN-SOL solves I-ISOL for ORD-Horn in $O(n^3)$ total time, where n is the number of temporal variables.

Proof (sketch). Correctness follows from Theorem 1, Theorem 2 and the results on ORD-Horn given in [Gerevini and Cristani, 1997]. Complexity follows from Theorems 1, 2 and the fact that for any sequence of $O(n^2)$ input constraints the total number of items entering into P is $O(n^2)$. $\square$

4 Conclusions and Future Work

We have presented some algorithms for incremental temporal reasoning in the context of PA, SIA and ORD-Horn, improving the complexity of existing techniques by a factor of $O(n)$ or $O(n^2)$. An interesting question is whether our techniques can be used for the decremental and fully-dynamic versions of the reasoning problems considered. Clearly, when constraints can only be removed from a CSP, satisfiability is preserved and no processing is needed. Similarly, for maintaining a solution. The decremental version of I-PMIN can be solved in $O(n^5)$ total time by an algorithm similar to INCREMENTAL-PA-MIN (here omitted for lack of space).

Performing an interleaved sequence of constraint assertions (refinements) and retractions appears to be intrinsically harder, and we conjecture that it requires algorithms with complexity higher than those presented here. For fully-dynamic problems our techniques can be used to amortize

complexity only for any *subsequence* of constraint assertions. E.g., consider an instance of the fully-dynamic version of PMIN in which we have k retractions during an input sequence of $O(n^2)$ assertions. When we process a retraction, we can recompute from scratch the minimal labels, while assertions are processed using our algorithm. The total time complexity is then $O(kn^4)$, which is still more efficient than applying van Beek's static $O(n^4)$ algorithm $O(kn^2)$ times.

While in this paper we have not addressed I-MIN for ORD-Horn, very recently we have proved that this problem can be solved in $O(n^6)$ total time. Future work includes incremental algorithms for other tractable classes [Krokhin *et al.*, to appear] and a deeper study of fully-dynamic problems.

References

- [Allen, 1983] J.F. Allen. Maintaining knowledge about temporal intervals. *Comm. of the ACM*, 26(1):832–843, 1983.
- [Ausiello *et al.*, 1991] G. Ausiello, G. Italiano, A. Marchetti-Spaccamela, and U. Nanni. Incremental algorithms for minimal length paths. *J. of Algorithms*, 12:615–638, 1991.
- [Bessière, 1996] C. Bessière. A simple way to improve path-consistency in Interval Algebra networks. *Proc. of AAAI-96*, 375–380, 1996.
- [Cormen *et al.*, 1990] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. The MIT Press, 1990.
- [Gerevini, 2003] A. Gerevini. Incremental Tractable Reasoning about Qualitative Temporal Constraints. *Technical Report*, DEA, University of Brescia, Italy, 2003.
- [Gerevini and Cristani, 1997] A. Gerevini and M. Cristani. On finding solutions in temporal constraint networks. *Proc. of IJCAI-97*, 1460–1465, 1997.
- [Gerevini and Schubert, 1995] A. Gerevini and L. Schubert. On computing the minimal labels in time point algebra networks. *Computational Intelligence*, 11(3):443–448, 1995.
- [Krokhin *et al.*, to appear] A. Krokhin, P. Jeavons, and P. Jonsson. The Tractable Subalgebras of Allen's Interval Algebra. *J. of ACM*, to appear.
- [Ladkin and Maddux, 1988] P. Ladkin, and R. Maddux. On binary constraint networks. TR, Kestrel Institute, 1988.
- [Ligozat, 1996] G. Ligozat. A new proof of tractability for ORD-Horn relations. *Proc. of AAAI-96*, 715–720, 1996.
- [Marchetti *et al.*, 1993] A. Marchetti-Spaccamela, U. Nanni, and H. Rohert. On-line graph algorithms for incremental compilation. *LNCS 790*, 113–151. Springer Verlag, 1993.
- [Montanari, 1974] U. Montanari. Networks of constraints: Fundamental properties and applications to picture processing. *Information Science*, 7(3):95–132, 1974.
- [Nebel and Bürckert, 1995] B. Nebel and H. Bürckert. Reasoning about temporal relations: A maximal tractable subclass of Allen's interval algebra. *J. of ACM*, 42(1), 1995.
- [Tarjan, 1985] R.E. Tarjan. Amortized computational complexity. *SIAM J. of Alg. and Dis. Meth.*, 6:306–318, 1985.
- [van Beek, 1992] P. van Beek. Reasoning about qualitative temporal information. *Artif. Intell.*, 58, 297–321, 1992.
- [Vilain and Kautz, 1986] M. Vilain and H.A. Kautz. Constraint propagation algorithms for temporal reasoning. *Proc. of AAAI-86*, 377–382, 1986.

Tractable Pareto Optimization of Temporal Preferences

Lina Khatib^{1,2} Paul Morris² Robert Morris²

1. Kestrel Technology

2. Computational Sciences Division

NASA Ames Research Center, MS 269-2

Moffett Field, CA 94035

Kristen Brent Venable

Department of Pure and Applied Mathematics

University of Padova

via G.Belzoni,7

35131 Padova, Italy

Abstract

This paper focuses on temporal constraint problems where the objective is to optimize a set of local preferences for when events occur. In previous work, a subclass of these problems has been formalized as a generalization of Temporal CSPs, and a tractable strategy for optimization has been proposed, where global optimality is defined as maximizing the minimum of the component preference values. This criterion for optimality, which we call “Weakest Link Optimization” (WLO), is known to have limited practical usefulness because solutions are compared only on the basis of their worst value; thus, there is no requirement to improve the other values. To address this limitation, we introduce a new algorithm that re-applies WLO iteratively in a way that leads to improvement of all the values. We show the value of this strategy by proving that, with suitable preference functions, the resulting solutions are Pareto Optimal.

1 Introduction

The notion of *softness* has been applied to either a constraint or planning goal, indicating that either can be satisfied to matters of degree. It is not hard to find applicable real world problems for such a notion. For example, in an earth orbiting spacecraft, sensitive instruments like imagers have *duty cycles*, which impose restrictions on the amount of use of the instrument. A duty cycle is typically a complex function based on both the expected lifetime of the instrument, as well as short term concerns such as the amount of heat it can be exposed to while turned on. Duty cycles impose constraints on the duration of the periods for which the instrument can be on, but it is natural to view this duration as flexible. For example, this restriction might be waived to capture an important event such as an active volcano. Thus, the flexibility of the duty cycle “softens” the constraint that the instrument cannot be on beyond a certain duration. Reasoning about soft constraints for planning or scheduling is for the purpose of finding a solution that satisfies the constraints to the highest degree possible.

For temporal reasoning problems, a simple method for evaluating the global temporal preference of a solution to

a Temporal CSP involving local temporal preferences was introduced in [Khatib *et al.*, 2001], based on maximizing the minimally preferred local preference for a time value. Because the locally minimally preferred assignment can be viewed as a sort of “weakest link” with respect to the global solution, we dub this method “weakest link optimization” (WLO), in the spirit of the television game show. WLO can be formalized using a generalization of Simple Temporal Problems (STPs), called STPs with Preferences (STPPs), that preserves the capability to tractably solve for solutions (with suitable preference functions associated with the temporal constraints). Unfortunately, as often occurs, this efficiency has a price. Specifically, WLO offers an insufficiently fine-grained method for comparing solutions, for it is based on a single value, viz., the “weakest link.” It is consequently easy to conceive of examples where WLO would accept intuitively inferior solutions because of this myopic focus. Although it is possible to consider more robust alternatives to a WLO strategy for evaluating solutions, it is not clear whether any of these methods would preserve the computational benefits of WLO. This impasse is the starting point of the work described in this paper.

We propose here to make WLO more robust by combining it with an iterative strategy for solving STPPs. The process involves repeatedly restricting temporal values for the weakest links, resetting their preference values, and applying the WLO procedure to the reduced problem that results from these changes. The intuition is a simple one, and we motivate this technique with an example from a Mars Rover planning domain. In Section 2, we summarize the soft constraint problem solver based on WLO introduced previously. We then illustrate in section 3 the deficiencies of WLO on a simple example, which also reveals the intuition underlying the proposed strategy for overcoming this deficiency. The main contribution of this paper is discussed in sections 4 and 5, which formalize this strategy and prove that any solution generated by an application of this strategy is in the set of Pareto optimal solutions for the original problem.

2 Reasoning about preferences with soft constraints

This section reviews the material first presented in [Khatib *et al.*, 2001]. There, a class of constrained optimization prob-

lems, called Temporal Constraint Satisfaction Problems with Preferences (TCSPPs), was first defined. A TCSPP is a generalization of classical TCSPs which allows for a representation of soft constraints. In classical TCSPs [Dechter *et al.*, 1991], a unary constraint over a variable X representing an event restricts the domain of X , representing its possible times of occurrence; the constraint is then shorthand for $(a_1 \leq X \leq b_1) \vee \dots \vee (a_n \leq X \leq b_n)$. Similarly, a binary constraint over X and Y restricts the values of the distance $Y - X$, in which case the constraint can be expressed as $(a_1 \leq Y - X \leq b_1) \vee \dots \vee (a_n \leq Y - X \leq b_n)$. A uniform, binary representation of all the constraints results from introducing a variable X_0 for the *beginning of time*, and recasting unary constraints as binary constraints involving the distance $X - X_0$.

A *soft temporal constraint* is a pair $\langle I, f \rangle$, where I is a set of intervals $\{[a, b], a \leq b\}$ of temporal values, and f is a function from $\bigcup I$ to a set A of values. Intuitively, f expresses local preferences for temporal values based on the value it assigns from A . For example, the soft constraint represented by $\langle \{[1, 3], [8, 12]\}, \min \rangle$ can be interpreted to mean that the temporal assignments must be selected from either of the intervals in the set, and the function $\min$ assigns a greater preference to smaller values. The cardinality of the set A , i.e., the number of distinct preference values, reflects the ability to discriminate among degrees of preference for temporal assignments. The class of TCSPPs in which each soft constraint consists of a single interval is called *Simple Temporal Problems with Preferences* (STPPs).

Local preferences combine to form global preferences for complete assignments. To formalize these operations, A can be structured in the form of a *c-semiring* [Bistarelli *et al.*, 1997]. A *semiring* is a tuple $\langle A, +, \times, \mathbf{0}, \mathbf{1} \rangle$ such that

- A is a set and $\mathbf{0}, \mathbf{1} \in A$;
- $+$, the additive operation, is commutative, associative and $\mathbf{0}$ is its identity element ($a + \mathbf{0} = a$);
- $\times$, the multiplicative operation, is associative, distributes over $+$, $\mathbf{1}$ is its identity element and $\mathbf{0}$ is its absorbing element ($a \times \mathbf{0} = \mathbf{0}$).

A *c-semiring* is a semiring in which $+$ is idempotent (i.e., $a + a = a, a \in A$), $\mathbf{1}$ is its absorbing element, and $\times$ is commutative. The semi-ring representation of operations on preference values is used as part of the proof of tractability for restricted sub-classes of TCSPP, which occurs below.

A *solution* to a TCSPP is a complete assignment to all the variables that satisfies the temporal constraints. An arbitrary assignment of values to variables has a *global preference value*, obtained by combining the local preference values using the semiring operations. A *c-semiring* induces a partial order relation $\leq_S$ over A to compare preference values of arbitrary assignments; $a \leq_S b$ can be read *b is more preferred than a*. Classical Temporal CSPs can be seen as a special case of TCSPP, with “soft” constraints that assign the “best” preference value to each element in the domain, and the “worst” value to everything else. The optimal solutions of a TCSPP are those solutions which have the best preference value in terms of the ordering $\leq_S$.

Weakest Link Optimization (WLO) is formalized via the semiring $S_{WLO} = \langle A, \max, \min, \mathbf{0}, \mathbf{1} \rangle$, that is, where for $a, b \in A$, $a + b = \max(a, b)$ and $a \times b = \min(a, b)$, and $\mathbf{1}$ ($\mathbf{0}$) is the best (worst) preference value. Given a solution t in a TCSPP with semiring S_{WLO} , let $T_{ij} = \langle I_{i,j}, f_{i,j} \rangle$ be a soft constraint over variables X_i, X_j and (v_i, v_j) be the projection of t over the values assigned to variables X_i and X_j (abbreviated as $(v_i, v_j) = t_{\downarrow X_i, X_j}$). The corresponding preference value given by f_{ij} is $f_{ij}(v_j - v_i)$, where $v_j - v_i \in I_{i,j}$. The global preference value of t , $val(t)$, is defined as $val(t) = \min\{f_{ij}(v_j - v_i) \mid (v_i, v_j) = t_{\downarrow X_i, X_j}\}$. Thus, a “weakest link value” for a solution t is any minimum $f_{ij}(v_j - v_i)$ that determines $val(t)$, and the *WLO-optimal solutions* to a problem are the ones that have a maximum weakest link value.

As with classical (binary) CSPs, TCSPPs can be arranged to form a network of nodes representing variables, and edges labeled with constraint information. Given a network of soft constraints, under certain restrictions on the properties of the semiring, it can be shown that local consistency techniques can be applied in polynomial time to find an equivalent minimal network in which the constraints are as explicit as possible. The restrictions that suffice for this result apply to

1. the “shape” of the preference functions used in the soft constraints;
2. the multiplicative operator $\times$ (it should be idempotent); and
3. the ordering of the preference values ($\leq_S$ must be a total ordering).

The class of restricted preference functions that suffice to guarantee that local consistency can be meaningfully applied to soft constraint networks is called *semi-convex*. This class includes linear, convex, and also some step functions. All of these functions have the property that if one draws a horizontal line anywhere in the Cartesian plane of the graph of the function, the set of X such that $f(X)$ is not below the line forms an interval. Semi-convexity is preserved under the operations performed by local consistency (intersection and composition). STPPs with semiring S_{WLO} can easily be seen to satisfy these restrictions.

The same restrictions that allow local consistency to be applied are sufficient to prove that STPPs can be solved tractably. Finding an optimal solution of the given STPP with semi-convex preference functions reduces to a two-step search process consisting of iteratively choosing a preference value, “chopping” every preference function at that point, then solving a STP defined by considering the interval of temporal values whose preference values lies above the chop line (semi-convexity ensures that there is a single interval above the chop point, hence that the problem is indeed an STP). Figure 1 illustrates the chopping process. It has been shown that the “highest” chop point that results in a solvable STP in fact produces an STP whose solutions are exactly the optimal solutions of the original STPP. Binary search can be used to select candidate chop points, making the technique for solving the STPP tractable. The second step, solving the induced STP, can be performed by transforming the graph associated

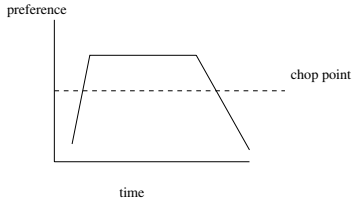


Figure 1: “Chopping” a semi-convex function

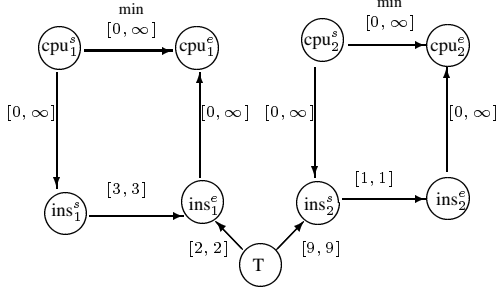


Figure 2: The STPP for the Rover Science Planning Problem where T is any timepoint

with this STP into a distance graph, then solving two single-source shortest path problems on the distance graph. (The solutions to these provide upper and lower time bounds for each event. If the problem has a solution, then for each event it is possible to arbitrarily pick a time within its time bounds, and find corresponding times for the other events such that the set of times for all the events satisfy the interval constraints.) For N nodes and E edges, the complexity of this phase is $O(EN)$ (using the Bellman-Ford algorithm [Cormen *et al.*, 1990]).

3 The problem with WLO

Formalized in this way, WLO offers what amounts to a coarse method for comparing solutions, one based on the minimal preference value over all the projections of the solutions to local preference functions. Consequently, the advice given to a temporal solver by WLO may be insufficient to find solutions that are intuitively more globally preferable. For example, consider the following simple Mars rover planning problem, illustrated in Figure 2. The rover has a sensing instrument and a CPU. There are two sensing events, of durations 3 time units and 1 time unit (indicated in the figure by the pairs of nodes labeled ins_1^s, ins_1^e and ins_2^s, ins_2^e respectively). There is a hard temporal constraint that the CPU be on while the instrument is on, as well as a soft constraint that the CPU should be on as little as possible, to conserve power. This constraint is expressed in the STPP as a function from temporal values indicating the duration that the CPU is on, to preference values. For simplicity, we assume that the preference function min on the CPU duration constraints is the negated identity function; i.e., $min(t) = -t$; thus higher preference values, i.e. shorter durations, are preferred. Because the CPU must be on at least as long as the sensing events, any globally preferred solution using WLO has preference value -3. The set of solu-

tions that have the optimal value includes solutions in which the CPU duration for the second sensing event varies from 1 to 3 time units. The fact that WLO is unable to discriminate between the global values of these solutions, despite the fact that the one with 1 time unit is obviously preferable to the others, is a clear limitation of WLO.

One way of formalizing this drawback of WLO is to observe that a WLO policy is not *Pareto Optimal*. To see this, we reformulate the set of preference functions of a STPP, $f_1, \dots, f_m$ as criteria requiring simultaneous optimization, and let $s = [t_1, \dots, t_n]$ and $s' = [t'_1, \dots, t'_m]$ be two solutions to a given STPP. s' dominates s if for each j , $f_j(t_j) \leq f_j(t'_j)$ and for some k , $f_k(t_k) < f_k(t'_k)$. In a Pareto optimization problem, the *Pareto optimal set* of solutions is the set of non-dominated solutions. Similarly, let the *WLO-optimal set* be the set of optimal solutions that result from applying the chopping technique for solving STPPs described above. Clearly, applying WLO to an STPP does not guarantee that the set of WLO-optimal solutions is a Pareto optimal set. In the rover planning problem, for example, suppose we consider only solutions where the CPU duration for the first sensing event is 3. Then the solution in which the CPU duration for the second sensing event is 1 time unit dominates the solution in which it is 2 time units, but both are WLO-optimal, since they have the same weakest link value.¹

Assuming that Pareto-optimality is a desirable objective in optimization, a reasonable response to this deficiency is to replace WLO with an alternative strategy for evaluating solution tuples. A natural, and more robust alternative evaluates solutions by summing the preference values, and ordering them based on preferences towards larger values. (This strategy would also ensure Pareto optimality, since every maximum sum solution is Pareto optimal.) This policy might be dubbed “utilitarian.” The main drawback to this alternative is that the ability to solve STPPs tractably is no longer apparent. The reason is that the formalization of utilitarianism as a semiring forces the multiplicative operator (in this case, sum), not to be idempotent (i.e., $a + a \neq a$), a condition required in the proof that a local consistency approach is applicable to the soft constraint reasoning problem.

Of course, it is still possible to apply a utilitarian framework for optimizing preferences, using either local search or a complete search strategy such as branch and bound. Rather than pursuing this direction of resolving the problems with WLO, we select another approach, based on an algorithm that interleaves flexible assignment with propagation using WLO.

4 An algorithm for Pareto Optimization

The proposed solution is based on the intuition that if a constraint solver using WLO could iteratively “ignore” the weakest link values (i.e. the values that contributed to the global solution evaluation) then it could eventually recognize solutions that dominate others in the Pareto sense. For example, in the Rover Planning problem illustrated earlier, if the weakest link value of the global solution could be “ignored,” the WLO solver could recognize that a solution with the CPU on

¹This phenomenon is often referred to in the literature as the “drowning effect.”

Inputs: STPP $P = (V, C)$

Output:

STP (V, C_P) whose solutions are Pareto optimal for P .

(1) $C_P = C$

(2) Do

(3) Solve (V, C_P) using WLO

(4) Delete all weakest link soft constraints from C_P

(5) For each deleted constraint $\langle [a, b], f \rangle$,

(6) add $\langle [a_{opt}, b_{opt}], f_{best} \rangle$ to C_P

(7) until (3)-(6) leave (V, C_P) unchanged

(8) Return (V, C_P)

Figure 3: STPP solver WLO+ returns a solution in the Pareto optimal set of solutions

for 1 time unit during the second instrument event is to be preferred to one where the CPU is on for 2 or 3 time units.

We formalize this intuition by a procedure wherein the original STPP is transformed by iteratively selecting what we shall refer to as a *weakest link constraint*, changing the constraint in such a way that it can effectively be “ignored,” and solving the transformed problem. A weakest link (soft) constraint for a WLO set of solutions is one in which the preference value of its duration in all the WLO solutions is the same as the chop level v of the optimal STP using WLO. For example, after applying WLO to the problem in Figure 2, the CPU duration constraint associated with the first sensing event will be a weakest link, since it now has a fixed preference value of -3. However, the CPU constraint for the second event will not be a weakest link since its preference value can still vary from -3 to -1.

We also define a weakest link constraint to be *open* if v is not the “best” preference value (i.e., $v < 1$, where 1 is the designated “best” value among the values in the semi-ring).

Formalizing the process of “ignoring” weakest link values is a two-step process of restricting the weakest links to their intervals of optimal temporal values, while eliminating their WLO restraining influence by resetting their preferences to a single, “best” value. Formally, the process consists of:

- Squeezing the temporal domain to include all and only those values which are optimally preferred; and
- Replacing the preference function by one that assigns the most preferred value (i.e. 1) to each element in the new domain.

The first step ensures that only the best temporal values are part of any solution, and the second step allows WLO to be re-applied to eliminate Pareto-dominated solutions from the remaining solution space.

The algorithm WLO+ (Figure 3) returns a Simple Temporal Problem (STP) whose solutions are contained in the WLO-optimal, Pareto-optimal solutions to the original STPP, P . Where C is a set of soft constraints, the STPP (V, C_P) is solved (step 3) using the chopping approach described earlier. In step 5, we denote the soft constraint that results from the two-step process described above as $\langle [a_{opt}, b_{opt}], f_{best} \rangle$, where $[a_{opt}, b_{opt}]$ is the interval of temporal values that are

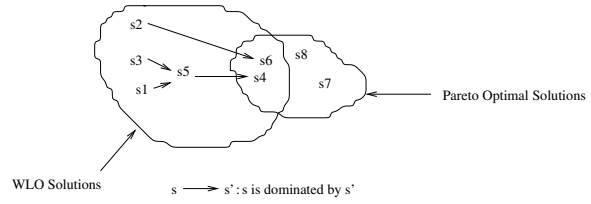


Figure 4: Relationships between Solution Spaces for STPPs that are WLO or Pareto Optimal

optimally preferred, and f_{best} is the preference function such that $f_{best}(v) = 1$ for any input value v . Notice that the run time of WLO+ is $O(|C|)$ times the time it takes to execute $Solve(V, C_P)$, which is a polynomial.

We now proceed to prove the main result, in two steps. In this section we assume the existence of weakest links at every iteration of the WLO+ algorithm, and show that the subset of solutions of the input STPP returned by WLO+ is contained in the intersection of WLO-optimal and Pareto-optimal solutions. In the next section we show that, given additional restrictions on the shape of the preference functions, such weakest links can be shown to always exist.

Given an STPP P , let $Sol_{WLO}(P)$ (resp. $Sol_{PAR}(P)$) be the set of WLO-optimal (respectively, Pareto-Optimal) solutions of P , and let $Sol_{WLO+}(P)$ be the set of solutions to P returned by WLO+. Then the result can be stated as follows.

Theorem 1 *If a weakest link constraint is found at each stage of WLO+, then $Sol_{WLO+}(P) \subseteq Sol_{WLO}(P) \cap Sol_{PAR}(P)$. Moreover, if P has any solution, then $Sol_{WLO+}(P)$ is nonempty.*

Proof:

First note that after an open weakest link is processed in steps (4) to (6), it will never again be an open weakest link (since its preference is reset to f_{best}). Since the theorem assumes a weakest link constraint is found at each stage of WLO+, the algorithm will terminate when the weakest link constraint is not open, i.e., when all the soft constraints in C_P have WLO preferences that equal the best (1) value.

Now assume $s \in Sol_{WLO+}(P)$. Since the first iteration reduces the set of solutions of (V, C_P) to $Sol_{WLO}(P)$, and each subsequent iteration either leaves the set unchanged or reduces it further, it follows that $s \in Sol_{WLO}(P)$. Now suppose $s \notin Sol_{PAR}(P)$. Then s must be dominated by a Pareto optimal solution s' . Let c be a soft constraint in C for which s' is superior to s . Thus, the preference value of the duration assigned by s to c cannot be 1. It follows that at some point during the course of the algorithm, c must become an open weakest link. Since s is in $Sol_{WLO+}(P)$, it survives until then, and so it must provide a value for c that is equal to the chop level. However, since s' dominates s , s' must also survive until then. But this contradicts the assumption that c is a weakest link constraint, since s' has a value greater than the WLO chop level. Hence, s is in $Sol_{PAR}(P)$, and so in $Sol_{WLO}(P) \cap Sol_{PAR}(P)$.

Next suppose the original STPP P has at least one solution. To see that $Sol_{WLO+}(P)$ is nonempty, observe that the modifications in steps (4) to (6), while stripping out solutions

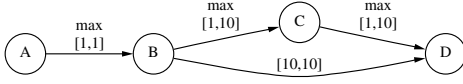


Figure 5: A unique WLO+ Solution.

that are not WLO optimal with respect to (V, C_P) , do retain all the WLO optimal solutions. Clearly, if there is any solution, there is a WLO optimal one. Thus, if the (V, C_P) in any iteration has a solution, the (V, C_P) in the next iteration will also have a solution. Since we are assuming the first $(V, C_P) = (V, C)$ has a solution, it follows by induction that $Sol_{WLO+}(P)$ is nonempty. $\square$

The theorem shows that it is possible to maintain the tractability of WLO-based optimization while overcoming some of the restrictions it imposes. In particular, it is possible to improve the quality of the flexible solutions generated within an STPP framework from being WLO optimal to being Pareto optimal.

Although the algorithm determines a nonempty set of solutions that are both WLO optimal and Pareto optimal, the set might not include all such solutions. Consider the example in figure 5. Assume the preference function for all soft constraints is given by $f(t) = t$, i.e., longer durations are preferred (signified by the *max* label on the edges). The WLO+ algorithm will retain a single solution where BC and CD are both 5. However, the solution where BC = 2 and CD = 8, which is excluded, is also both Pareto optimal and WLO optimal. (Note that AB, with a fixed value of 1, is the weakest link.)

Many optimization schemes seek what is known as *utilitarian* optimality, where the objective is to maximize the sum of the local preferences. However, the WLO+ solutions are not necessarily utilitarian optimal with respect to all solutions or even the WLO solutions. For example, in figure 5, if the preference function is $f(t) = t^2$, a utilitarian optimal WLO solution would be given by BC = 1 and CD = 9, but WLO+ will still return the solution where BC and CD are both 5.

We can summarize the position taken in this paper by saying that utilitarian strategies, while attractive in many ways, are apparently intractable. The WLO+ approach provides some of the same benefit at lower cost. For example, non-competing constraints are fully optimized by WLO+. For competing constraints, WLO+ tends to divide the preferences as equally as possible. In some applications, this might be more desirable than a utilitarian allocation.

5 Existence of Weakest Links

In this section we show that under suitable conditions, a weakest link constraint always exists. This involves a stronger requirement than for WLO: the preference functions must be convex, not merely semi-convex. This would include linear functions, cycloids, and upward-pointing parabolas, for example, but not Gaussian curves, or step functions. (Later on, we will see this requirement can be relaxed somewhat so that Gaussians can be permitted.)

Before proceeding, we note that while a solution s of an

STPP P is defined in terms of an assignment to each variable, it also determines a value $s(e)$ for each edge e , given by $s(e) = s(Y) - s(X)$ where X and Y are the start and end variables of e , respectively. We will use this notation in what follows.

Now consider any consistent STPP P . The *minimal network* [Dechter *et al.*, 1991] corresponding to P is another STPP P' . The constraints between any two points X and Y in P' are formed by intersecting the constraints induced by all possible paths between X and Y in P .

In the following, a preference function f is said to be *convex* if $\{< x, y > | y \leq f(x)\}$ is a convex set. The claim of the existence of weakest links can be stated as follows:

Theorem 2 *Let P be an STPP with continuous domains and convex preference functions. Then there will be at least one weakest link constraint for the WLO optimal set of solutions.*

Proof:

Consider the (minimal) STPP P_{opt} that corresponds to the optimal chopping level for P (as described in the WLO algorithm). Suppose there is no weakest link constraint. Then for each edge constraint e there is a solution s_e to P_{opt} such that $f(s_e(e)) > opt$, where f is the preference function for the edge.²

Let $\bar{s}$ be the average of all the s_e solutions, i.e.

$$\bar{s}(X) = 1/|E| \sum_{e \in E} s_e(X) \text{ for all } X \in V$$

where E is the set of edges and V is the set of temporal variables. By the linearity of the constraints, it is easy to show that $\bar{s}$ is also a solution to P_{opt} . For example, if $s_e(Y) - s_e(X) \geq a$ for all e , then $\bar{s}(Y) - \bar{s}(X) = 1/|E| \sum_{e \in E} (s_e(Y) - s_e(X)) \geq (1/|E|)|E|a = a$.

Since $\bar{s}$ is a solution, we must have $f(\bar{s}(e)) \geq opt$ for all edges e . Notice, however, that there must be some edge e such that $f(\bar{s}(e)) = opt$, otherwise $\bar{s}$ would be a solution with value greater than the optimal value. It follows that $\bar{s}(e) \neq s_e(e)$, since we already know that $f(s_e(e)) > opt$.

Thus, either $\bar{s}(e) < s_e(e)$ or $\bar{s}(e) > s_e(e)$. We consider only the case where $\bar{s}(e) < s_e(e)$. (The proof is similar in the other case.) Note also that $\min_{c \in E} s_c(e) < \bar{s}(e)$, since the minimum of a set of numbers can only be equal to the average if all the numbers are equal. It follows that $\min_{c \in E} s_c(e) < \bar{s}(e) < s_e(e)$. However, $f(\min_{c \in E} s_c(e)) \geq opt$, $f(\bar{s}(e)) = opt$, and $f(s_e(e)) > opt$. This violates the convexity of f , which establishes the theorem. $\square$

The operations in the WLO+ algorithm preserve the convexity property of the preference functions. Each stage of WLO+ repeats a WLO calculation. Thus, theorem 2 implies

Corollary 2.1 *Suppose P is an STPP with continuous domains and convex preference functions. Then a weakest link is found at each iteration of WLO+.*

An example of why the existence result does not apply more generally to semi-convex functions is found in figure 6. The

²To avoid excessive subscripting, we suppress the implied e subscript on f here and in what follows. In all cases, the applicable preference function will be clear from the context.

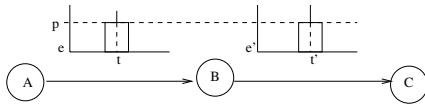


Figure 6: An STPP with no weakest link

STPP in the figure contains two semi-convex step-like functions with optimal preference values associated with durations t and t' . Assume the STPP is minimal, and that the assignment $e = t, e' = t'$ is inconsistent. Then the highest possible chop point is p , and no weakest link exists, i.e., for neither e nor e' is it the case that, for every solution s , p is the value returned by the preference function associated with that constraint for the duration assigned by s .

6 Discussion and Related Work

An examination of the proof of theorem 2 shows that the weakest link constraint exists under a somewhat less restrictive condition than convexity: it is enough, assuming semi-convexity, to require that plateaus (subintervals of non-zero length where the preference function is constant) can only occur at the global maximum of the preference function. This means, for example, that the theorem is applicable in principle to any semi-convex smooth function such as a Gaussian curve.

However, in the practical setting of a computer program where numbers are computed to a finite precision and continuous curves are approximated, some adjustments may need to be made. Note that a representation as a discretized step function does not satisfy the no-plateau condition. An alternative is to treat a discretized function as corresponding to a piecewise linear function where the linear segments join successive points on the discretized graph. Even there, the long tails of a Gaussian curve may get approximated by horizontal segments. However, generally we can trim the domain of the curve to eliminate the flat tails without excluding all the solutions. In that case, the discretized Gaussian is acceptable. (Note that figure 6 could be simulated by an example involving extreme Gaussians where the tails are essential for the solution.)

Note that preferences such as longest or shortest durations, or closest to a fixed time, which appear to be the most useful in practice, can be easily modeled within this framework.

WLO+ has been implemented and tested on randomly generated problems, where each semi-convex preference function is a quadratic $ax^2 + bx + c$, with randomly selected parameters and $a \leq 0$. We compared the best solution found after applying WLO+ with the quality of the earliest solution found using the chop solver, using the utilitarian measure of quality (i.e., summing preference values). An average improvement of between 6 and 10% was observed, depending on constraint density (more improvement on lower density problems). Future research will focus on the application of WLO+ to the rover science planning domain.

The results described here are clearly relevant to any effort whose objective is representing and reasoning about preferences and utility. A detailed survey of this vast literature is clearly beyond our scope; here we provide pointers to work that exhibits significant overlap. First, the idea of extending CSPs to solve multi-criteria optimization problems is proposed in [Torrens and Faltings, 2002]; this work also uses Pareto-optimality as a criterion for ordering solutions. Second, the idea of applying the notion of degrees of satisfaction to solving temporal reasoning problems has been applied previously [Dubois and Prade, 1989]. Third, a number of graphical-based representations of local preferences have appeared; in [Bacchus and Grove, 1995], for example, an approach is taken based on drawing connections between preferences and probabilities, as expressed in a Bayesian network. Finally, for a survey of AI-based approaches to preferences and utility, with an emphasis on qualitative approaches, the reader is referred to [Doyle and Thomason, 1999].

7 Summary

This paper has presented a reformulation of problems in the optimization of temporal preferences using a generalization of Temporal CSPs. The practical context from which this effort arose is temporal decision-making in planning, where associated with domains representing temporal distances between events is a function expressing preferences for some temporal values over others. The work here extends previous work by overcoming limitations in the approach that arose when considerations of efficiency in reasoning with preferences resulted in coarseness in the evaluation procedure for global temporal assignments.

References

- [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, March 1997.
- [Bacchus and Grove, 1995] F. Bacchus and A. Grove. Graphical Models of Preferences and Utility. *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, 3–10, Morgan Kaufmann, 1999.
- [Cormen et al., 1990] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. MIT press, Cambridge, MA, 1990.
- [Dechter et al., 1991] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49, 61–95, 1991.
- [Doyle and Thomason, 1999] J. Doyle and R. Thomason. Background to Qualitative Decision Theory. *AI Magazine*, 55–68, Summer, 1999.
- [Dubois and Prade, 1989] D. Dubois and H. Prade. Processing Fuzzy Temporal Knowledge. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(4), 1989.
- [Khatib et al., 2001] L. Khatib, P. Morris, R. Morris and F. Rossi. Temporal Reasoning about Preferences. In *Proceedings of IJCAI-01*. Morgan Kaufman, 1995.
- [Torrens and Faltings, 2002] M. Torrens and B. Faltings. Using Soft CSPs for Approximating Pareto-Optimal Solution Sets. In *AAAI Workshop Proceedings Preferences in AI and CP: Symbolic Approaches*, AAAI Press, 2002.

Automatic Video Interpretation: A Novel Algorithm for Temporal Scenario Recognition

Van-Thinh VU, François BREMOND, Monique THONNAT

Project ORION of I.N.R.I.A. Sophia Antipolis,
2004 route des Lucioles, BP93-06902 Sophia Antipolis Cedex, France.
{Thinh.Vu,Francois.Bremond,Monique.Thonnat}@sophia.inria.fr
<http://www-sop.inria.fr/orion/orion-eng.html>

Abstract

This paper presents a new scenario recognition algorithm for Video Interpretation. We represent a scenario model by specifying the characters involved in the scenario, the sub-scenarios composing the scenario and the constraints combining the sub-scenarios. Various types of constraints can be used including spatio-temporal and logical constraints. In this paper, we focus on the performance of the recognition algorithm. Our goal is to propose an efficient algorithm for processing temporal constraints and combining several actors defined within the scenario. By efficient we mean that the recognition process is linear in function of the number of sub-scenarios and in most of the cases in function of the number of characters. To validate this algorithm in term of correctness, robustness and processing time in function of scenario and scene properties (e.g. number of persons in the scene), we have tested the algorithm on several videos of a bank branch and of an office, in on-line and off-line mode and on simulated data. We conclude by comparing our algorithm with the state of the art and showing how the definition of scenario models can influence the results of the real-time scenario recognition.

1 Introduction

A problem of current focus in cognitive vision is Automatic Video Interpretation. The goal is to develop a systematic methodology for the design, implementation and integration of cognitive vision systems for recognizing scenarios involved in a scene depicted by a video sequence. An Automatic Video Interpretation System as described in Figure 1, takes as input (1) a priori knowledge containing scenario models predefined by experts and the 3D geometric and semantic information of the observed environment and (2) video streams acquired by the camera(s). The output of the system is the set of recognized scenarios at each instant. In this paper, we focus on the module of scenario recognition. The scenario recognition module takes as input the a priori knowledge of

the scene and a stream of individuals tracked by a vision module.

To solve scenario recognition issues, we first propose a language to describe scenario models and second a temporal constraint resolution approach to recognize in real-time scenario occurrences. Our scenario representation is mainly based on the representation of [Vu *et al*, 2002] and inspired by the work of [Ghallab, 1996]. In this paper, we focus on the optimization of the recognition method. We first enhance the processing of temporal operators by pre-compiling scenario models to decompose them into simpler scenario models. By this way, the scenario recognition algorithm uses a linear search compared to an exponential search for non-compiled scenarios. Secondly, we propose a novel algorithm to recognize composed scenarios that takes advantages of the actors of its sub-scenarios when they are recognized instead of trying all combinations of actors as this is often the cases for similar state of the art algorithms.

We present in section 2 some related works. Our scenario representation is described in section 3. The recognition algorithm is detailed in section 4. We conclude our paper by showing experimental results to validate this new algorithm.

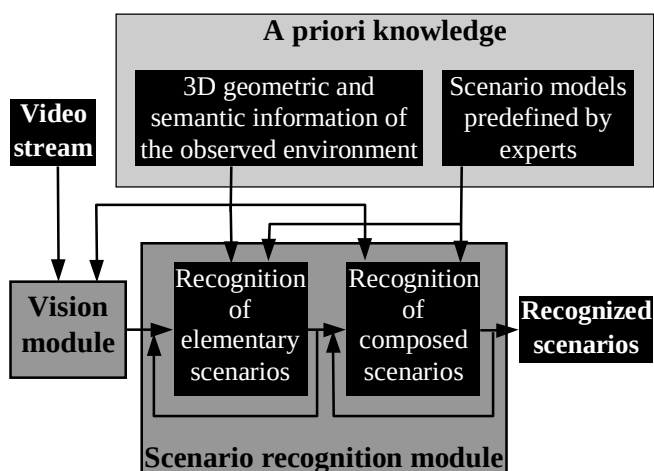


Figure 1. Overview of an Automatic Video Interpretation System.

2 Related works

Since the years 90s, a problem of focus in cognitive vision has been Automatic Video Interpretation. There are now several research units and companies defining new approaches to design systems that can understand human activities in dynamic scenes. Three main categories of approaches are used to recognize temporal scenarios based on (1) a *probabilistic/neural network* combining potentially recognized scenarios, (2) a *symbolic network* that Stores Totally Recognized Scenarios (STRS) and (3) a *symbolic network* that Stores Partially Recognized Scenarios (SPRS).

For the computer vision community, a natural approach consists in using a probabilistic/neural network [Oliver and Pentland, 2000]. The nodes of this network correspond usually to scenarios that are recognized at a given instant with a computed probability. For example, [Howell and Buxton, 2002] proposed an approach to recognize a scenario based on a neuronal network (time delay Radial Basis Function). [Hongeng *et al.*, 2000] proposed a scenario recognition method that uses concurrence Bayesian threads to estimate the likelihood of potential scenarios. Because video processing is very sensitive to noisy images, these probabilistic methods are useful to give an interpretation of the scene while taking into account the stochastic variations of the analysis. In our case, we use classical filtering techniques to get rid off these variations and to obtain coherent data that can be associated with symbolic values.

For the artificial intelligent community, a natural way to recognize a scenario is to use a symbolic network which nodes correspond usually to the boolean recognition of scenarios. For example, [Rota and Thonnat, 2000] used a declarative representation of scenarios defined as a set of spatio-temporal and logical constraints. They used a traditional constraint resolution technique to recognize scenarios. To reduce the processing time for the recognition step, they proposed to check the consistency of the constraint network using the AC4 algorithm. More recently, [Gerber *et al.*, 2002] defined a method to recognize a scenario based on a fuzzy temporal logic. In the same year, [Vu *et al.*, 2002] present an approach to optimize the temporal constraint resolution by ordering in time the sub-scenarios of the scenario to be recognized. The common characteristic of these approaches is to store all totally recognized scenarios (recognized in the past).

Another approach consists in using a symbolic network and to store partially recognized scenarios (to be recognized in the future). For example, [Ghallab, 1996] has used the terminology *chronicle* to express a *temporal scenario*. A chronicle is represented as a set of temporal constraints on time-stamped events. The recognition algorithm keeps and updates partial recognition of scenarios using the propagation of temporal constraints based on RETE algorithm. Their applications are dedicated to the control of turbines and telephonic networks. [Chleq and Thonnat, 1996] made an adaptation of temporal constraints propagation for video surveillance. In the same period, [Pinhanez and Bobick, 1997] have used Allen's

interval algebra to represent scenarios and have presented a specific algorithm to reduce its complexity.

All these techniques allow an efficient recognition of scenarios, but there are still some temporal constraints which cannot be processed. For example, most of these approaches require that the scenarios are bounded in time [Ghallab, 1996], or process temporal constraints and atemporal constraints in the same way [Rota and Thonnat, 2000].

We can distinguish two main categories of approaches to recognize a scenario based on a symbolic network: the STRS approaches recognize scenarios based on an analysis of scenarios recognized in the past [Rota and Thonnat, 2000; Vu *et al.*, 2002], whereas the SPRS approaches recognize scenarios based on an analysis of scenarios that can be recognized in the future [Ghallab, 1996]. The STRS approaches recognize a scenario by searching in the set of previously recognized scenarios a set of sub-scenarios matching the scenario model to be recognized. Thus, if the system fails to recognize a scenario, it will have to retry the same process (re-verify the same constraints) at the next instant, implying a costly processing time. A second problem is that STRS algorithms have to store and maintain all occurrences of previously recognized scenarios. The SPRS approaches recognize a scenario by predicting the expected scenarios to be recognized at the next instants. Thus, the scenarios have to be bounded in time to avoid the never ending expected scenarios. A second problem is that SPRS algorithms have to store and maintain all occurrences of partially recognized scenarios, implying a costly storing space.

The method presented in this article is a STRS approach taking advantages of the SPRS approaches. The objective is to reduce the processing time (1) when searching in the past (list of previously recognized scenarios) for an occurrence of a given scenario model and (2) when trying to recognize a scenario involving several actors by avoiding checking all combinations of actors.

3 Scenario Representation

Our goal is to make explicit all the knowledge necessary for the system to be able to recognize scenarios occurring in the scene. The description of this knowledge has to be declarative and intuitive (in natural terms), so that the experts of the application domain can easily define and modify it. Thus, the recognition process uses only the knowledge represented by experts through scenario models.

Let Ω be the set of scenario models and Φ be the set of scenario instances (recognized scenarios). For a scenario model $\omega \in \Omega$ and a scenario instance $\rho \in \Phi$, we note $\omega = \alpha(\rho)$ the scenario model of ρ and we note $\rho(\omega)$ the set of recognized scenarios of the model ω . A scenario is composed of four parts:

- a) $\alpha(\omega)$ is the set of *actor variables (characters)* involved in the scenario ω . $\alpha(\rho)$ corresponds to the actors. An actor can be a person tracked as a *mobile object* by a vision module or a *static object* of the observed environment like a chair.

- b) $\sigma(\omega)$ is the set of **sub scenarios** that compose ω . ω is an *elementary scenario* if $\sigma(\omega) = \emptyset$, if not, ω is called a *composed scenario*. Each sub-scenario is represented by a variable v . These variables are called *temporal variables* because their value is a recognized scenario defined on a temporal interval. We note $\rho(v)$ a scenario instance corresponding to the value of the temporal variable v , and $\alpha(v)$ the set of variables corresponding to the actors of $\rho(v)$.
- c) $\phi(\omega)$ is the set of sub-scenarios that should not occur during the recognition of the scenario ω . We call $\phi(\omega)$ the **forbidden sub scenarios**. $\alpha_\phi(\omega)$ is the set of *forbidden* actor variables involved in $\phi(\omega)$ but not already defined in $\alpha(\omega)$.
- $\phi(\omega)$ and $\alpha_\phi(\omega)$ are called the *forbidden variables*.
- d) $\kappa(\omega)$ is the set of **constraints** of ω . There are three types of constraints:
- the set of **temporal** constraints, noted $\kappa^T(\omega)$, on at least one variable of $\sigma(\omega)$ and not on any forbidden variable,
 - the set of **atemporal** constraints, noted $\kappa^A(\omega)$, on only $\alpha(\omega)$,
 - the set of **forbidden** constraints, noted $\kappa^F(\omega)$, on any forbidden variable.

The three subsets $\kappa^T(\omega)$, $\kappa^A(\omega)$ and $\kappa^F(\omega)$ constitute a partition of $\kappa(\omega)$. We use the operator "and" to link the constraints within a set of constraints. To use the operator "or", we propose to define two similar scenario models with different constraints. An elementary scenario is only composed of a set of characters and atemporal constraints.

Figure 2 represents a model of a composed scenario "Bank attack". This scenario involves two actors, a cashier and a robber.

In our representation, any scenario ω involves at least one person, and is defined on a time interval. An interval is represented by its starting and ending times noted $start(\omega)$ and $end(\omega)$. Defining the scenarios on a time interval is important for the experts to describe scenarios in a natural way.

```
Scenario(Bank_attack,
  Characters((cashier:Person), (robber:Person))
  SubScenarios(
    (cas_at_pos, inside_zone, cashier, "Back_Counter")
    (rob_enters, changes_zone, robber,
      "Entrance_zone", "Infront_Counter")
    (cas_at_safe, inside_zone, cashier, "Safe")
    (rob_at_safe, inside_zone, robber, "Safe")
  )
  ForbiddenSubScenarios(
    (any_in_branch, inside_zone, any_p, "Branch"))
  Constraints(
    Temporal(
      (rob_enters during cas_at_pos)
      (rob_enters before cas_at_safe)
      (cas_at_pos before cas_at_safe)
      (rob_enters before rob_at_safe)
      (rob_at_safe during cas_at_safe)
    )
    Atemporal(
      (cashier ≠ robber)
    )
    Forbidden(
      (any_p ≠ cashier) (any_p ≠ robber)
      (any_in_branch during rob_at_safe)))
)
```

Figure 2. One example of "Bank attack" scenario is composed of four steps: (1) the cashier is at his/her position behind the counter, (2) the robber enters the bank and moves toward the front of the counter then (3) both of them arrive at the safe door and (4) nobody else in the branch during the attack. The forbidden sub-scenario (step 4) is not necessary to recognize this "Bank attack" scenario. We have included this constraint just to show the possibility of modeling forbidden sub-scenarios.

4 Scenario Recognition

The scenario recognition process has to detect which scenario is happening from a stream of observed persons tracked by a vision module at each instant. The recognition process takes also as input the a priori knowledge of the scene and the scenario models. As defined in the previous section, there are two types of scenarios: elementary and composed.

4.1 Recognition of elementary scenarios

The algorithm to recognize an elementary scenario model ω_e consists in a loop of the selection of a set of actors then the verification of the corresponding atemporal constraints $\kappa^A(\omega_e)$ until all combinations of actors have been tested. Once a set of actors satisfies all constraints $\kappa^A(\omega_e)$, we say that the elementary scenario ω_e is *recognized* and we generate an elementary scenario *instance* ρ attached with the corresponding scenario model, the set of actors and the recognition time t . The scenario instance is then stored in the list of recognized scenarios. If at the previous instant, a scenario instance ρ' of same type (same model, same actors) was recognized on a time interval $[t_0, t-1]$, the two scenario instances are merged into a scenario instance that is recognized on the time interval $[t_0, t]$.

The selection of actors leads the recognition algorithm to an exponential combination in function of the number of actor variables. However, in practice, there are few actor variables in elementary scenario models, so the recognition algorithm is still real-time.

4.2 Compilation of composed scenarios

A composed scenario is a sequence of sub-scenarios ordered in time. Each sub-scenario corresponds to a temporal variable in the corresponding scenario model. The STRS algorithms of the state of the art perform at each instant an extensive search process among all possible scenarios and sub-scenarios leading to an exponential algorithm. For example, for a given scenario $\omega = (\omega_1 \text{ before } \omega_2 \text{ before } \omega_3)$; if a scenario instance ρ_{ω_3} of ω_3 has been recognized, it makes sense to try to recognize the main scenario ω . Therefore, the STRS algorithms will try all combinations of scenario instances $\rho(\omega_1)$, $\rho(\omega_2)$ with ρ_{ω_3} . We propose to decompose the scenario model into a set of simple scenario models containing at most two sub-scenarios.

To compile a predefined composed scenario model ω , we define three steps: (1) build a graph with the temporal variables $\sigma(\omega)$, (2) generate intermediate scenario models for ω and (3) link the generated intermediate scenario models based on the constraints $\kappa(\omega)$.

As proposed by [Ghallab, 1996], we first build a graph which nodes correspond to the temporal variables and which arcs correspond to the temporal constraints $\kappa^T(\omega)$. The arcs are oriented and are associated with a time interval corresponding to the time delay between the ending time of the two variables. For example, the constraint c_i between v_i , v_j is associated with an interval $[a, b]$ indi-

cating that v_j can end in the interval $[end(v_i)+a, end(v_i)+b]$. The constraint *before* is associated with $[1, \infty]$. After building the graph (called initial graph) with all temporal constraints between temporal variables $\sigma(\omega)$, we compute the equivalent complete graph and we check the graph consistency. These two graphs are equivalent because the only difference between them is the redundancy of some constraints. Then, we simplify the complete graph by removing unnecessary arcs to obtain the least constrained graph. For any triangle ABC of the graph, an arc AC is redundant (unnecessary), if the arcs AB and BC imply a stronger constraint. The simplified graph is equivalent to the two other graphs. The initial and simplified graphs for the scenario "Bank attack" (Figure 2) are shown on Figure 3. Thanks to the simplified graph, all the temporal variables $\sigma(\omega)$ are ordered by their ending time.

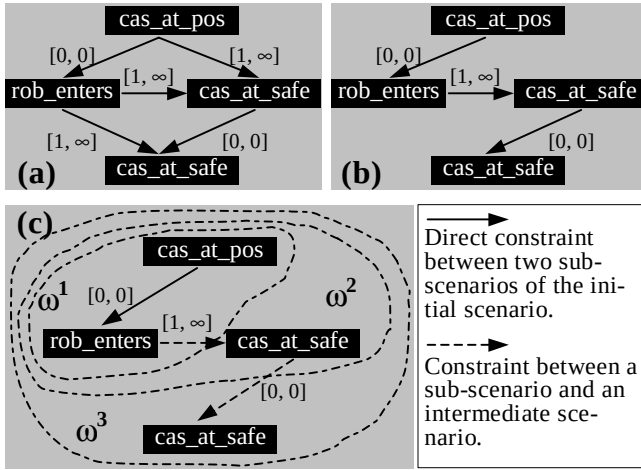


Figure 3. To compile the scenario "Bank attack", we build: (a) the graph with all temporal constraints $\kappa^T(\omega)$, (b) the simplified graph with only the necessary constraints and (c) the generated intermediate scenario models.

Second, we generate intermediate scenario models composed at most of two sub-scenarios. For each intermediate scenario model ω , we call *start* (noted $\xi(\omega)$) the first sub-scenario of ω ; and we call *termination* (noted $\tau(\omega)$) the second sub-scenario of ω .

As $\sigma(\omega) = (v_1, v_2, \dots, v_n)$ is a sequence of n ($n > 2$) ordered temporal variables, we generate $n-1$ intermediate models $\omega^1, \omega^2, \dots, \omega^{n-1}$ as followed:

- a) $\sigma(\omega^1) = (v_1, v_2)$ and $\sigma(\omega^i) = (v^i, v_{i+1})$ for $i > 1$, where v^i is a new temporal variable corresponding to the scenario model ω^{i-1} ,
- b) $\alpha(\omega^i) = \alpha(\xi(\omega^i)) \cup \alpha(\tau(\omega^i)) = \alpha(v_1) \cup \dots \cup \alpha(v_{i+1})$,
- c) $\phi(\omega^{n-1}) = \phi(\omega)$ and $\phi(\omega^i) = \emptyset$ for $i < n-1$, $\alpha_\phi(\omega^{n-1}) = \alpha_\phi(\omega)$ and $\alpha_\phi(\omega^i) = \emptyset$ for $i < n-1$,
- d) $\kappa^A(\omega^1) = \kappa^A(\omega) \cap \kappa(\alpha(\omega^1))$ and $\kappa^A(\omega^i) = \kappa^A(\omega) \cap \kappa(\alpha(\omega^i)) - \kappa(\alpha(\omega^{i-1}))$ for $i > 1$
 $\kappa^T(\omega^1)$ contains the constraints corresponding to the arcs entering v_{i+1} (i.e. $\tau(\omega^i)$) in the simplified graph.
 $\kappa^F(\omega^{n-1}) = \kappa^F(\omega)$ and $\kappa^F(\omega^i) = \emptyset$ for $i < n-1$.

The resulting scenario model ω^{n-1} is equivalent to the initial scenario model ω . This two scenarios have the same actor variables and equivalent set of constraints. The only difference is that the constraints of the scenario ω are verified at several intermediate levels corresponding to the intermediate scenario models as shown on Figure 4. Because any sequence of temporal variables can be ordered by their ending time, so any scenario model can be decomposed into intermediate scenarios containing only one or two temporal variables.

The recognition of compiled scenario models is described in the next section. The gain in processing time is due to the search algorithm: we just try several times to link two scenario instances instead of trying to link together a whole set of combinations of scenario instances.

```

Scenario(Bank_attack_1, #  $\omega^1$ 
  Characters((cashier:Person), (robber:Person))
  SubScenarios(
    (cas_at_pos, inside_zone, cashier, "Back_Counter")
    (rob_enters, changes_zone, robber, "Entrance_zone", "Infront_Counter"))
  Constraints((cas_at_pos during rob_enters)
    (cashier  $\neq$  robber) ))

Scenario(Bank_attack_2, #  $\omega^2$ 
  Characters((cashier:Person), (robber:Person))
  SubScenarios(
    (att_1, Bank_attack_1, cashier, robber)
    (cas_at_safe, inside_zone, cashier, "Safe" ))
  Constraints(
    ((start of att_1) before cas_at_safe) ))

Scenario(Bank_attack_3, #  $\omega^3$ 
  Characters((cashier:Person), (robber:Person))
  SubScenarios(
    (att_2, Bank_attack_2, cashier, robber)
    (rob_at_safe, inside_zone, robber, "Safe" ))
  ForbiddenSubScenarios(
    (any_in_branch, inside_zone, any_p, "Branch"))
  Constraints(
    (rob_at_safe during (termination of att_2))
    (any_p  $\neq$  cashier) (any_p  $\neq$  robber)
    (any_in_branch during rob_at_safe) ))

```

Figure 4. Three intermediate scenario models are generated for the compilation of the scenario model "Bank_attack", and the initial model is equivalent to "Bank_attack_3".

4.3 Recognition of composed scenarios

The recognition of a composed scenario model ω_c is triggered by a scenario template, which has been generated when the last sub-scenario ρ_t terminating ω_c has been recognized. The scenario template contains ω_c and the scenario instance ρ_t with its list of actors $\alpha(\rho_t)$ that partially instantiates $\alpha(\omega_c)$. As ω_c is composed of two sub-scenarios, only one sub-scenario ρ_s starting ω_c still needs to be found.

If such a scenario instance ρ_s has been previously recognized in the past, then we are able to finish instantiating the remaining actors of $\alpha(\omega_c)$. Thus, just a few combinations of actors need to be checked avoiding an exponential search.

The last step of the algorithm consists in verifying whether all temporal and atemporal constraints ($\kappa^T(\omega_c)$ and $\kappa^A(\omega_c)$) are satisfied with ρ_s and ρ_t . If one forbidden constraint of $\kappa^F(\omega_c)$ cannot be satisfied then the scenario ω_c is recognized and stored in the list of recognized scenarios.

4.4 Discussion

In the domain of temporal scenario recognition and among SPRS algorithms, the chronicle recognition algorithm [Ghallab, 1996] is one of the most popular. By storing partially recognized scenarios, it can speed up the whole recognition process. A partially recognized scenario corresponds to a prediction and enables to store all previous computations that do not need to be recomputed at the following instants. A main difference between the chronicle algorithm and our algorithm is that the chronicle algorithm has been developed to process scenarios defined with only one "actor" and can only recognize events detected at one time point. Thus, this algorithm is efficient for the configuration "mono-actor". However, in the configuration "multi-actors", the chronicle algorithm has to duplicate the partially recognized scenarios for each combination of actors not already instantiated. This can lead to an explosion of memory allocation and to an exponential search. Our algorithm is as efficient for the configuration "mono-actor" because we store the recognized scenarios which have been compiled. Moreover, it is efficient for the configuration "multi-actors" because the recognized scenarios do not need to be duplicated even if some actors are not instantiated. The worst case occurs with elementary scenarios because to recognize them at the current instant, all combinations of actors need to be checked. In real world applications, elementary scenarios do not contain many actor variables (less than 3) making the proposed algorithm sufficient to obtain an operational real time video interpretation system.

5 Experiments and results

To validate our recognition algorithm, we first integrated the algorithm with a vision module to obtain an operational interpretation system and then we have realized four types of tests: (1) on recorded videos taken in a bank branch and in two metro stations (one in Belgium and one in Spain) to verify if the algorithm can correctly recognize the predefined scenario models, (2) on live videos acquired on-line from cameras installed in an office, in a metro station and in a bank branch to verify if the algorithm can work robustly on a long time mode, (3) on recorded videos taken in a bank branch and on simulated data to study how the complexity of the algorithm depends on the scenario models (i.e. number of sub-scenarios and of actor variables) and (4) on simulated data to study how the complexity of the algorithm depends on the complexity of the scene (i.e. number of persons in the scene).

In the first experiment, we verify on recorded videos that the algorithm correctly recognizes several types of "Bank attack" scenarios and several types of "Vandalism against a ticket machine" scenarios in a metro station. The vandalism scenario involves in general two individuals, one looking around to check whether nobody is coming and the other one attempting several times to break the ticket machine. Table 1 shows that the predefined scenarios were correctly recognized in most of the

cases. The interpretation system fails to recognize some scenarios only in the cases when the vision module misses to detect the people in the scene. We have not detected any false alarm during all the experiment. The non-detection of false alarms can be explained by the fact that the scenarios are very constrained and there are unlikely to be recognized by error.

In the second experiment, we installed the interpretation system in an office and in a bank and we connected the system to one or two on-line cameras to acquire directly live videos. In this experiment, we used the bank scenarios and we slightly modified them to use them in the office. We ran the system in the bank for few hours and continuously during 4h in the office. As in the first experiment, the scenarios were most of the time correctly recognized, showing that the recognition algorithm can work reliably and robustly in real-time and in continuous mode.

	Number of tested sequences	Average number of persons/frame	Recognition rate (%)	Number of false alarms
Bank cam. 1	10	4	80	0
Bank cam. 2	1	2	100	0
Metro cam. 2	3	2	100	0

Table 1. The recognition of temporal scenarios in videos of a bank branch and of a metro station.

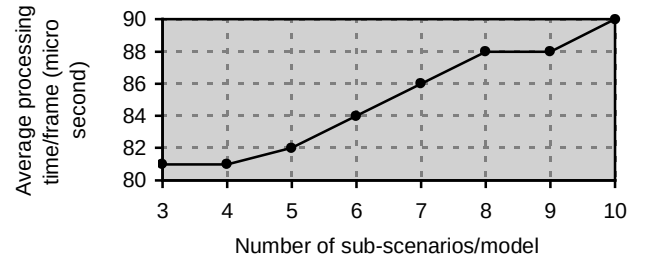


Figure 5. The processing time of the new algorithm is closely linear time in function of the number of sub-scenarios.

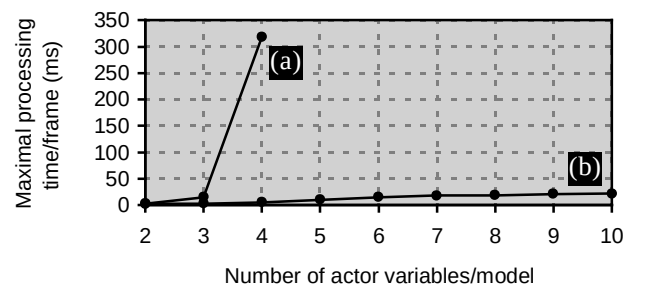


Figure 6. The processing time (a) of the old algorithm and (b) of the new algorithm depend on the number of actor variables of predefined scenario models.

In the third experiment, we studied the processing time of the recognition algorithm in function of the scenario models. First, we studied the processing time of the algorithm focusing on the resolution of temporal constraints. In this experiment (shown on Figure 5), we tested eight

configurations of scenario models: the first configuration is made of scenarios containing 3 sub-scenarios and the last configuration is made of scenarios containing 10 sub-scenarios. On the bank videos containing about 300 frames, we found that the processing time of the classical STRS algorithm is exponential in function of the number of sub-scenarios, whereas the processing time of our algorithm is closely linear in function of the number of sub-scenarios.

Second, we studied the processing time of the algorithm focusing on the number of actor variables of the scenario models. In this experiment (shown on Figure 6), we tested nine configurations of scenario models: the first configuration is made of scenarios involving 2 actor variables and the last configuration is made of scenarios involving 10 actor variables. To run the algorithm with enough actors, we simulated bank videos containing 35 persons. On these videos, we found that the processing time of the classical STRS algorithm is exponential in function of the number of actor variables, whereas the processing time of our algorithm is closely linear in function of the number of actor variables.

In the fourth experiment, we studied the processing time of the recognition algorithm in function of the scene. To have a continuous variation of the scene, we simulated the scene. We built a scene environment with eight zones of interest and ten pieces of equipment. We simulated the individuals evolving in the scene at each instant. In these simulated videos, the number of individuals changed from 30 up to 240. To verify if our algorithm can recognize in real-time the predefined scenarios, we measured the maximal processing time per frame. We found that, the maximal processing time for each frame is 100ms for a scene of 240 persons. We also found that the average processing time for each frame is closely linear in function of the number of persons. Figure 7 shows several tests of this experiment to illustrate how the processing time depends on the complexity of the scene.

With the fourth experiment, we can conclude that our recognition algorithm can recognize in real-time the predefined scenarios if the number of persons/frame is less than 240. All these tests are realized on a PC linux: CPU 700MHz, 320MB RAM.

6 Conclusion

In the literature, there are two categories of symbolic approaches to recognize temporal scenarios: the STRS algorithms reasoning on the past and the SPRS algorithms reasoning on the future. First, we have shown that the STRS algorithms recognize usually a scenario by performing an exponential combination search. Then, we have explained that even if our proposed algorithm is a STRS algorithm, it checks temporal constraints nevertheless by performing a linear search thanks to a step of pre-compilation of scenarios. Second, we have also shown that the SPRS algorithms have to try all combinations of actors to be able to recognize "multi-actors" scenarios. Thanks to the pre-compilation step this drawback for our algorithm is limited to elementary

scenarios. Thus, processing time can still be an issue depending on the complexity of scenarios. For these two reasons, the proposed algorithm enables the integrated video interpretation system to be real-time. Up to our knowledge, this video interpretation system is the first operational system able to recognize complex temporal scenarios.

Our future work consists in taking care of the errors and the uncertainty of the vision module. The goal is to be able to continue the interpretation of the videos even when the vision module cannot cope with the real world complexity and then to be able to recognize more complex scenarios.

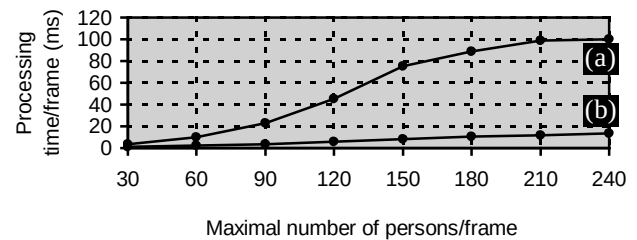


Figure 7. The (a) maximal and (b) average processing time/frame of the new algorithm depend on the number of detected persons.

References

- [Chleq and Thonnat, 1996] Nicolas Chleq and Monique Thonnat. *Real-time image sequence interpretation for video-surveillance applications*. International conference on Image Processing (ICIP'96). Proceeding IEEE ICIP'96. Vol 2. pp 801-804. Switzerland. 09/1996.
- [Ghallab, 1996] Malik Ghallab. *On Chronicles: Representation, On-line Recognition and Learning*. 5th International Conference on Principles of Knowledge Representation and Reasoning (KR'96), USA, 11/1996.
- [Gerber et al., 2002] R. Gerber, H. Nagel and H. Schreiber. *Deriving Textual Descriptions of Road Traffic Queues from Video Sequences*. The 15-th European Conference on Artificial Intelligence (ECAI'2002), Lyon, France, 21-26/07/2002.
- [Hongeng et al., 2000] S. Hongeng, F. Brémont and R. Nevatia. *Representation and Optimal Recognition of Human Activities*. In IEEE Proceedings of Computer Vision and Pattern Recognition, USA, 2000.
- [Howell and Buxton, 2002] A.J. Howell and H. Buxton. *Active vision techniques for visually mediated interaction*. Image and Vision Computing, 2002.
- [Oliver and Pentland, 2000] Nuria Oliver and Alex Pentland. *Graphical Models for Driver Behavior Recognition in a SmartCar*. Proceedings of IEEE Intl. Conference on Intelligent Vehicles 2000 Detroit. Michigan, 10/2000.
- [Pinhanez and Bobick, 1997] Claudio Pinhanez and Aaron Bobick. *Human Action Detection Using PNF Propagation of Temporal Constraints*. M.T.T Media Laboratory Perceptual Section Report No. 423, 04/1997.
- [Rota and Thonnat, 2000] Nathanaël Rota and Monique Thonnat. *Activity Recognition from Video Sequences using Declarative Models*. 14th European Conference on Artificial Intelligence (ECAI 2000), Berlin, W. Horn (ed.) IOS Press, Amsterdam, 20-25/08/2000.
- [Vu et al, 2002] Van-Thinh Vu, François Bremond and Monique Thonnat. *Temporal Constraints for Video Interpretation*. The 15-th European Conference on Artificial Intelligence (ECAI'2002), Lyon, France, 21-26/07/2002.

USER MODELING

Corpus-based, Statistical Goal Recognition

Nate Blaylock and James Allen

Department of Computer Science

University of Rochester

Rochester, New York, USA

{blaylock,james}@cs.rochester.edu

Abstract

Goal recognition for dialogue systems needs to be fast, make early predictions, and be portable. We present initial work which shows that using statistical, corpus-based methods to build goal recognizers may be a viable way to meet those needs. Our goal recognizer is trained on data from a *plan corpus* and then used to determine the agent's most likely goal based on that data. The algorithm is linear in the number of goals, and performs very well in terms of accuracy and early prediction. In addition, it is more easily portable to new domains as does not require a hand-crafted plan library.

1 Introduction

Much work has been done over the years in *plan recognition*, which is the task of inferring an agent's goal and plan based on observed actions. *Goal recognition* is a special case of plan recognition, in which only the goal is recognized.

Goal and plan recognition have been used in a variety of applications including intelligent user interfaces [Lesh *et al.*, 1999], dialogue systems [Carberry, 1990b; Allen *et al.*, 2000] and machine translation [Alexandersson, 1995].

We are especially interested in applying goal recognition to dialogue systems, in order to aid natural language understanding and intention recognition. We do not intend to use a goal recognizer to *directly* recognize communicative intentions (the goals behind a user's utterance); rather, we intend to use it to identify a user's domain goals to quickly help narrow the search space for more costly intention recognition routines (e.g., [Lochbaum, 1998; Chu-Carroll and Carberry, 2000]).

This application places several demands on our goal recognizer:

1. **Speed:** Dialogues happen in real-time, and the system is expected to understand a user's utterance and generate a response in a short amount of time.
2. **Early/partial prediction:** We need accurate goal predictions very early on in the exchange, as the system needs this information to better respond to a user's utterance. If full recognition is not immediately available,

the system needs at least partial information to allow it to act on the user's utterance.

3. **Portability:** We want to be able to rapidly port our dialogue system to new domains.

We present initial work in which we use corpus-based methods to build a goal recognizer. Our recognizer is fast (linear in the number of possible goals), makes early predictions, and is easier to port to new domains than many systems, as it does not require a hand-crafted domain plan library.

We first discuss the corpus-based approach we take in goal recognition. We then report on initial experiments that we have performed and discuss results. Finally, we comment on related work and then discuss future directions for our work.

2 The Corpus-based Approach

The use of statistical methods, based on corpora, has revolutionized the field of Natural Language Processing over the past 10+ years. The method is as follows: one uses a corpus of data to train a statistical model which is then used to make predictions on future data. Seemingly simple methods have yielded good results in many areas of NLP.¹

We apply a similar approach to the task of goal recognition. We use a *plan corpus* (a list of goals and the plans an agent executed to achieve them) to train statistical models which can predict an agent's goal based on an observed sequence of actions. As we show below, initially work shows several possible advantages over previous work on goal and plan recognition: recognition is fast (linear in the number of goals), robust (can handle unknown actions and plans) and does not require a hand-crafted plan library.

2.1 Recognition using N-gram Models

We define the task of goal recognition as follows: given an observed sequence of n actions so far $A_1, A_2, \dots, A_n$, (which, for compactness, we will represent as $A_{1,n}$), find the most likely goal G :

$$G^* = \operatorname{argmax} P(G|A_{1,n}) \quad (1)$$

Using Bayes' Rule, this becomes:

$$G^* = \operatorname{argmax} \frac{P(A_{1,n}|G)P(G)}{P(A_{1,n})} \quad (2)$$

¹For a good review, see [Manning and Schütze, 1999].

Goal #	Goal Description	Sessions
1	Find a file named 'core'	11
2	Find a file that contains the word 'motivating' and whose name ends in '.tex'	11
3	Find a machine that has low (<1.0) load; AND determine if Oren Etzioni is logged into the machine named chum	5
4	Compress all large files (>10,000 bytes) in the Testgrounds subdirectory tree	2
5	Compress all files in the directory named 'backups' [Don't use *]	3
6	Find a large file (>100,000 bytes) that hasn't been changed for over a month	8
7	Find a file that contains less than 20 words	8
8	Find a laser printer in Sieg Hall that has an active print job	6
9	Find a Sun on that has low (<1.0) load; AND determine if Dan Weld is active on the machine named chum	2
10	Find a file named oldpaper in neal/Testgrounds subdirectory	1
11	Find a file of length 4 in neal/Testgrounds subdirectory	1
12	See if Dan Weld is logged in to chum	1
Total		59

Table 1: Goals and their counts in the UNIX corpus

Since $P(A_{1,n})$ is constant in the argmax, we can drop it:

$$G^* = \operatorname{argmax} P(A_{1,n}|G)P(G) \quad (3)$$

Using the Chain Rule, we can rewrite this as:

$$G^* = \operatorname{argmax} P(A_n|A_{1,n-1}, G)P(A_{n-1}|A_{1,n-2}, G) \dots P(A_1|G)P(G) \quad (4)$$

These conditional distributions are very large and difficult to estimate, therefore, we make an n -gram assumption, i.e., we assume that an action A_i is only dependent on the goal G and the j actions preceding it ($A_{i-j,i-1}$). For a unigram model, we assume that A_i is independent of all other actions. In this case, our goal recognition equation becomes:

$$G^* = \operatorname{argmax} P(G) \prod_{i=1}^n P(A_i|G) \quad (5)$$

If we assume that A_i is independent of everything but G and A_{i-1} , we get a bigram model:

$$G^* = \operatorname{argmax} P(G) \prod_{i=2}^n P(A_i|A_{i-1}, G) \quad (6)$$

We estimate $P(G)$, $P(A_i|G)$ and $P(A_i|A_{i-1}, G)$ using a plan corpus. Then, for recognition, we first initialize our goal probabilities with $P(G)$. For each action we observe, we multiply each goal's score by the corresponding conditional probability.

This algorithm has the nice feature of compositionality — new observations produce conditional probabilities, which are simply multiplied with the previous predictions. This results in computational savings, since updates are linear in the number of goals. It also gives us a good model for early prediction (as desired for our dialogue system), since the model is based on actions observed so far and does not require all actions in the plan execution.

Goal Types	12
Goal Sessions	59
Action Types	22
Total Actions	412
Average Actions/Goal	7

Table 2: Statistics for the UNIX corpus

3 Experiments

3.1 The Plan Corpus

We performed several experiments using Lesh's UNIX plan corpus [Lesh, 1998]. The corpus was gathered from human UNIX users (CS undergraduates) at the University of Washington. Users were given a task in UNIX (a goal), and were instructed to solve it using a subset of UNIX commands (no pipes, no `awk`, etc.) The students' commands and results were recorded, as well as whether or not they successfully accomplished the goal. There were 59 successful goal sessions which involved 12 different goals. Table 1 shows the individual goals and the number of successful goal sessions for each.

We automatically removed unsuccessful commands, such as typos, from each execution. Remaining commands were stripped of arguments to a base command type form. This means our training set consisted only of action types (such as `ls`, `grep`, etc.), and did not consider flags or arguments. We hope to extend our model to incorporate that additional information in the future (see below).

Table 2 shows some relevant statistics from the resulting plan corpus. There were 12 possible goals and 22 different action types used in the goal sessions. On average, there were 7 actions per goal session.

We used cross-validation testing on 56 test cases.² Because

²The total number of goal sessions was 59, but sessions involving goals 10, 11 and 12 were not used as test data, since these were the only exemplars of their goals (see Table 1). These cases were

of the small size of the UNIX corpus, the training set was formed by removing only the test case from the corpus for each test case.

3.2 Evaluation Metrics

As pointed out by Lesh [Lesh, 1998], there is a lack of agreed-upon metrics and benchmarks for reporting results for plan and goal recognizers. We use the following metrics to report our results as they measure the attributes we are seeking for a goal recognizer (as described above). A test case is the set of actions executed for a goal. The actions are fed one by one to the recognizer, which makes predictions after every action. Each metric is for a single test case. For reporting multiple test cases, the metric is averaged across all cases.

- *Accuracy*: The number of correct predictions divided by the total number of observed actions.
- *Converged*: Whether or not the final prediction was correct (i.e., whether the recognizer *finish* with the correct answer).
- *Convergence point*: If the recognizer converged, at which point in the input it started giving only the correct answer. This is reported as an action number (i.e., after observing x actions); it is also reported as a percentage of the input according to the following equation:

$$\frac{\text{action\# converged on} - 1}{\text{\#actions in execution} - 1}$$

Intuitively, this shows how far through the plan execution the recognizer started (exclusively) predicting the right goal, with 0% meaning after the first action, and 100% after the final action. A low percentage indicates that the recognizer is making accurate early predictions.

3.3 Unigram model

For our first experiment, we trained unigram models on the data (based on Equation 5). To provide for unseen data, where $P(A_i|G)$ was 0, $P(A_i|G)$ was set to be a very small constant. This smoothing technique allows goal G to still remain possible, in case other evidence makes it likely, despite the fact that action A_i was not seen in relation to it in the training data.

With the unigram model, our recognizer achieved an accuracy of 63.1%, with 73.2% of the cases converging. For cases which converged, the average point of convergence was 23.1% through the input, or after observing an average of 2.4 actions.

Because a unigram model assumes independence of all actions, it is equivalent to treating actions as an unordered list.³ This assumption is partly to blame for the relatively low accuracy and convergence, since action ordering is important in the domain. We attempt to rectify this by using a bigram model below.

It is interesting to note the low point of convergence. This shows that, for the tests which did converge, they converged fairly quickly, i.e., the recognizer was able to tell fairly quickly what the goal was. This quick convergence seems

still used in training for other test cases, however, and were still candidate goals for recognition.

³This is a list and not a set since actions can be repeated.

partly due to the fact that some goals (such as goal 8, for example) were highly correlated with a single command (such as `lpq` in this case), which immediately boosted the probability of the correct goal.

3.4 Bigram model

In our next experiment, we used a bigram model (based on Equation 6), which encodes at least some ordering into the actions by considering both the current action and the preceding action.

We prepend a special `start` action to the front of each execution, which handles the special case of Equation 6 in which $n = 1$, i.e., when we've only seen one action so far. This also encodes information about which actions tend to begin executions, which may or may not be correlated to the goal.

For the case where $P(A_i|A_{i-1}, G)$ equals 0, i.e., the bigram was never seen in conjunction with the goal, we use a unigram back-off model which uses the estimation $P(A_i|A_{i-1}, G) \approx P(A_i|G)$. As with the unigram model above, if $P(A_i|G)$ equals 0, we smooth this with a small constant.

As expected, the bigram model performed much better than the unigram model, with an accuracy of 71.7% and with 83.9% of cases converging. For the tests cases which converged, they converged on average 22.3% through the input, or after 2.3 actions.

While accuracy and convergence were markedly better than the unigram model, the convergence point only improved slightly in the bigram model. However, this was already quite good, and may simply mean that we are approaching a limit on how soon the prediction can be made in this particular domain.

As for convergence, it is illustrative to look at a goal-by-goal breakdown of results. Table 3 shows convergence results for each goal type. For the 9 cases which didn't converge, 5 are goal 6, which is often misclassified as either goal 1 or goal 11. This seems mostly to be due to the fact that these three goals are very similar. Goal 1 is to find a file named 'core', goal 6 is to find a large file that hasn't been changed for over a month, and goal 11 is to find a file of length 4. All of these have to do with finding a file with a certain set of attributes, and in fact, the command `ls` can be used to learn all of these attributes about a file. Goal sessions for all three of these goals mostly consisted of the commands `cd` and `ls`.

3.5 Goal Abstractions

In our final experiment, we defined an abstraction hierarchy for goal types. The abstract types and goals they encompass are shown in Table 4.

As we discussed above, a dialogue system needs to be able to reason quickly, accurately and early about a user's goals and intentions. If the user's specific goal cannot be determined, a dialogue system can often use partial information in formulating a response to the user. These abstract goal classes represent that partial information. If we are unable to determine a specific goal, the system can perhaps determine what the user's abstract goal is, and this may be enough to formulate a response.

Goal#	Convergence	Competitors
1	10/11 (90.9%)	6:1
2	11/11 (100.0%)	none
3	4/5 (80.0%)	9:1
4	2/2 (100.0%)	none
5	2/3 (66.7%)	4:1
6	3/8 (37.5%)	1:2, 11:3
7	8/8 (100.0%)	none
8	6/6 (100.0%)	none
9	1/2 (50.0%)	3:1

Table 3: Convergence by goal in the bigram experiment

Abstract Goal	Goals which Instantiate
Find a file with attributes	1, 2, 6, 7, 10, 11
Find a machine with attributes	3, 9, 12
Compress files with attributes	4, 5
Find a printer with attributes	8

Table 4: Abstract goal types

The probability of an abstract goal is calculated by simply summing the probabilities of each of its children. For this experiment, we use the same bigram model as before and perform an additional summing at each step to calculate the most likely abstract goal.

The recognizer predicted abstract goals very well, with 91.0% accuracy and 100.0% of the cases converging. The average convergence point was 11.6% through the input, or after 1.7 actions. Table 5 summarizes the results of all three experiments.

It is important to note that abstract goal recognition does not occur in competition to, but rather in conjunction with the specific goal recognition. Of course, it is best to recognize the specific goal, which our bigram model seemed to do fairly well, but we can also recognize the abstract goal more quickly and more accurately. And, there may be many cases where it will be sufficient for the dialogue system to have just the abstract goal in order to help the user.

4 Discussion

These initial results are encouraging, but there still remain significant challenges for scaling up the system to sufficiently complex domains. In this section, we discuss our goal recognizer in light of the desiderata for dialogue systems we men-

	Unigram	Bigram	Abstract Goals
Accuracy	63.1%	71.8%	91.0%
Converged	73.2%	83.9%	100.0%
Conv. Point	23.1%	22.3%	11.6%
Conv. Action#	2.4	2.3	1.7

Table 5: Results of the experiments on the UNIX corpus

tioned above. We then discuss several challenges that remain.

4.1 Desiderata

Speed Because it involves a simple probability lookup, our recognizer is linear in the number of goals (and training it is linear in the amount of training data). Speed is a big advantage to this approach. By comparison, logic-based reasoning recognizers like [Kautz, 1991] are exponential in the size of the plan library.⁴ Several systems [Vilain, 1990; Lesh, 1998] improve on this time complexity, but at the expense of expressiveness.

Early/partial prediction Our recognizer was able to make correct predictions 22.3% (2.3 actions) through the input with 83.9% overall accuracy, and give correct abstract results 11.9% (1.7 actions) through the input with 100.0% overall accuracy. This early prediction is crucial to our domain of dialogue systems.

Most work does not report how early the recognizer makes correct predictions. Lesh [Lesh, 1998] simulates a task-completion agent, which, upon recognizing the user's goal, steps in to complete the task. He reports⁵ a convergence point of 29.9% (after 8.7 actions for an average plan length of 26.8 actions) for the task of searching for a printer with attributes (4 predicates) and a convergence point of 23.1% (after 4.9 actions for an average plan length of 17.9 actions) for a restricted cooking domain, both with 100.0% accuracy because Lesh uses a 'strict consistency' approach. For most domains, however, (like the full UNIX corpus), it is unclear if it could make such early predictions. Because it is based on 'strict consistency', a goal hypothesis must become logically impossible before it can be ruled out. In fact, even in for the domains he reports statistics in, the recognizer rarely recognizes the user's *actual* goal exclusively, rather, based on the set of possible goals, it tries to recognize a *sufficient* goal that covers all possible goals. As Lesh points out, many domains do not lend themselves to the prediction of sufficient goals.

Portability The portability of our goal recognizer depends on the existence of a plan corpus. If a plan corpus exists or can be created for the new domain (see section below), all we have to do is use it to train models for the new domain.⁶ On the other hand, most goal recognizers (e.g., [Vilain, 1990; Carberry, 1990a; Kautz, 1991; Charniak and Goldman, 1993; Paek and Horvitz, 2000]), require a complete, hand-crafted plan library in order to perform recognition, which can require a significant amount of knowledge engineering for each domain.

⁴Granted, these systems are performing plan recognition and not just goal recognition, which makes the comparison unfair. However, very little work has been done on goal recognition in its own right, so plan recognizers are all we have to compare against.

⁵Lesh reports the average length of plan with and without the task-completion agent, which we used to calculate these convergence points.

⁶Models could also be trained for different individuals within a single domain by using a user- (or group-) specific corpus. With other approaches, user-specific models have greatly improved recognition (e.g., [Lesh, 1998]).

Hong's recognizer [Hong, 2001] only requires knowledge of plan operators, not the library, but it is unable to make early predictions, as it usually does not make end-goal predictions until after it has seen the entire executed plan. Lesh [Lesh, 1998] requires only knowledge of plan operators and domain goal predicates.

4.2 Challenges

Domain size With only 12 goals and 22 action types, the UNIX domain is quite small, and it is unclear whether our recognizer would scale to larger domains. One immediate fault of our recognizer is that it does not handle parameterized goals. Each of the 12 goals above is treated as an atomic unit, and not as a goal schema instantiated with parameters. One straightforward way to handle parameters would be to treat a goal schema as an abstract goal, with each possible set of parameter instantiations as a separate, more specific goal. However, this would explode the number of goals and, in the case of 2 or more parameters, lead to a multiple-inheritance abstraction hierarchy, which is not supported by our current abstract goal score calculation model.

Hierarchical plans Another shortcoming of the current recognizer is that, although it handles goal abstraction, it does not handle hierarchical plans. Complex plans covering longer time-scales are less likely to be identifiable from a few observations alone (which tend to reflect more immediate subgoals). Ideally, we would want to recognize subgoals for partial results, even if we still cannot recognize the high-level goal.

Data collection In some domains (like operating systems), it may be possible to collect enough data from users to train a recognizer. In most domains, however, it will be infeasible to collect enough data on users solving goals in order to build effective statistical models. Furthermore, even if this data could be collected, the inner structure of the user's hierarchical plan would not be explicit from the data (i.e., we can only observe the primitive actions, not the subgoal structure that motivates the actions).

As the next step in our research, we plan to explore the use of AI planners to generate artificial plan corpora to be used for training. The approach we plan to take combines planning and Monte-Carlo simulation to generate plan corpora. The idea is to generate plans stochastically (allowing distributions over different aspects of the planning process, such as the goals, situations and action decompositions). By combining "context-free" Monte-Carlo simulation techniques with richly context-dependent planning algorithms, we hope to obtain a corpus that captures likely user behavior. In addition, this generated corpus has the big advantage that the subgoal hierarchy that generates the observed actions is also known.

5 Related Work

As mentioned above, [Vilain, 1990; Lesh, 1998] improve speed over [Kautz, 1991], but do so at the expense of expressiveness. Also, these goal recognizers are typically not able to make early predictions, as they are unable to distinguish

between consistent goals, even if one is more likely than the other.

There are several lines of research which incorporate probabilistic reasoning into plan and goal recognition. [Carberry, 1990a] and [Bauer, 1994] use Dempster-Shafer theory and [Charniak and Goldman, 1993], [Pynadath and Wellman, 1995], and [Paek and Horvitz, 2000] use Belief Networks to represent the likelihood of possible plans and goals to be attributed to the user. All of these methods, however, require a complete plan library as well as the assignment of probability distributions over the library.

[Appelt and Pollack, 1991] and [Goldman *et al.*, 1999] cast plan recognition as weighted abduction. However, this also requires a plan library and the acquisition of weights for abductive rules. Abduction is also computationally hard, and it is unclear whether such routines would be fast enough for complex domains.

Probably the closest work to ours is [Albrecht *et al.*, 1998], which uses a dynamic belief network (DBN) to do goal (quest) recognition in a multi-user dungeon (MUD) domain. The belief network takes into account actions, locations, and previous quest in recognizing the player's current quest. Similar to our own work, their model uses bigram independence assumptions for both actions and locations. Conditional probability distributions for the network are estimated from a corpus of MUD sessions.

Although our approaches are similar in terms of the use of corpora for training, there appear to be a few significant differences. Albrecht *et al.* encode state into their model (in the form of location and previous quest), whereas our current system considers only actions. Perhaps more significantly, they note that the bigram independence assumption is an inherent part of DBNs, since relaxing it (say, by using trigram or 4-gram models) would cause a state-space explosion for the DBN [Albrecht *et al.*, 1998, 12]. Relaxing the bigram assumption in our approach should have negligible effect on system speed, as probability updates are done by simple lookup, regardless of the size of the n -gram. To help alleviate space issues, we can store only the higher-order n -grams that we have good estimates for and use a backoff model similar to that used in our bigram model. We believe that higher-order n -grams will have significant predictive power, and plan to test their use in future research.

6 Conclusions and Future Work

We have presented our initial work on using statistical corpus-based techniques for goal recognition. Our recognizer is fast (linear time), does early and partial prediction, and can be ported to new domains more easily than many recognizers. We showed several initial experiments using the goal recognizer, in which we achieved high recognition rates with high accuracy early prediction.

There are several areas of future work that we are interested in. Several were alluded to above: scaling to larger domains, incorporating hierarchical plans, and using AI planners to generate artificial plan corpora.

In addition, we plan to collect a larger human-generated corpus in the UNIX domain. With more training data, we

would like to explore using trigram and 4-gram models as well as more advanced data mining techniques to train the statistical model.

Finally, as mentioned above, we would like to see how performance can be improved by training on a user-specific corpus. Similarly, we would like to see how different planners (say a reactive versus a deliberative planner) predict user behavior. Perhaps different planners could be used to model different domains (say domains with time pressure), or even different personality types.

Acknowledgments

We would like to thank Neal Lesh and Jun Hong for sharing their data with us. We would also like to thank the anonymous reviewers for their helpful comments.

This material is based upon work supported by Department of Education grant no. P200A000306; ONR research grant no. N00014-01-1-1015; and National Science Foundation grant no. E1A-0080124. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the above-mentioned organizations.

References

- [Albrecht *et al.*, 1998] David W. Albrecht, Ingrid Zukerman, and Ann E. Nicholson. Bayesian models for keyhole plan recognition in an adventure game. *User Modeling and User-Adapted Interaction*, 8:5–47, 1998.
- [Alexandersson, 1995] Jan Alexandersson. Plan recognition in VERBMobil. In M. Bauer, editor, *IJCAI 95 Workshop on The Next Generation of Plan Recognition Systems: Challenges for and Insight from Related Areas of AI (Working Notes)*, pages 2–7, Montreal, Canada, 1995.
- [Allen *et al.*, 2000] J. Allen, D. Byron, M. Dzikovska, G. Ferguson, L. Galescu, and A. Stent. An architecture for a generic dialogue shell. *Journal of Natural Language Engineering special issue on Best Practices in Spoken Language Dialogue Systems Engineering*, 6(3):1–16, December 2000.
- [Appelt and Pollack, 1991] Douglas E. Appelt and Martha E. Pollack. Weighted abduction for plan ascription. *User Modeling and User-Adapted Interaction*, 2:1–25, 1991.
- [Bauer, 1994] M. Bauer. Integrating probabilistic reasoning into plan recognition. In A. Cohn, editor, *Proceedings of the 11th European Conference on Artificial Intelligence*, pages 620–624, Amsterdam, Netherlands, August 1994. John Wiley & Sons.
- [Carberry, 1990a] Sandra Carberry. Incorporating default inferences into plan recognition. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 471–478, 1990.
- [Carberry, 1990b] Sandra Carberry. *Plan Recognition in Natural Language Dialogue*. ACL-MIT Press Series on Natural Language Processing. MIT Press, 1990.
- [Charniak and Goldman, 1993] Eugene Charniak and Robert P. Goldman. A Bayesian model of plan recognition. *Artificial Intelligence*, 64(1):53–79, 1993.
- [Chu-Carroll and Carberry, 2000] Jennifer Chu-Carroll and Sandra Carberry. Conflict resolution in collaborative planning dialogues. *International Journal of Human-Computer Studies*, 53(6):969–1015, 2000.
- [Goldman *et al.*, 1999] Robert P. Goldman, Christopher W. Geib, and Christopher A. Miller. A new model of plan recognition. In *Uncertainty in Artificial Intelligence: Proceedings of the Fifteenth Conference (UAI-1999)*, pages 245–254, San Francisco, CA, 1999. Morgan Kaufmann Publishers.
- [Hong, 2001] Jun Hong. Goal recognition through goal graph analysis. *Journal of Artificial Intelligence Research*, 15:1–30, 2001.
- [Kautz, 1991] Henry Kautz. A formal theory of plan recognition and its implementation. In J. Allen, H. Kautz, R. Pelavin, and J. Tenenbergh, editors, *Reasoning about Plans*, pages 69–125. Morgan Kaufman, San Mateo, CA, 1991.
- [Lesh *et al.*, 1999] Neal Lesh, Charles Rich, and Candace L. Sidner. Using plan recognition in human-computer collaboration. In *Proceedings of the Seventh International Conference on User Modeling*, Banff, Canada, June 1999. Springer-Verlag. Also available as MERL Tech Report TR-98-23.
- [Lesh, 1998] Neal Lesh. *Scalable and Adaptive Goal Recognition*. PhD thesis, University of Washington, 1998.
- [Lochbaum, 1998] Karen E. Lochbaum. A collaborative planning model of intentional structure. *Computational Linguistics*, 24(4):525–572, 1998.
- [Manning and Schütze, 1999] Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, Massachusetts, 1999.
- [Paek and Horvitz, 2000] Tim Paek and Eric Horvitz. Conversation as action under uncertainty. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence (UAI-2000)*, Stanford, CA, June 2000.
- [Pynadath and Wellman, 1995] David V. Pynadath and Michael P. Wellman. Accounting for context in plan recognition, with application to traffic monitoring. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 472–481, Montreal, Canada, 1995. Morgan Kaufmann.
- [Vilain, 1990] Marc Vilain. Getting serious about parsing plans: a grammatical analysis of plan recognition. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 190–197, 1990.

A General Model for Online Probabilistic Plan Recognition

Hung H. Bui

Department of Computing
Curtin University of Technology
PO Box U1987, Perth, WA 6001, Australia
URL: <http://www.cs.curtin.edu.au/~buihh>
Email: buihh@cs.curtin.edu.au

Abstract

We present a new general framework for online probabilistic plan recognition called the Abstract Hidden Markov Memory Model (AHMEM). The new model is an extension of the existing Abstract Hidden Markov Model to allow the policy to have internal memory which can be updated in a Markov fashion. We show that the AHMEM can represent a richer class of probabilistic plans, and at the same time derive an efficient algorithm for plan recognition in the AHMEM based on the Rao-Blackwellised Particle Filter approximate inference method.

1 Introduction

The ability to perform plan recognition can be very useful in a wide range of applications such as monitoring and surveillance, decision supports, and team work. However the plan recognizing agent's task is usually complicated by the uncertainty in the plan refinement process, in the outcomes of actions, and in the agent's observations of the plan. Dealing with these issues in plan recognition is a challenging task, especially when the recognition has to be done online so that the observer can react to the actor's plan in real-time.

The uncertainty problem has been addressed by the seminal work [Charniak and Goldman, 1993] which phrases the plan recognition problem as the inference problem in a Bayesian network representing the process of executing the actor's plan. More recent work has considered dynamic models for performing plan recognition online [Pynadath and Wellman, 1995; 2000; Goldmann *et al.*, 1999; Huber *et al.*, 1994; Albrecht *et al.*, 1998]. While this offers a coherent way of modelling and dealing with various sources of uncertainty in the plan execution model, the computational complexity and scalability of inference is the main issue, especially for dynamic models.

Inference in dynamic models such as the Dynamic Bayesian Networks (DBN) [Nicholson and Brady, 1994] is more difficult than in a static model. Inference in a static network utilizes the sparse structure of the graphical model to make it tractable. In the dynamic case, the DBN belief state that we need to maintain usually does not preserve the conditional independence properties of the single time-slice

network, making exact inference intractable even when the DBN has a sparse structure. Thus, online plan recognition algorithms based on exact inference will run into problems when the belief state becomes too large, and will be unable to scale up to larger or more detailed plan hierarchies.

In our previous work, we have proposed a framework for online probabilistic plan recognition based on the Abstract Hidden Markov Models (AHMM) [Bui *et al.*, 2002]. The AHMM is a stochastic model for representing the execution of a hierarchy of contingent plans (termed *policies*). Scalability in *policy recognition* in the AHMM is achieved by using an approximate inference scheme known as the Rao-Blackwellised Particle Filter (RBPF) [Doucet *et al.*, 2000]. It has been shown that this algorithm scales well w.r.t. the number of levels in the plan hierarchy.

Despite its computational attractiveness, the current AHMM is limited in its expressiveness, in particular, its inability to represent an uninterrupted sequence of plans and actions. This is due to the fact that each policy in the AHMM is purely reactive on the current state and has no memory. This type of *memoryless* policies cannot represent an uninterrupted sequence of sub-plans since they have no way of remembering the sub-plan in the sequence that is currently being executed. In other words, the decision to choose the next sub-plan can only be dependent on the current state, and not on the sub-plans that have been chosen in the past. Other models for plan recognition such that the Probabilistic State Dependent Grammar (PSDG) [Pynadath and Wellman, 2000; Pynadath, 1999] are more expressive and do not have this limitation. Unfortunately, the existing exact inference method for the PSDG in [Pynadath, 1999] has been found to be flawed and inadequate [Bui, 2002].

The main motivation in this paper is to extend the existing AHMM framework to allow for policies with memories to be considered. We propose an extension of the AHMM called the Abstract Hidden Markov mEmory Model (AHMEM). The expressiveness of the new model encompasses that of the PSDG [Pynadath and Wellman, 2000], thus the new model removes the current restriction of the AHMM. More importantly, we show that the RBPF approximate inference method used for the AHMM can be extended to the more general AHMEM as well, ensuring that the new generalized model remains computationally attractive. To the best of our knowledge, we are the first to provide a scalable inference method

for this general type of hierarchical probabilistic plan hierarchy.

The paper is structured as follows. Section 2 provides a more detailed discussion of the AHMM, PSDG and related models for online probabilistic plan recognition. The AHMEM is introduced in section 3 and the algorithms for plan recognition are presented in section 4. Experimental results with a prototype system are provided in section 5. Finally, we conclude and discuss directions for future work in section 6.

2 Related Models for Online Probabilistic Plan Recognition

In the AHMM [Bui *et al.*, 2000; 2002], an agent's probabilistic plan is modeled by an abstract Markov policy (AMP). An AMP is an extension of a policy in Markov Decision Processes (MDP) defined within a subset of the environment state space so that it can select other more refined AMPs and so on to form a hierarchy of policies. The AMP is thus similar to a contingent plan that prescribes which sub-plan should be invoked at each applicable state of the world. The noisy observation about the environment state can be modelled by making the state "hidden", similar to the hidden state in the Hidden Markov Models [Rabiner, 1989]. The stochastic process resulting from the execution of an AMP is termed the *Abstract Hidden Markov Model*. Intuitively, the AHMM models how an AMP causes the adoption of other policies and actions at different levels of abstraction, which in turn generate a sequence of states and observations. In the plan recognition task, an observer is given an AHMM corresponding to the actor's plan hierarchy, and is asked to infer about the current policy being executed by the actor at all levels of the hierarchy, taking into account the sequence of observations currently available. This problem is termed *policy recognition* [Bui *et al.*, 2002].

Scalability of policy recognition in the AHMM is achieved by using a hybrid inference method, a variant of the Rao-Blackwellised particle filter (RBPF) [Doucet *et al.*, 2000]. When applied to DBN inference, Rao-Blackwellisation [Casella and Robert, 1996] splits the network into two sets of variables: the set of variables that need to be sampled (termed the Rao-Blackwellising (RB) variables), and the set of remaining variables whose belief state conditioned on the RB variables need to be maintained via exact inference (termed the Rao-Blackwellised (RB) belief state). The RBPF thus allows us to combine sampling-based approximate inference with exact inference to achieve efficiency and improve accuracy.

The Probabilistic State-Dependent Grammar (PSDG) [Pynadath, 1999; Pynadath and Wellman, 2000] can be described as the Probabilistic Context Free Grammar (PCFG) [Jelinek *et al.*, 1992], augmented with a state space, and a state transition probability table for each terminal symbol of the PCFG. In addition, the probability of each production rule is made state dependent. As a result, the terminal symbol now acts like primitive actions and the non-terminal symbol chooses its expansion depending on the current state. The AHMM is equivalent to a special class of PSDG where only produc-

tion rules of the form $X \rightarrow YX$ and $X \rightarrow \emptyset$ are allowed. The first rule models the adoption of a lower level policy Y by a higher level policy X , while the second rule models the termination of the policy X . The PSDG model considered by Pynadath and Wellman allows for more general rules of the form $X \rightarrow Y_1 \dots Y_m X$, i.e., the recursion symbol must be located at the end of the expansion. Thus in a PSDG, a policy might be expanded into a sequence of policies at the lower level which will be executed one after another before control is returned to the higher level policy.

Although more expressive than the AHMM, the existing computational method for inference with the PSDG remains inadequate. Pynadath proposed an exact method for updating the belief state of the PSDG in a "compact" closed form. The proposed algorithm seemingly gets around the exponential blow up in the size of the belief state. Unfortunately, the derivation of the algorithm is based on a flawed assumption that the higher levels in the belief state are independent of the lower levels given the current level. For more details about the flaw in the inference algorithm for PSDG, interested readers are referred to [Bui, 2002].

The AHMM, PSDG, and the proposed AHMEM are related to the Hierarchical Abstract Machines (HAM) [Parr, 1998] used in abstract probabilistic planning. In this model, the policy is represented by a stochastic finite automaton, which can call other automata at the lower level. Despite their representational similarity, the computational techniques for AHMEM and related models are intended for plan recognition whereas the HAM model is used for speeding up the process of finding an optimal policy for MDP.

If we ignore the state dependency, the DBN structure of the AHMEM and PSDG is similar to the structure of the Hierarchical Hidden Markov Model (HHMM) [Fine *et al.*, 1998; Murphy and Pashkin, 2001]. However, while the HHMM is a type of Probabilistic Context Free Grammar (PCFG), the AHMEM and PSDG are not due to the state dependency in the model.

3 The Abstract Hidden Markov Memory Models

This section introduces the Abstract Hidden Markov Memory Models (AHMEM), an extension of the AHMM where the policy can have internal memory. Our main aim is to construct a general model for plan recognition whose expressiveness encompasses that of the current AHMM and PSDG models, while retaining the computational attractiveness of the AHMM framework. We first define the AHMEM in subsection 3.1. The DBN structure of the new model is given in subsection 3.2.

3.1 The Model

Consider an MDP-like model with S representing the states of the environment, and A representing the set of primitive actions available to the agent. Each action $a \in A$ is defined by its transition probability from the current state s to the next state s' : $\sigma_a(s, s')$. The set of abstract policies will include every primitive actions. Furthermore, the AHMM [Bui *et al.*,

2000] defines higher level abstract policies on top of a set of policies as follows:

Definition 1 (Abstract Markov Policy (AMP)). Let Π be a set of AMPs, an AMP π^* over Π is defined as a tuple $\langle S_{\pi^*}, D_{\pi^*}, \beta_{\pi^*}, \sigma_{\pi^*} \rangle$ where:

- $S_{\pi^*} \subset \cup_{\pi \in \Pi} S_{\pi}$ is the set of applicable states.
- $D_{\pi^*} \subset \cup_{\pi \in \Pi} D_{\pi}$ is the set of destination states.
- $\beta_{\pi^*} : D_{\pi^*} \rightarrow (0, 1]$ is the set of stopping probabilities such that $\beta_{\pi^*}(d) = 1, \forall d \in D_{\pi^*} \setminus S_{\pi^*}$.
- $\sigma_{\pi^*} : S_{\pi^*} \times \Pi \rightarrow [0, 1]$ is the selection function where $\sigma_{\pi^*}(s, \pi)$ is the probability that π^* selects the policy π at the state s .

An AMP as defined above is purely reactive, in the sense that it selects the next policy at the lower level based only on the current state s . This restricts the set of behaviours that an AMP can represent. For example, it will not be able to represent a plan consisting of a few sub-plans, one followed by another regardless of the state sequence. To represent this kind of plans, the agent needs to have some form of internal memory to remember the current stage of execution. Let M be a set of possible internal memory states. We first extend the definition of our policy to include a memory variable which takes on values in M and is updated after each stage of execution of the policy.¹

Definition 2 (Abstract Markov Policy with Memory (AMPE)). Let Π be a set of AMPEs, an AMPE π^* over Π is defined as a tuple $\langle S_{\pi^*}, D_{\pi^*}, \beta_{\pi^*}, \sigma_{\pi^*}, \iota_{\pi^*}^e, \sigma_{\pi^*}^e \rangle$ where:

- $S_{\pi^*} \subset \cup_{\pi \in \Pi} S_{\pi}$ is the set of applicable states.
- $D_{\pi^*} \subset \cup_{\pi \in \Pi} D_{\pi}$ is the set of destination states.
- $\beta_{\pi^*} : D_{\pi^*} \times M \rightarrow (0, 1]$ is the terminating probability. $\beta_{\pi^*}(d, m)$ is the probability that the policy π^* would stop if the current state is d and the current memory value is m .
- $\sigma_{\pi^*} : S_{\pi^*} \times M \times \Pi \rightarrow [0, 1]$ is the policy selection probability. $\sigma_{\pi^*}(s, m, \pi)$ is the probability that π^* selects the policy π at the state s and memory value m .
- $\iota_{\pi^*}^e : S_{\pi^*} \times M \rightarrow [0, 1]$ is the initial distribution of memory values. $\iota_{\pi^*}^e(s, m)$ is the probability that the initial memory is m if π^* commences at state s .
- $\sigma_{\pi^*}^e : S_{\pi^*} \times M \times M \rightarrow [0, 1]$ is the memory transition probability. $\sigma_{\pi^*}^e(s, m, m')$ is the probability that the next memory value is m' given that the current memory value is m and the current state is s .

Subsequently, we will drop the subscript π^* if there is no confusion about the policy in the context. Note that all the states in $D \setminus S$ are called terminal states and thus $\beta(d, m) = 1, \forall m \in M, d \in D \setminus S$. Also, the policy selection, memory initial and transition probability have to be proper probability

¹One can argue that we can always incorporate the memory variable in the environment state, and hence we gain no extra representational power just by introducing the memory variables. However, incorporating the memory variables in the state variable would blow up the size of the state space and thus defeat our purpose of keeping the model computationally feasible.

distributions, i.e. $\sum_{\pi \in \Pi} \sigma(s, m, \pi) = 1, \sum_{m \in M} \iota^e(s, m) = 1$, and $\sum_{m' \in M} \sigma^e(s, m, m') = 1$.

When an AMPE π^* is executed from a state s , it first initialises its memory value m according to the distribution $\iota^e(s, m)$. Then a policy at the lower level π will be selected according to the distribution $\sigma(s, m, \pi)$. This policy π will be executed until it terminates at some state s' . At the new state s' , the policy π^* itself will terminate with probability $\beta(s', m)$. If it does not terminate, the memory variable will be given a new value m' according to the transition probability $\sigma^e(s', m, m')$. Then a new policy at the lower level π' is selected with probability $\sigma(s', m', \pi')$ and so on.

Using the AMPEs, we can construct a hierarchy of abstract policies in the same way as in the AHMM. We start with a set of primitive actions $\Pi_0 = A$, and for $k = 1, \dots, K$ build a set of new AMPEs Π_k on top of Π_{k-1} . Then, if a top-level policy π^K is executed, it invokes a sequence of level-(K-1) policies, each of which invokes a sequence of level-(K-2) policies and so on. A level-1 policy will invoke a sequence of primitive actions which leads to a sequence of states. We can then introduce the hidden states and model the noisy observation of the state by an observation model $\Pr(o_t | s_t) = \omega(s_t, o_t)$. The dynamic process of executing a top-level AMPE is termed the Abstract Hidden Markov mEmory Model (AHMEM).

Some special cases of the AHMEM are worth mentioning here. First, the AHMM itself is a special AHMEM. The memoryless policies of the AHMM are equivalent to AMPEs where the dependency on the memory variable is ignored (e.g. when M is a singleton set). The class of PSDG considered in [Pynadath, 1999] can also be easily converted to an AHMEM. The terminal symbols in the PSDG are equivalent to the primitive actions. Each non-terminal symbol is equivalent to a memoryless policy. In addition, each sequence $Y = Y_1 Y_2 \dots Y_n$ encountered on the RHS of a production $X \rightarrow Y_1 Y_2 \dots Y_n$ or $X \rightarrow Y_1 Y_2 \dots Y_n X$ is equivalent to a policy whose memory m taking on the values $1, \dots, n$. Y then simply selects the (memoryless) policy Y_i if $m = i$.

Note that in the AHMEM definition, we assume a balanced policy hierarchy for the ease of presentation (all the actions must appear at the same bottom level). However, we can also specify an unbalanced hierarchy by introducing some dummy policies which are equivalent to primitive actions at the higher levels in the hierarchy.

3.2 DBN Representation of the AHMEM

The DBN structure of an AHMEM is very similar to that of an AHMM: at each time slice t , the variable s_t represents the current state, $\pi_t^k, k = 0, \dots, K$ represents the current policy at all levels, e_t^k represents whether π_t^k terminates at the current time, and $l_t = \max\{k | e_t^k = T\}$ denotes the highest level of termination. In addition to these variables, we need to introduce the memory variable m_t^k for $k \geq 1$ to represent the current memory value of the policy π_t^k (the primitive action π_t^0 has no memory). The 2-time-slice DBN is given in Fig. 1. The structure of this network is described below. Note that if x is a variable, we use the notation $x_{i:j}$ to denote the sequence $(x_i, x_{i+1}, \dots, x_j)$, and similarly $x^{i:j}$ to denote $(x^i, x^{i+1}, \dots, x^j)$.

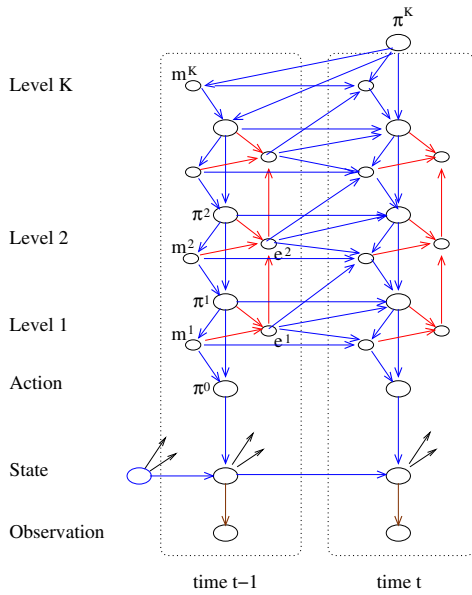


Figure 1: The 2-time-slice DBN for AHMEM

Policy Termination and Selection

The policy termination and selection model for the AHMEM is essentially the same as in the AHMM, except for the dependency on the value of the current memory variable m_t^k . We note that the same context specific independence [Boutilier *et al.*, 1996] properties of the AHMM still hold. For example, if $e_t^{k-1} = F$ then $e_t^k = F$ and e_t^k becomes independent of the remaining parents of this variable. Now consider the variable π_t^k . If $e_{t-1}^k = F$ then $\pi_t^k = \pi_{t-1}^k$ and π_t^k is independent of the remaining parents; otherwise, $\Pr(\pi_t^k | \pi_{t-1}^{k+1}, s_{t-1}, m_{t-1}^{k+1}) = \sigma_{\pi_t^{k+1}}(s_{t-1}, m_{t-1}^{k+1}, \pi_{t-1}^k)$ and π_t^k is independent of π_{t-1}^k .

Memory Update

Consider the variable m_t^k . There are two parents of this node that act like context variables: e_{t-1}^{k-1} and e_{t-1}^k . If $e_{t-1}^{k-1} = F$, the policy at the lower level has not terminated and memory is not updated at this time. Thus, $m_t^k = m_{t-1}^k$, and m_t^k becomes independent of all the remaining parents. If $e_{t-1}^{k-1} = T$ then there are two cases. If $e_{t-1}^k = F$, the policy at level k continues from the previous time, and thus the memory value will be updated according to the memory transition model: $\Pr(m_t^k | \pi_t^k, s_{t-1}, m_{t-1}^k) = \sigma_{\pi_t^k}^e(s_{t-1}, m_{t-1}^k, m_t^k)$. If $e_{t-1}^k = T$, the policy at level k has just started at state s_{t-1} , and thus the memory value will be initialised: $\Pr(m_t^k | \pi_t^k, s_{t-1}) = \sigma_{\pi_t^k}^e(s_{t-1}, m_t^k)$. In this case, m_t^k is independent of the previous memory value m_{t-1}^k .

3.3 Independence Properties in the AHMEM

Even though AMPEs are more expressive than memoryless policies, they remain “autonomous”, in the sense that the higher layers have no influence over the state of an AMPE during its execution. The only way the higher layers can influence the current state of an AMPE is through the conditions

at the start: either through the starting state or the starting time. Thus, the conditional independence theorem for policies in the AHMM still holds in this more general setting. We state the theorem for the AHMEM below. The proof for the AHMM [Bui *et al.*, 2002] can be directly extended to this general case using the context specific independence properties described in the previous subsection.

Theorem 1. Let τ_t^k and b_t^k be two random variables representing the starting time and the starting state, respectively, of the current level- k policy π_t^k : $\tau_t^k = \max\{t' < t | e_{t'}^k = T\}$ and $b_t^k = s_{\tau_t^k}$. Then given the policy π_t^k and its starting state and time, the set of current policies and memories at the higher levels are independent of the set of current policies, memories and state at the lower level:

$$\pi_t^{k+1:K}, m_t^{k+1:K} \perp \pi_t^{0:k-1}, m_t^{1:k}, s_t | \pi_t^k, b_t^k, \tau_t^k \quad (1)$$

Let $r_t = (s_t, l_t)$. Note that knowing $l_{1:t-1}$ is equivalent to knowing precisely when each of the policies starts and ends. Therefore, given $r_{1:t-1}$, the starting time and state of every current policy are known. The following corollary is thus a direct consequence of theorem 1.

Corollary 1. Let $\mathcal{C}_t$ represent the conditional joint distribution $\Pr(\pi_t^{0:K}, m_t^{1:K}, s_t | r_{1:t-1})$. Then $\mathcal{C}_t$ has the following Bayesian network factorization:

$$\mathcal{C}_t = \left(\prod_{k=1}^K \Pr(\pi_t^k, m_t^k | \pi_t^{k-1}, r_{1:t-1}) \right) \Pr(\pi_t^0, s_t | r_{1:t-1})$$

$\mathcal{C}_t$ thus also has the following undirected network representation. We first form the set of cliques: $c_0 = \{\pi_t^0, s_t\}$, $c_k = \{\pi_t^k, m_t^k, \pi_t^{k-1}\}$, $k = 1 \dots, K$. Note that the set of cliques $c_{0:K}$ form a chain of cliques in this order, therefore we term $\mathcal{C}_t$ the *policy-clique chain*. This extends the concept of the *policy chain* in the memoryless case of the AHMM [Bui *et al.*, 2000]. $\mathcal{C}_t$ can be factored into the product of potentials on these cliques: $\mathcal{C}_t \propto \prod_{k=0}^K \psi_k(c_k)$. When $\psi_k = \Pr(\pi_t^k, m_t^k | \pi_t^{k-1}, r_{1:t-1})$ and $\psi_0 = \Pr(\pi_t^0, s_t | r_{1:t-1})$, i.e. when the potential factorization is the same as the directed network factorization, the potentials are said to be in *canonical* form. Any potential representation of the clique chain can be canonicalized by first perform message passing (exact inference) to compute the marginal at each clique. The canonical form can then be computed directly from these marginals. Later on we will use the undirected representation of $\mathcal{C}_t$ for exact inference, and the canonical form (directed representation) of $\mathcal{C}_t$ for obtaining samples from the joint distribution using simple forward sampling.

4 Approximate Inference for AHMEM

In this section, we look at the online inference problem in the AHMEM. Assume that at time t , we have a sequence of observations about the environment state $o_{1:t-1} = (o_1, \dots, o_{t-1})$. We need to compute the belief state of the DBN which is the joint distribution of all the current variables given this observation sequence: $\Pr(\pi_t^{0:K}, m_t^{1:K}, s_t, l_t, o_t | o_{1:t-1})$. From this, we can answer various queries about the current status of the plan execution.

For example, the marginal probability of π_t^k tells us about the current policy the actor is executing at some level k ; the probability $\Pr(m_t^k | \pi_t^k)$ tells us about the current stage of execution of a policy π_t^k ; the probability $\Pr(e_t^k | \pi_t^k)$ tells us if π_t^k will end after the current time, etc.

Since there is no compact closed form representation for the above belief state, exact inference in the structure of the AHMEM is intractable when K is large. However, theorem 1 suggests that we can apply the Rao-Blackwellised Particle Filter (RBPF) to this problem in a similar way as in the AHMM [Bui *et al.*, 2002], i.e. by using r_t as the Rao-Blackwellising (RB) variables. The *Rao-Blackwellised (RB) belief state* is then similar to the original belief state of the AHMEM, except that now $r_{1:t-1}$ are known: $B_t = \Pr(\pi_t^{0:K}, m_t^{1:K}, l_t, s_t, o_t | r_{1:t-1})$. Note that the RB belief state B_t can be obtained directly from the policy-clique chain C_t by adding in the network representing the conditional distribution of o_t and $e_t^{1:K}$, i.e. $B_t = \Pr(e_t^{1:K} | \pi_t^{0:K}, m_t^{1:K}, s_t) \Pr(o_t | s_t) C_t$ (see Fig. 2).

Two main steps in the RBPF procedure are: (1) updating the RB belief state using exact inference, and (2) sampling the RB variable r_t from the current RB belief state.

4.1 Updating the RB Belief State

Fig. 2 shows the modified 2-time-slice DBN when the RB variables are known ($s_t = s, l_t = l$). We note that all the nodes from above level l remain unchanged, while all the links across time slices from level l and below can be removed. This greatly simplifies the network structure, allowing the updating operations to be performed efficiently.

Since the B_t can be obtained directly from C_t , all we have to do is to compute C_{t+1} from C_t . The procedure for updating as usual has two stages: (1) absorbing the new evidence, i.e. from C_t , we need to compute $C_{t+} = \Pr(\pi_t^{0:K}, m_t^{1:K} | r_{1:t})$; and (2) projecting to the new time slice, i.e. from C_{t+} , we need to compute C_{t+1} .

In principle, we apply a simplified version of the junction-tree algorithm [Jensen, 1996] on the undirected network representation of C_t to perform the update step. This is in fact a generalization of the arc-reversal procedure which operates on directed network representation of the policy chain in the AHMM [Bui *et al.*, 2002]. The algorithm for updating the RB belief state is given in Fig. 3. The absorbing step involves simply incorporating the evidence likelihood into the potentials of C_t to obtain the potentials for C_{t+} . The projecting step involves first adding time-slice $t+1$ to C_{t+} (see Fig. 2), then marginalizing the now redundant variables $\pi_t^{0:l}, m_t^{1:l+1}$ in the old time slice. The marginalization is done by performing message passing between the cliques of the 2-time-slice network shown in Fig. 2.

Since the potentials of C_t from level $l+2$ to level K stay unchanged, the complexity of the algorithm is $O(l)$ where l is the highest level of termination. Furthermore, if one of these potentials is in canonical form, it remains in canonical form after the updating procedure.

4.2 RBPF for AHMEM

The full RBPF algorithm for policy recognition in the AHMEM is provided in Fig. 4. The general structure is the

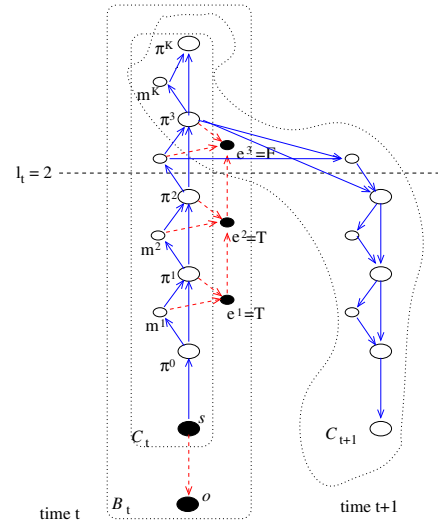


Figure 2: Two time slices of the RB belief state

same as the RBPF procedure for AHMM [Bui *et al.*, 2002]. At each time step t , the algorithm maintains a set of N samples, each consists of a value for s_{t-1} and l_{t-1} , and a parametric representation of C_t . The differences are in the details of how to obtain new samples and update the RB belief state.

To obtain each new sample (s_t, l_t) , we first need to canonicalize the potentials of C_t . However, assuming that C_{t-1} is in canonical form, we only have to canonicalize the potentials of C_t between level 0 and level $l+1$ with complexity $O(l)$. Thus the complexity of the sampling step is $O(Nl)$. Since the complexity of the updating step is also $O(Nl)$, the overall complexity of the algorithm at each time step is $O(Nl)$. Furthermore, the distribution for l usually decays exponentially, thus the average complexity is only $O(N)$.

If at some time t , an estimation e.g. $\Pr(\pi_t^K | o_{1:t-1})$ is required, we need to compute $h = C_t(\pi_t^K)$ for each sample. This involves performing message passing for the entire chain C_t . Thus the complexity for this time step is $O(NK)$. Other types of queries are also possible, as long as the probability required can be computed from the RB belief state. For example, we can ask the question: if the actor is currently executing π_t^k , what is the current stage of execution of this policy? To answer this query, we need to compute the conditional probability $\Pr(m_t^k | \pi_t^k, o_{1:t-1})$. This can easily be achieved by replacing the h function in the algorithm with $h = C_t(m_t^k | \pi_t^k)$.

5 Experimental Results

We have implemented the above algorithm in a surveillance domain to demonstrate its working in a practical application. The environment consists of a spatial area which has two separate rooms and a corridor monitored by a set of 5 cameras (Fig. 5). The monitored area is divided into a grid of cells, and the cell coordinates constitute the overall state space S . The coordinates returned by the cameras are modelled as the noisy observations of the true coordinates of the tracked person, and over time, provide the sequence of observations $o_{1:t}$.

```

Input: potentials for  $\mathcal{C}_t$ , evidence  $s$  and  $l$ 
Output: new potentials for  $\mathcal{C}_{t+1}$ 
Begin
/* absorbing */
 $\psi_1 \leftarrow \psi_1 \times \psi_0(\pi_t^0, s)$ 
 $\psi_{l+1} \leftarrow \psi_{l+1} \times (1 - \beta_{\pi_t^{l+1}}(s, m_t^{l+1}))$ 
For  $k = 1, \dots, l$ 
 $\psi_k \leftarrow \psi_k \times \beta_{\pi_t^k}(s, m_t^k)$ 
/* projecting */
Construct the new cliques:
 $c_{l+1}^e = \{m_t^{l+1}, m_{t+1}^{l+1}, \pi_t^{l+1}\}$ ,  $c_{l+1}^l = \{\pi_t^{l+1}, m_{t+1}^{l+1}, \pi_{t+1}^l\}$ 
 $\psi_{l+1}^e = \sigma_{\pi_t^{l+1}}^e(s, m_t^{l+1}, m_{t+1}^{l+1})$ ,  $\psi_{l+1}^l = \sigma_{\pi_t^{l+1}}^l(s, m_{t+1}^{l+1}, \pi_{t+1}^l)$ 
Perform message passing along the path
 $c_1 \rightarrow c_2 \rightarrow \dots \rightarrow c_{l+1} \rightarrow c_{l+1}^e \rightarrow c_{l+1}^l$ 
 $\psi_{l+1} = \psi_{l+1}^e$ 
 $\psi_0 = \sigma_{\pi_{t+1}^0}(s, s_{t+1})$ 
For  $k = 1 \dots l$ 
 $\psi_k = \sigma_{\pi_{t+1}^k}(s, m_{t+1}^k, \pi_{t+1}^{k-1}) \psi_{\pi_{t+1}^k}^e(s, m_{t+1}^k)$ 
End

```

Figure 3: Updating $\mathcal{C}_t$

At the top-level, each policy models a type of activity performed by the person. For example, *print* is a policy that involves a sequence of going to the computer and the printer, possibly going to the paper store if the printer is out of paper, and exiting. Note that this policy cannot be represented in the AHMM framework. In the AHMEM, this can be represented by a policy whose memory transition model is given in Fig. 6. The policies at the lower level, such as going to a computer, model the person's trajectories toward a set of special landmarks in the environment. These policies are constructed as memoryless policies in the same way as in the AHMM [Bui *et al.*, 2002].

The result of querying the top-level policy for the trajectory in Fig. 5 is given in Fig. 7. The system correctly identifies the most probable policy as *print*. The result of querying the memory variable of this policy is given in Fig. 8. The system correctly identifies the sequence of lower-level policies invoked by the *print* policy. These results are obtained by running the RBPF algorithm with 3000 samples. The average processing time for each observation is approximately 0.9 seconds on a 1.7GHz desktop machine. For a more detailed description of this surveillance system, readers are referred to [Nguyen *et al.*, 2003].

6 Conclusion

In conclusion, we have presented the Abstract Hidden Markov Memory Models (AHMEM), a general framework for representing and recognizing probabilistic plans online. The new framework extends the AHMM by allowing the policies to have internal memories which are updated in a Markov fashion. This allows the AHMEM to represent a richer set of hierarchical probabilistic plans, including those belonging to the class of PSDG previously considered. Furthermore,

```

Begin
For  $t = 1, 2, \dots$ 
/* sampling step */
For each sample  $i = 1, 2, \dots, N$ 
Absorb  $o_t$  into  $\mathcal{C}_t^{(i)}$ .
Canonicalize  $\mathcal{C}_t^{(i)}$ , compute  $w_t = \mathcal{B}_t(o_t)$ .
Sample  $s_t^{(i)}, l_t^{(i)}$  from  $\mathcal{C}_t^{(i)}$  and  $\Pr(e_t^{1:K} | s_t, \pi_t^{0:K}, m_t^{1:K})$ 
Update weight  $w^{(i)} = w_t$ 
/* re-sampling step */
Normalize the weight  $\tilde{w}^{(i)} = \frac{w^{(i)}}{\sum_{i=1}^N w^{(i)}}$ 
Re-sample the sample set according to  $\tilde{w}^{(i)}$ 
/* exact step */
For each sample  $i = 1, 2, \dots, N$ 
Compute  $\mathcal{C}_{t+1}^{(i)}$  from  $\mathcal{C}_t^{(i)}$  and  $s_t^{(i)}, l_t^{(i)}$ 
Compute  $h^{(i)} = \mathcal{C}_{t+1}^{(i)}(\pi_{t+1}^k)$ 
/* Estimation step */
Compute the estimator  $\Pr(\pi_{t+1}^k | o_{1:t}) \approx \hat{f} = \frac{1}{N} \sum_{i=1}^N h^{(i)}$ 
End

```

Figure 4: RBPF for AHMEM

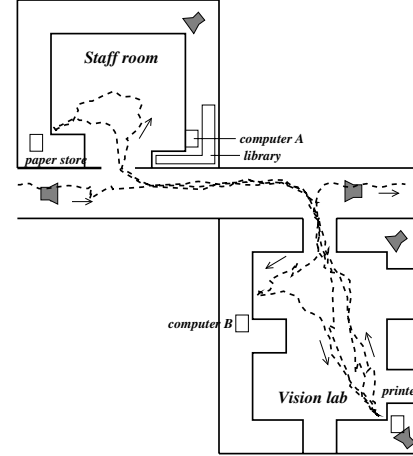


Figure 5: The environment and a person's trajectory

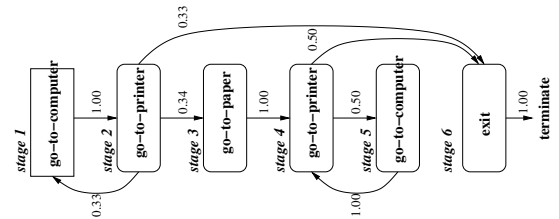


Figure 6: Memory transition of the policy “print”

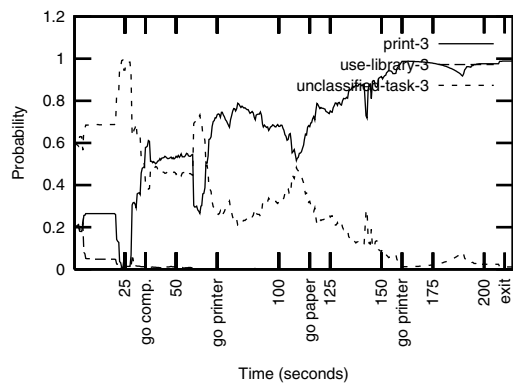


Figure 7: Querying the top level policies

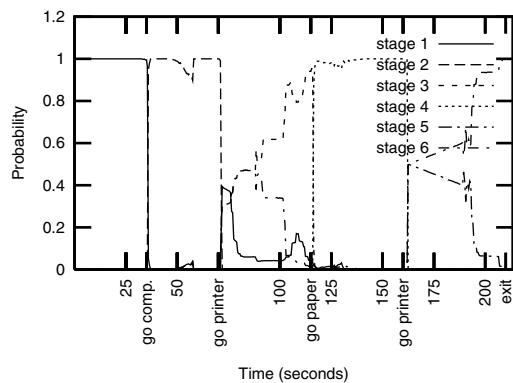


Figure 8: The progress of executing the policy “print”

we have shown that the Rao-Blackwellised Particle Filter (RBPF) approximate inference method used for the AHMM can be extended to the AHMEM, resulting in efficient and scalable procedures for plan recognition in this general setting. Our work demonstrates the advantage of phrasing probabilistic plan recognition as inference in DBN so that suitable approximate methods can be employed to cope with the complexity issue. A similar approach can be applied to more general models to consider more complex plan constructs such as multi-agent plans, interleaving plans etc.

References

- [Albrecht *et al.*, 1998] David W. Albrecht, Ingrid Zukerman, and Ann E. Nicholson. Bayesian models for keyhole plan recognition in an adventure game. *User Modelling and User-adapted Interaction*, 8(1–2):5–47, 1998.
- [Boutilier *et al.*, 1996] Craig Boutilier, Nir Friedman, Moises Goldszmidt, and Daphne Koller. Context-specific independence in Bayesian networks. In *Proceedings of the Twelfth Annual Conference on Uncertainty in Artificial Intelligence*, 1996.
- [Bui *et al.*, 2000] Hung H. Bui, Svetha Venkatesh, and Geoff West. On the recognition of abstract Markov policies. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-2000)*, 2000.
- [Bui *et al.*, 2002] Hung H. Bui, Svetha Venkatesh, and Geoff West. Policy recognition in the Abstract Hidden Markov Models. *Journal of Artificial Intelligence Research*, 17:451–499, 2002.
- [Bui, 2002] Hung H. Bui. Efficient approximate inference for online probabilistic plan recognition. Technical Report 1/2002, School of Computing Science, Curtin University of Technology, Perth, WA, Australia, 2002. Available from <http://www.cs.curtin.edu.au/~buihh/>.
- [Casella and Robert, 1996] George Casella and Christian P. Robert. Rao-Blackwellisation of sampling schemes. *Biometrika*, pages 81–94, 1996.
- [Charniak and Goldman, 1993] E. Charniak and R. P. Goldman. A Bayesian model of plan recognition. *Artificial Intelligence*, 64:53–79, 1993.
- [Doucet *et al.*, 2000] Arnaud Doucet, Nando de Freitas, Kevin Murphy, and Stuart Russell. Rao-Blackwellised particle filtering for dynamic Bayesian networks. In *Proceedings of the Sixteenth Annual Conference on Uncertainty in Artificial Intelligence*, 2000.
- [Fine *et al.*, 1998] Shai Fine, Yoram Singer, and Naftali Tishby. The hierarchical Hidden Markov Model: Analysis and applications. *Machine Learning*, 32, 1998.
- [Goldman *et al.*, 1999] Robert Goldman, Chistopher Geib, and Chirs Miller. A new model of plan recognition. In *Proceedings of the Fifteenth Annual Conference on Uncertainty in Artificial Intelligence*, 1999.
- [Huber *et al.*, 1994] Marcus J. Huber, Edmund H. Durfee, and Michael P. Wellman. The automated mapping of plans for plan recognition. In *Proceedings of the Tenth Annual Conference on Uncertainty in Artificial Intelligence*, 1994.
- [Jelinek *et al.*, 1992] F. Jelinek, J. D. Lafferty, and R. L. Mercer. Basic methods of probabilistic context free grammar. In P. Laface and R. De Mori, editors, *Recent Advances in Speech Recognition and Understanding*, pages 345–360. Springer-Verlag, 1992.
- [Jensen, 1996] F. Jensen. *An Introduction to Bayesian Networks*. Springer, 1996.
- [Murphy and Pashkin, 2001] Kevin Murphy and Mark Pashkin. Linear time inference in hierarchical HMMs. In *NIPS-01*, 2001.
- [Nguyen *et al.*, 2003] Nam T. Nguyen, Hung H. Bui, Svetha Venkatesh, and Geoff West. Recognising and monitoring high-level behaviours in complex spatial environments. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR-03)*, 2003.
- [Nicholson and Brady, 1994] A. E. Nicholson and J. M. Brady. Dynamic belief networks for discrete monitoring. *IEEE Transactions on Systems, Man and Cybernetics*, 24(11):1593–1610, 1994.
- [Parr, 1998] Ronald Parr. *Hierarchical control and learning for Markov Decision Processes*. PhD thesis, University of California, Berkeley, 1998.
- [Pynadath and Wellman, 1995] David V. Pynadath and Michael P. Wellman. Accounting for context in plan recognition, with application to traffic monitoring. In *Proceedings of the Eleventh Annual Conference on Uncertainty in Artificial Intelligence*, 1995.
- [Pynadath and Wellman, 2000] David V. Pynadath and Michael P. Wellman. Probabilistic state-dependent grammars for plan recognition. In *Proceedings of the Sixteenth Annual Conference on Uncertainty in Artificial Intelligence*, 2000.
- [Pynadath, 1999] David V. Pynadath. *Probabilistic grammars for plan recognition*. PhD thesis, Computer Science and Engineering, University of Michigan, 1999.
- [Rabiner, 1989] Lawrence R. Rabiner. A tutorial on Hidden Markov Models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

VISION

Use of Off-line Dynamic Programming for Efficient Image Interpretation

Ramana Isukapalli

101 Crawfords Corner Road
Lucent Technologies
Holmdel, NJ 07733 USA
risukapalli@lucent.com

Russell Greiner

Department of Computing Science
University of Alberta
Edmonton, AB T6G 2E8 Canada
greiner@cs.ualberta.ca

Abstract

An *interpretation system* finds the likely mappings from portions of an image to real-world objects. An *interpretation policy* specifies when to apply which imaging operator, to which portion of the image, during every stage of interpretation. Earlier results compared a number of policies, and demonstrated that policies that select operators which maximize the information gain per cost, worked most effectively. However, those policies are *myopic* — they rank the operators based only on their *immediate* rewards. This can lead to inferior overall results: it may be better to use a relatively expensive operator first, if that operator provides information that will significantly reduce the cost of the subsequent operators.

This suggests using some lookahead process to compute the quality for operators *non-myopically*. Unfortunately, this is prohibitively expensive for most domains, especially for domains that have a large number of complex states. We therefore use ideas from reinforcement learning to compute the *utility* of each operator sequence. In particular, our system first uses dynamic programming, over *abstract* simplifications of interpretation states, to pre-compute the utility of each relevant sequence. It does this off-line, over a training sample of images. At run time, our interpretation system uses these estimates to decide when to use which imaging operator. Our empirical results, in the challenging real-world domain of face recognition, demonstrate that this approach works more effectively than myopic approaches.

1 Introduction

Interpretation is the process of finding the likely mapping from portions of an image to real-world objects. It is the basis for a number of imaging tasks, including recognition (“is objectX in the image?”) and identification (“which object is in the image?”), as well as several forms of tracking (“find all moving objects of typeX in this sequence of images”), etc. [PL95; HR96]. It is important that an interpretation system (*IS*) be *efficient* as well as *accurate*. Any *IS* should have

access to an inventory of “imaging operators” — like edge detectors, region growers, corner locators, etc., each of which, when applied to any portion of the image, returns meaningful tokens (like circles, regions of the same color, etc.). An “interpretation policy” specifies which operator to apply to which portion of the image, during each step of the interpretation process. Such policies must, of course, specify the details: perhaps by specifying exactly which bottom-up operators to use, and over what portion of the image, if and when to switch from bottom-up to top-down, which aspects of the model to seek, etc.

Our earlier work [IG01b] considered various types of policies, towards demonstrating that an “information theoretic policy”, which selects operators that maximize the information gain per unit cost of the imaging operator, work more effectively than others. However, the [IG01b] policies evaluate each operator $o(\cdot)$ *myopically* — i.e., independent of the cost and effectiveness of subsequent operators that would be applied, after performing this $o(\cdot)$. To see why this is problematic, assume the task is to determine whether there is an airplane in an image, by seeking the various parts of an airplane — e.g., the fuselage, wings, engine pods, tailpiece, etc. Now consider the o_{ep} operator that detects and locates the engine pods. As the engine pods are small and often partially-occluded, o_{ep} is probably expensive. However, once these parts have been located, we expect to find the associated wings very easily, and then the remaining parts required to identify the entire airplane. A myopic policy, which evaluates an operator based only on its immediate cost, would miss this connection, and so would probably prefer a cheaper operator over the expensive o_{ep} . A better policy would consider “operator interactions” (here, relating o_{ep} to (say) the “wing finder operator”) when deciding which operator to apply. The data in Section 5 (related to the complex task of face recognition) shows that such non-myopic policies can be both more accurate, and more efficient.

Of course, non-myopic policies must use some type of lookahead to evaluate the quality of its operators; this can be combinatorially expensive to compute. We address this concern by (1) dealing with an abstract version of the “interpretation state space”, and then by (2) using dynamic programming techniques over this abstracted space, pre-computing many relevant “utility” [SB98] values *off-line* [BDB00]. In particular, our system computes the utility of imaging operator se-

quences in each state s encountered, based on data from a set of training examples. It produces a *policy*, which maps each state s to the operator o_s^* that appears to be the most promising, incorporating this lookahead. At run-time, our system finds the best matching state and applies the operator associated with that situation. Our empirical results, in the domain of face recognition, show that such policies work effectively — better than the best results obtained by any of the earlier myopic systems.

Section 2 presents relevant related work, to help frame our contributions. Section 3 discusses the salient features of interpretation strategies. Section 4 then presents our approach, of using reinforcement learning in image interpretation, within the domain of face recognition, using operators that each correspond to types of eigenfeatures. Section 5 provides empirical results that support our claims.

2 Related work

We produce an interpretation of an image by applying a sequence of operators, where each operator maps the current state (typically a partial interpretation) to a new state. As the effects of an operator may depend on information not explicitly contained in the state description, this mapping is stochastic. Moreover, each operator has a cost, and the state associated with the final interpretation has a “quality” (i.e., its accuracy as an interpretation). As such, we view this image-interpretation task as a “Markov Decision Problem” (MDP). This means we can apply the host of reinforcement learning techniques [SB98] to this problem. This specific application area, of image interpretation, follows the pioneering work of Draper [BDB00], which shows that dynamic control policies can outperform hand-coded policies. We extend their work by addressing and exploiting the issue of operator interactions and by doing a systematic analysis of the cost and accuracy tradeoffs in face recognition.

While we could encode each state as the entire image, this would be too unwieldy. Boutilier et al. [BDH99] survey several types of representations in planning problems and discuss ways to exploit them to ease the computational cost of policies or plans. Their work focuses on abstraction, aggregation and decomposition techniques. We use abstractions to reduce the size of each state, and hence of the state space we are exploring; moreover, we are not concerned with the general planning problem. Our system is similar to the “forest inventory management system” [BDL02]. In their system, the output of some operators provide the input to some other operator, while in our system the result of one operator is used to narrow the search for some other operator. Moreover, our system can exploit the structure specific to our task to bound the lookahead depth.

3 Framework

As suggested above, our overall objective is to produce an *effective* interpretation policy — e.g., one that *efficiently* returns a *sufficiently accurate* interpretation, where accuracy and efficiency are each measured with respect to the underlying task and the distribution of images that will be encountered. This section makes this framework more precise. Subsection 3.1

specifies our particular task; Subsection 3.2 lists the strategies we will evaluate; Subsection 3.3 outlines our performance domain, face recognition; and Subsection 3.4 describes the specific operators we will use.

3.1 Input to the Interpretation System

We assume that our interpretation system “ IS ” is given the following information:

★ The **distribution** D of images that the IS will encounter, encoded in terms of the distribution of objects and views that will be seen, etc. For our face recognition task, this corresponds to the distribution of all people the system will see, which varies over race, gender and age, as well as poses and sizes. We approximate this using the images given in the training set. See Figure 1.

★ The **task** $T = \langle \mathcal{V}; C_{max}, P_{min} \rangle$ includes two parts: First, $\mathcal{V}$ specifies the objects that the IS should seek, and what IS should return. Second, the task specification also provides the “evaluation criteria” for any policy, which is based on both the expected “accuracy” (P_{min}) and the maximum interpretation “cost” (C_{max}), which the IS should not exceed.

Here, $\mathcal{V}$ specifies our task is to identify the person from his/her given test image (wrt the people included in the training set), subject to the accuracy P_{min} and cost requirements C_{max} .

★ The **set of possible “operators”** $O = \{o_i\}$ includes (say) various edge detectors, region growers, graph matchers, etc. For each operator o_i , we must specify

- its input and output,
- its “effectiveness”, which specifies the accuracy of the output, as a function of the input.
- its “cost”, as a function of (the size of) its input and parameter setting.

We will use a specific set of operators in our face recognition task; we describe these in detail in Section 3.4.

Borrowing from the MDP literature, we view the main input to each operator as its “state”. As we are dealing with scenes, we could use the entire pixel image as (part of) the state. However, for reasons of efficiency, we will often use an *abstracted* view $\mathcal{F}(s)$ of (our interpretation of the) the scene s . Section 3.4 presents the specific abstraction we are using.

3.2 Strategies

Strategy INFOGAIN: selects the operator that provides the largest information gain (per unit cost) at each step. This myopic strategy first computes the expected information gain $EIG(o)$ for each possible operator and argument combination o , as well as the cost $c(o)$. It then executes the operator that maximizes their ratio, $o^* = \operatorname{argmax}_o \{EIG(o)/c(o)\}$.

We focus on this strategy as a number of earlier empirical studies have demonstrated that it was the best of all of the (myopic) strategies considered, across a number of task-contexts and domains, including simple blocks world [IG01a], car recognition (identifying the make and model of a car based on its tail light assembly [IG01a]), as well as the face recognition system considered here [IG01b].

Strategy BESTSEQ: selects an operator o_{BS}^* that appears to be the most promising in the current abstract state $\mathcal{F}(s)$, as

given by the utility $o_{BS}^* = \pi^*(\mathcal{F}(s))$, where π^* is the (optimal) mapping from state to actions. See Section 4 for details.

3.3 Face recognition task

We investigate the efficiency and accuracy of the strategies listed above in the domain of face recognition [TP91; PMS94; PWHR98; EC97]. This section briefly discusses the prominent “eigenface” technique of face recognition that forms the basis of our approach; then presents our framework, describing the representation and the operators we use to identify faces; and finally presents our face interpretation algorithm, for identifying a person from a given image of his/her face.

Eigenface and EigenFeature Method: Many of today’s face recognition systems use *Principal Component Analysis* (PCA) [TP91]: Given a set of training images of faces, the system first forms the covariance matrix C of the images, then computes the k main eigenvectors of C (“eigenfaces”). Every training face h_i is then projected into this coordinate space (“facespace”), producing a vector $\Omega_i = [\omega_1^{(i)}, \omega_2^{(i)}, \dots, \omega_k^{(i)}]$.

During recognition, the test face h_{test} is similarly projected into the facespace, producing the vector Ω_{test} , which is then compared with each of the training faces. The best matching training face is taken to be the interpretation [TP91].

Following [PMS94], we extend this method to recognize facial features — eyes, nose, mouth, etc. — which we then use to help identify the individual in a given test image. We partition the training data into two sets, $T = \{h_{1,i}\}$ for constructing the eigenfeatures and $S = \{h_{2,i}\}$ for collecting statistics, where each set contains at least one face of each of the n people. (Feature regions for the training data T were extracted using the operators explained in 3.4 and verified manually for correctness). Letting $\text{id}(h)$ denote the person whose face is given by h , we have $\text{id}(h_{1,i}) = i = \text{id}(h_{2,i})$ for $i = 1..n$; each remaining $h_{1,j}$ and $h_{2,j}$ ($j > n$) also maps to $1..n$; i.e., $\text{id}(h_{1,j}), \text{id}(h_{2,k}) \in \{1, \dots, n\}$.

We use PCA on the mouth regions of each T image, to produce a set of eigenvectors; here *eigen-mouths*. For each face image h_i , let $\Omega_i^{(m)}$ be the “feature space” encoding of h_i ’s mouth-region. We will later compare the feature space encoding $\Omega_{test}^{(m)}$ of a new image h_{test} against these $\{\Omega_i^{(m)}\}$ vectors, with the assumption that $\Omega_{test}^{(m)} \approx \Omega_i^{(m)}$ suggests that h_{test} is really person i — i.e., finding that $\|\Omega_{test}^{(m)} - \Omega_i^{(m)}\|$ is small should suggest that $\text{id}(h_{test}) = i$. (Note $\|\cdot\|$ refers to the L_2 norm, aka Euclidean distance.) To quantify how strong this belief should be, we compute $M = |T| \times |S|$ values $\{d_{i,j}\}$ where each $d_{i,j} = \|\Omega_{1,i}^{(m)} - \Omega_{2,j}^{(m)}\|$ is the Euclidean distance between the “eigen-mouth encodings” of T ’s $h_{1,i}$ and S ’s $h_{2,j}$. Using the T training data, we can learn a mapping from these values to probabilities $P(\|\Omega_{test}^{(m)} - \Omega_i^{(m)}\| | \text{id}(h_{test}) = i)$, which we can use to estimate $P(\Omega_{test}^{(m)} | \text{id}(h_{test}) = i)$. (See [IG01b] for details.)

We compute similar estimates for the other facial features, such as nose (n), left eye (le) and right eye (re). We then use the Naïve-Bayes assumption [DH73] (that features are independent, given a specific person)¹ to compute a cumulative

“face probability” from these “feature probabilities”: the values $\|\Omega_{test}^{(f)} - \Omega_i^{(f)}\|$ for each feature f , corresponding to a set of $P(\Omega_{test}^{(m)} | \text{id}(h_{test}) = i)$ values (one for each individual i). We then compute

$$\begin{aligned} P(\text{id}(h_{test}) = i | \Omega_{test}^{(m)}, \Omega_{test}^{(le)}, \Omega_{test}^{(n)}) \\ = \alpha \cdot P(\text{id}(h_{test}) = i) \cdot \\ P(\Omega_{test}^{(m)}, \Omega_{test}^{(le)}, \Omega_{test}^{(n)} | \text{id}(h_{test}) = i) \\ = \alpha \cdot P(\text{id}(h_{test}) = i) \cdot P(\Omega_{test}^{(m)} | \text{id}(h_{test}) = i) \cdot \\ P(\Omega_{test}^{(le)} | \text{id}(h_{test}) = i) \cdot P(\Omega_{test}^{(n)} | \text{id}(h_{test}) = i) \end{aligned} \quad (1)$$

where α is a scaling constant.

3.4 Operators

We use four classes of **operators**, $O = \{o_{le}(k, \ell, r), o_{re}(k, \ell, r), o_n(k, \ell, r), o_m(k, \ell, r)\}$ to detect respectively “left eye”, “right eye”, “nose” and “mouth”. Each specific operator also takes several parameters: $k \in \{25, 30, 35, 40, 45\}$ specifies the number of eigen-vectors being considered; the other parameters $\ell = \langle \ell_x, \ell_y \rangle$ and $r = \langle r_x, r_y \rangle$ specify the space this operator will sweep, looking for this feature: It will look in the rectangle $\langle \ell_x \pm r_x, \ell_y \pm r_y \rangle$; i.e., sweep $r_x \times r_y$ pixels, centered at $\langle \ell_x, \ell_y \rangle$.

Each instantiated operator $o(\cdot) \in O$ takes as input the image of a test face h_{test} , and returns a probabilistic distribution over the individuals. It has three subtasks: SubTask#1 locates the feature $h_{test}^{(f)}$ from within the entire face h_{test} . Here we use a simple template matching technique in which we search in the fixed region r of size $r_x \times r_y$ pixels, centered at pixel ℓ . SubTask#2 then projects the relevant region of the test image $h_{test}^{(f)}$ into the feature space — computing $\Omega_{test}^{(f)}$ of dimension k . SubTask#3 uses this $\Omega_{test}^{(f)}$ to compute first the values $d_{i,test} = \|\Omega_{test}^{(f)} - \Omega_i^{(f)}\|$ for each person i and then to compute the probability $P(\text{id}(h_{test}) = i | \Omega_{test}^{(f)})$ for each person i . We use Equation 1 to update the distribution when considering the 2nd and subsequent features; see [IG01b]. For each eigenspace dimension k , we empirically determined the cost (in milliseconds) of the four operators —

$$\begin{aligned} C(o_{le}(k, r, \ell)) &= 9.25k + 0.25|r| - 446.00 \\ C(o_{re}(k, r, \ell)) &= 10.35k + 0.27|r| - 454.27 \\ C(o_n(k, r, \ell)) &= 9.95k + 0.23|r| - 486.51 \\ C(o_m(k, r, \ell)) &= 14.4k + 0.26|r| - 308.44 \end{aligned}$$

where $|r| = r_x \times r_y$ is the size of the range. While increasing the dimensionality k of the feature space should improve the accuracy of the result, here we see explicitly how this will increase the cost.

4 Use of Dynamic Programming

This section briefly overviews MDPs, presents “state abstraction” in face recognition and shows how dynamic programming can be used to compute the utility of operators in abstracted states. We also discuss operator interaction and describe how it can be exploited in face recognition.

uation. However, this classification has been found to work well in practice [Mit97].

¹Of course, this assumption is almost assuredly false in our sit-

4.1 Markov Decision Problem

A *Markov Decision Problem* can be described as a 4-tuple $\langle \mathcal{S}, A, M, R \rangle$ where $\mathcal{S} = \{s_1, s_2, \dots, s_n\}$ is a finite set of states, $A = \{a_1, a_2, \dots, a_m\}$ is a finite set of actions, $M: \mathcal{S} \times A \times \mathcal{S} \rightarrow [0, 1]$ is the state transition probability function ($M_{s,s'}^a = P(s' | s, a)$ is the probability that taking action a in situation s leads to being in state s') and $R: \mathcal{S} \times A \rightarrow \mathbb{R}$ is the reward an agent gets for taking an action $a \in A$ in state $s \in \mathcal{S}$. The *Markov property* holds if the transition from state s_i to s_j using action a depends only on s_i and not the previous history. A *policy* $\pi: \mathcal{S} \rightarrow A$ is a mapping from states to actions. For any policy π , we can define a utility function U_π such that

$$U_\pi(s) = \max_a \left\{ R(s, a) + \sum_{s'} M_{s,s'}^a \times U_\pi(s') \right\}$$

which corresponds to the expected cumulative rewards of executing the apparently-optimal action a in state s , then following policy π after that.

Given an MDP, we naturally seek an optimal policy π^* — i.e., a policy that produces the *optimal* cumulative reward for each state, $U^*(s)$. Dynamic programming provides a way to compute this optimal policy, by computing the utilities of the best actions. Of course, given the values of $U^*(\cdot)$, the optimal action at each state s is simply $\pi^*(s) = \operatorname{argmax}_a \{R(s, a) + \sum_{s'} M_{s,s'}^a \times U^*(s')\}$.

This can be challenging in the general setting, where sequences of actions can map one state to itself; much of the work in Reinforcement Learning [SB98] is designed to address these issues. In our current case, however, we will see there is a partial order on the states, meaning no sequence of actions can map a state to itself. Here, we can use dynamic programming to compute the optimal utility for each “final step”, then use these values to compute the optimal action (and utility) for each penultimate state, and so forth.

4.2 State abstraction

An MDP involves “states”. In our face recognition task, any attempt to define states in terms of the pixel values of an image would be problematic, as there will be far too many states to enumerate. Following [BDB00; BDL02], we use the notion of “abstract states”, which basically redefines an original state in a much more compact form, using only the certain aspects of the state. Since we recognize a person using the features, we define an abstraction function

$$\mathcal{F}(s) = \langle C, \vec{P}, x_{le}, y_{le}, x_{re}, y_{re}, x_n, y_n, x_m, y_m \rangle \quad (2)$$

where s is the actual complex state as present in the image and $\langle x_i, y_i \rangle$ denotes the location (center) of feature f_i (left eye, right eye, nose or mouth) in the image. (Here C is the total cost we have spent so far, and $\vec{P} = \langle P(\text{id}() = x | \vec{E}) \rangle_x$ is the current posterior distribution over the possible faces, based on the current evidence $\vec{E}$.) The location of some feature f_i may not yet be known in an image; here, we use the value of $\perp$ for both x_i and y_i . Further, we say two abstractions are “ δ -equivalent”, written $\mathcal{F}(s_1) \approx_\delta \mathcal{F}(s_2)$, iff

- s_1 and s_2 have located the same set of features — i.e., $x_i(s_1) = \perp$ iff $x_i(s_2) = \perp$ and

- $\sum_{i: x_i(s_1) \neq \perp} \|x_i(s_1) - x_i(s_2)\| + \|y_i(s_1) - y_i(s_2)\| \leq \delta$ — i.e., when the distance (in pixels) between the centers of feature f_i in s_1 and s_2 is small ($\leq \delta$).

As we are using normalized images that contain only faces, denoting the distance in absolute pixel values is not an issue. In general, δ is a small, predefined constant; we used 10.²

The result of abstraction is that a large number of complex states can be described by a small number of compact state descriptions. Of course, the same abstract state can represent multiple states — $\mathcal{F}(s_1) = \mathcal{F}(s_2)$ when $s_1 \neq s_2$.

We use a lookahead algorithm to compute the U -values. Since we consider only four features, we need a lookahead of only depth four. Moreover, we can do this during the training phase. We will use the reward function

$$R(s, a) = \begin{cases} \vec{P}(\text{id}() = x^* | \vec{E}) & \text{if } a = \text{Stop} \\ -\alpha \cdot t(a) & \text{otherwise} \end{cases} \quad (3)$$

that penalizes each operator a by α times the time $t(a)$ it required (in seconds) as well as a positive score for obtaining the correct interpretation (here x^* represents that correct identity of the person in the image.) We used $\alpha = 0.2$. During the training phase, our system finds the optimal operator sequence — i.e., the one that has the maximum $U'(s) = U(\mathcal{F}(s))$ value, based on the abstracted state $\mathcal{F}(s)$.

4.3 Dynamic Programming

This section shows how to compute $U(s)$ utility values.

Tree expansion: We use four classes of operators $O = \{o_{le}(k, \ell, r), o_{re}(k, \ell, r), o_n(k, \ell, r), o_m(k, \ell, r)\}$, each with 5 different values of $k \in \{25, 30, 35, 40, 45\}$, for a total of 20 operators (see Section 3.4). The parameters $\ell = \langle \ell_x, \ell_y \rangle$ and $r = \langle r_x, r_y \rangle$ specify the rectangular area where the operator will search. Our system does not have to search over their values; instead, it directly computes their values from the locations of the features $\langle f_1, f_2, \dots, f_i \rangle$ ($i < 4$) we have already detected; see Section 4.4.

Our system expands the operator tree exhaustively using a depth-first search. Of course, the depth of the tree is at most 4 (not 20), as we only consider at most one instance of each operator along any path.

Computing $U(s)$: We compute the various utility values using a dynamic programming approach:

(i) We first apply every sequence of 4 different operators to the original image, to produce the set of all possible $s^{(4)}$ leaf states. Then set $U(s^{(4)}) = P(\text{id}() = x^* | \vec{E})$, where x^* is the correct interpretation, and $\vec{E}$ is the observed locations of the various features.

(ii) Now consider each state $s^{(3)}$ that involves some set of 3 operators. For each, we can consider all 5 + 1 possible actions: either **Stop**, or apply the remaining operator (with one of the 5 possible k values). We can trivially compute the utility of each option: as $P(x^* | \vec{E}')$ for the **Stop** action, and $-\alpha t(a) + U(s^{(4)})$ if the action a takes time $t(a)$ and produces

²For this value of δ , we found that there were about 1600 entries of $\mathcal{F}(s)$ totally.



Figure 1: Training images (top); test images (bottom)

the state $s^{(4)}$. We set $U(s^{(3)})$ to be the largest of these 6 values, and $\pi_{BS}(s^{(3)})$ to the action which produced this largest values.

After computing $U(s^{(3)})$ for all “depth-3” states, we then recur, to deal with depth-2 states, and so forth.

Of course, these states here actually refer only to the abstracted state $\mathcal{F}(s)$. Moreover, we “bin” these values: Suppose we have encountered some abstracted state $\mathcal{F}(s_1)$. If we later find a state s_2 , where $\mathcal{F}(s_1) \approx_\delta \mathcal{F}(s_2)$, and determine $U(\mathcal{F}(s_1)) < U(\mathcal{F}(s_2))$, we would then reset the value of $U(\mathcal{F}(s_1))$ and $\pi_{BS}(\mathcal{F}(s_1))$ to be the values found for $\mathcal{F}(s_2)$.

Retrieving the most promising operator during interpretation: This entire procedure of computing the optimal policy π_{BS} is done off-line, during the training phase. During interpretation (performance phase), in any state s_i we (i) find the “nearest neighbour” $\mathcal{F}(s_j)$, such that the sum of the distances between the corresponding features in $\mathcal{F}(s_i)$ and $\mathcal{F}(s_j)$ is the minimum over all possible entries; and (ii) return the operator $\pi_{BS}(\mathcal{F}(s_j))$, which is either “Stop”, or an instantiated operator — say $o_n(25, \langle 50, 66 \rangle, \langle 60, 52 \rangle)$. If not-Stop, the run-time system then executes this operator to locate another feature, which produces a new state (updating the total cost and posterior distribution as well). It then determines the next action to perform, and so forth.

There is one place where our system might perform differently from what the policy dictates: As we are maintaining the *actual* cost so far, and the *actual* posterior distribution over interpretations, our run-time system will actually terminate if either the actual cost has exceeded our specs, or if the highest probability is above the minimal acceptable value.

Notice, given our specific set-up — e.g., only 4 types of operators that can only be executed once, and which always succeed, etc. — the policy obtained can be viewed as a “straight-line” policy: seek one specific feature, then another, until achieving some termination condition; see Figure 2 for an example. In general, this basic dynamic programming approach could produce more complex policies.

4.4 Operator dependencies

Our face recognition system will exploit a certain type of operator interactions, *viz.*, using the result of one operator to simplify a subsequent one. That is, after detecting some feature (say the left eye), we expect to have some idea where to find

```

 $\pi_{BS}(\langle C, \vec{P}, \langle x_{le}, y_{le}, x_{re}, y_{re}, x_n, y_n, x_m, y_m \rangle \rangle)$ 
If  $C \geq C_{max} \vee P_{min} \leq \max_x P(\text{id}() = x | \vec{E})$ 
Then Return( Stop )
Else Case  $\langle x_{le}, y_{le}, x_{re}, y_{re}, x_n, y_n, x_m, y_m \rangle$  of
  *  $\langle \perp, \perp, \perp, \perp, \perp, \perp, \perp, \perp \rangle$ :
    Return(  $o_{le}(25; \langle 31, 52 \rangle \pm \langle 21, 22 \rangle)$  )
  *  $\langle x_{le}, y_{le}, \perp, \perp, \perp, \perp, \perp, \perp \rangle$ :
    Return(  $o_{re}(25; \langle x_{le} + 46, y_{le} + 8 \rangle \pm \langle 25, 17 \rangle)$  )
  *  $\langle x_{le}, y_{le}, x_{re}, y_{re}, \perp, \perp, \perp, \perp \rangle$ :
    if  $\langle x_{le}, y_{le}, x_{re}, y_{re} \rangle \approx_\delta \langle 16, 43, 50, 38 \rangle$ 
    then Return(  $o_n(30; \langle x_{re} - 6, y_{re} + 22 \rangle \pm \langle 24, 22 \rangle)$  )
    else Return(  $o_n(25; \langle x_{re} - 6, y_{re} + 22 \rangle \pm \langle 24, 22 \rangle)$  )
  *  $\langle x_{le}, y_{le}, x_{re}, y_{re}, x_n, y_n, \perp, \perp \rangle$ :
    if  $\langle x_{le}, y_{le}, x_{re}, y_{re}, x_n, y_n \rangle \approx_\delta \langle 19, 41, 52, 41, 33, 51 \rangle$ 
    then Return(  $o_m(30; \langle x_n + 14, y_n + 30 \rangle \pm \langle 31, 19 \rangle)$  )
    else Return(  $o_m(25; \langle x_n + 14, y_n + 30 \rangle \pm \langle 31, 18 \rangle)$  )
  * Return( Stop )

```

Figure 2: Policy π_{BS} learned by BESTSEQ, for $\langle C_{max} = \infty, P_{min} = 0.9 \rangle$

the other features (e.g., the right eye). We model this as a linear function, mapping from the location of the left eye to the expected location of the right eye. If the left eye was detected at $\langle x_{le}, y_{le} \rangle$, then the expected location of the right eye would be the window $\langle x_{le} + 46, y_{le} + 8 \rangle \pm \langle 25, 17 \rangle$. We also model the “variance” — *i.e.*, the size of the search window around this expected position. To be more precise, let us assume that the search window for some feature f_1 is of size $r_{x1} \times r_{y1}$ pixels. After detecting feature f_2 , we instead search for f_1 in a smaller region $r'_{x1} \times r'_{y1}$, centered at the location that we computed from f_2 's location, using that linear function.

After observing several features (say nose and left eye), we will have several estimates for the location of the current feature (here right eye). Here we take the smallest bounding box and use that as the search area for locating the current feature. This “factoring” means we need only consider a set of 4×3 transformations (of each of the four feature versus the other three features), rather than deal with all possible subsets.

To make these ideas more concrete: Initially, given no other information, we would look for the right eye in the region $\langle 67, 52 \rangle \pm \langle 25, 22 \rangle$. However, after finding the left eye at location $\ell_{le} = \langle 25, 50 \rangle$ our system knows that, if asked to look for the right eye, it should search in the region $\langle 71, 58 \rangle \pm \langle 25, 17 \rangle$. (We are not committing to looking for the right eye at this time; just indicating where to search, if requested.) Notice this region is different from the one we would consider if we had not located the left eye; moreover, if we also knew that the nose was at location $\ell_n = \langle 30, 50 \rangle$ (as well as left eye), we would use a yet more refined region — here $\langle 61, 55 \rangle \pm \langle 15, 12 \rangle$.

4.5 Image interpretation policies

During interpretation, INFOGAIN and (the policy produced by) BESTSEQ each iteratively select an operator $o(k) \in O$. (Recall that the values of r and ℓ are determined from the context (of other detected features); we therefore do not need to specify them here.) INFOGAIN chooses an instantiated operator $o = o_{IG}(k_{IG})$ that has the maximum $EIG(o)/c(o)$

value. BESTSEQ selects an operator $o_{BS}^* = \pi_{BS}(\mathcal{F}(s))$ where s is the current state and π_{BS} is the (optimal) mapping from state to actions. In each case, the operator is applied to the appropriate region in the given test face image and the distribution is updated...until one face is identified with sufficiently high probability ($> P_{min}$) or the system fails (by exhausting all the possible operators, or cost $> C_{max}$).

5 Experiments

We used face images of 102 different people, each 92×112 pixels. We assigned 187 images to the set T , another 187 images to S and used another 333 as test images.³ As shown in Figure 1, the faces are more or less in the same pose (facing front), with some small variation in size and orientation.⁴ We considered all 20 operators based on the four features listed above and $k \in \{25, 30, 35, 40, 45\}$ for each feature. The BESTSEQ interpretation policy decides precisely what operator $o_i(k, \ell, r)$ to apply on which portion of the image, given each state (which here corresponds to a specific set of information gathered); see Figure 2.

Basic Experiment: We set $P_{min} = 0.9$ and $C_{max} = \infty$ (i.e., no upper limit on identification cost).⁵ In each “set-up”, we assigned a random probability to each person (corresponding to drawing a sample with replacement). On each run, we picked one face randomly from the test set as the target, and identified it using each of the policies. We repeated this for a total of 30 runs per set-up, then changed the probability distribution and repeated the entire process again, for a total of 15 set-ups. The accuracy of identification for INFOGAIN and BESTSEQ are 89.78% and 90.44%, respectively. (Recall both were required to obtain at least $P_{min} = 90\%$.) The average cost of identification was 1.045 ± 0.038 and 0.717 ± 0.030 seconds⁶ for INFOGAIN and BESTSEQ respectively. Hence, we see BESTSEQ took much less time than INFOGAIN to obtain this level of accuracy.

The first two rows of Table 1 shows how often INFOGAIN (resp., BESTSEQ) used one (resp., two, three, four) operators before terminating, over these 450 runs. We observe the INFOGAIN applied slightly more operators than BESTSEQ on the average. More importantly, as BESTSEQ can exploit simple operator interactions, the search area it uses for the subsequent operators can be narrower than INFOGAIN’s. These two factors result in a much lower cost for BESTSEQ.

Bounding the Cost: In many situations we need to impose a hard restriction on the total cost; we therefore considered var-

³We assume that any test face-image belongs to one of the people in the training set, but probably with a different facial expression or in a slightly different view, and perhaps with some external features not in the training image (like glasses, hat, etc.), or vice versa.

⁴(1) These faces were downloaded from the web sites whitechapel.media.mit.edu and www.cam-orl.co.uk. (2) This work assumes the head has already been located and normalized; if not, we can use standard techniques [TP91] first.

⁵The other part of the task, $\mathcal{V}$, is “identifying people”; this is true for all of the experiments discussed here.

⁶All the experiments reported in this paper were run on a Pentium 866 MHz. PC with 128 MB RAM running Linux OS 2.2.19-12.

Table 1: How often BESTSEQ (INFOGAIN) used k Operators

	C_{max}	1	2	3	4	ave
IG	∞	7.11	57.56	10.00	25.33	2.54
BS	∞ ($P_{min} = 0.9$)	8.89	56.00	15.33	19.79	2.46
BS	2.00	11.33	64.89	3.78	20.00	2.32
BS	1.50	11.33	64.89	3.78	20.00	2.32
BS	1.00	11.33	64.89	5.33	19.11	2.34
BS	0.50	65.33	34.67	0.00	0.00	1.35
BS	0.25	100.0	0.00	0.00	0.00	1.00

BS = BESTSEQ; IG = INFOGAIN

ious values of C_{max} in the range [0.25–2.0] seconds. We then picked one face randomly from the test set, and identified the test image for each of these maximal costs, using INFOGAIN and BESTSEQ.

As always, we terminate whenever the probability of any person exceeds P_{min} or if the cost exceeds C_{max} , returning the most likely interpretation.

We repeated this experiment for a total of 15 set-ups (each with a different distribution over the people) and with 30 random runs (target face images) per set-up. We found that 74.29% of the policies produced matched the policy shown in Figure 2.

Figure 3(a) plots the accuracy (the percentage of correct identifications) for the policies found for various values of C_{max} . In general, the policies produced by BESTSEQ had better accuracy than the ones produced by INFOGAIN. BESTSEQ found different policies for different costs. The bottom 5 rows of Table 1 show how often BESTSEQ used one (resp., two, three, four) operators for different values of C_{max} before terminating.

For low cost values (under 0.5 seconds), BESTSEQ performs much better than INFOGAIN (accuracy of $\approx 65.33\%$, versus $\approx 1.78\%$). As BESTSEQ was able to exploit operator interactions, it was able to apply a second (and probably a third) operator by using a smaller sweep area (ℓ, r) within the allotted time. Since INFOGAIN does not exploit this, it needed significantly more time.

Varying the Minimum Accuracy: In this experiment, we varied P_{min} from 0.1 to 0.9. For each of these values, we chose a face randomly from the test set as the target and identified it using each of the two policies. During the process, the first person i in the training set for which $P(\text{id}(h) = i | \cdot) > P_{min}$ is returned (or if cost $> C_{max}$, the most probable face is returned). We repeated this for 30 different faces (runs) per set-up, and repeated the entire process for a total of 15 different set-ups. About 70.4% of these policies matched Figure 2.

We evaluated the results in two different ways. First, Figure 3(b) compares the percentage of wrong identifications of each policy, for each P_{min} value. We see no significant difference in error between BESTSEQ and INFOGAIN. The second graph, Figure 3(c), compares the average cost of each policy, for each P_{min} value. Here we see that BESTSEQ has much lower cost than INFOGAIN. Again, this is because BESTSEQ can use operator interactions to sweep a narrower region.

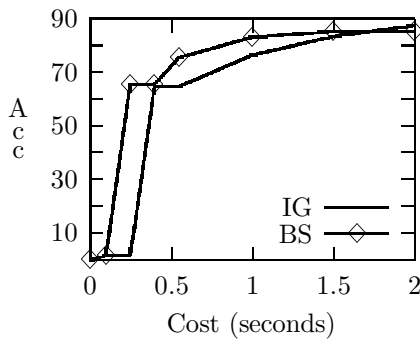
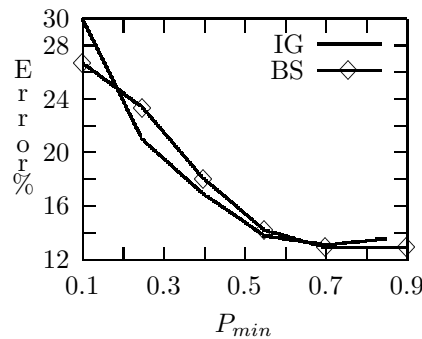
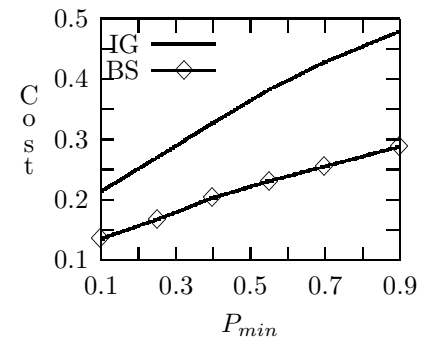


Figure 3: (a) Cost vs. Accuracy



(b) Min. Accuracy vs. Error



(c) Min. Accuracy vs. Cost

6 Conclusions

Future Work: This paper explored one type of operator interaction: where the output of one operator can be used to help a subsequent operator — here by reducing the time it will require to search. Moreover, our operators were relatively simple, as they always succeeded (or at least, think that they succeed), and they could be run in any order. As a consequence, our policy was quite simple — just straight-line sequence of operators.

There are many ways to extend our analysis. First, it might be useful to allow each operator to return not just a position, but also other information, such as confidence (which would become part of the state). The best policy, then, could use that information when deciding on the proper future action to take. If the abstracted state also included other information about the image, such as average intensity, the policy could use that information as well. This could result in a policy that was more complicated and, presumably, more accurate.

Finally, it would be interesting to consider operators that have some explicit dependency — e.g., one cannot run the “connect edgel” operator unless we have already run some “produce edgel” operator. These precedence constraints would add yet other challenges to our framework.

Contributions: This paper shows how dynamic programming can be used to build efficient interpretation systems. We argue that computing the *utility* of operator sequences, which incorporate the benefits of operator interactions, can play a significant role towards building efficient interpretation systems. However, in several real world situations, such systems may have (i) to deal with complex and unwieldy states, and (ii) to explore the operator space to find the most promising operator sequence, which is expensive. We addressed the first issue by using simplified abstract states in face recognition, and addressed the second by using an off-line limited lookahead search to find the most promising operator sequence.

We framed the image interpretation task as a Markov Decision Problem and used ideas from reinforcement learning to compute the utility of imaging operators *non-myopically* over a finite lookahead depth in an operator space. We built an interpretation system that uses these concepts and compared its performance with a successful myopic system in face recognition. Our results show that the former has a much better performance in various experiments that address the cost and accuracy tradeoffs.

Acknowledgments

The authors gratefully acknowledge the generous support from Canada’s Natural Science and Engineering Research Council and the Alberta Ingenuity Centre for Machine Learning.

References

- [BDB00] J. Bins B. Draper and K. Baek. Adore: Adaptive object recognition. In *Videre, 1(4)*, pages 86–99, 2000.
- [BDH99] C Boutilier, T Dean, and S Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *J. Artificial Intelligence Research*, 1999.
- [BDL02] V Bulitko, B Draper, and I. Levner. MR ADORE control policies. TR, Dept. of Computing Science, U. of Alberta, 2002.
- [DH73] R.O. Duda and P.E. Hart. *Pattern Classification and Scene Analysis*. Wiley, New York, 1973.
- [EC97] K Etemad and R Chellappa. Discriminant analysis for recognition of human faces. *J. Optical Society of America*, 1997.
- [HR96] R Huang and S Russell. Object identification: A bayesian analysis with application to traffic surveillance. *Artificial Intelligence*, 1996.
- [IG01a] R. Isukapalli and R. Greiner. Efficient car recognition policies. In *ICRA*, pages 2134–2139, Seoul, 2001.
- [IG01b] R. Isukapalli and R. Greiner. Efficient interpretation policies. In *IJCAI*, Seattle, 2001.
- [Mit97] T M Mitchell. *Machine Learning*. McGraw-Hill, 1997.
- [PL95] A R Pope and D Lowe. Learning object recognition models from images. In *Early Visual Learning*, 1995.
- [PMS94] A Pentland, B Moghaddam, and T Starner. View-based and modular eigenspaces for face recognition. In *IEEE CVPR*, 1994.
- [PWH98] P Phillips, H Wechsler, J Huang, and P Rauss. The feret database and evaluation procedure for face recognition algorithms. *Image and Vision Computing*, 1998.
- [SB98] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, 1998.
- [TP91] M Turk and A Pentland. Eigenfaces for recognition. *J. Cognitive Neuroscience*, 1991.

Switching Hypothesized Measurements: A Dynamic Model with Applications to Occlusion Adaptive Joint Tracking

Yang Wang Tele Tan

Institute for Infocomm Research, Singapore
{ywang, teletan}@i2r.a-star.edu.sg

Kia-Fock Loe

Dept. Computer Science, National Univ. Singapore
loekf@comp.nus.edu.sg

Abstract

This paper proposes a dynamic model supporting multimodal state space probability distributions and presents the application of the model in dealing with visual occlusions when tracking multiple objects jointly. For a set of hypotheses, multiple measurements are acquired at each time instant. The model switches among a set of hypothesized measurements during the propagation. Two computationally efficient filtering algorithms are derived for online joint tracking. Both the occlusion relationship and state of the objects are recursively estimated from the history of measurement data. The switching hypothesized measurements (SHM) model is generally applicable to describe various dynamic processes with multiple alternative measurement methods.

1 Introduction

Visual tracking is important in such application areas as human-computer interaction, surveillance, and visual reconstruction. Given a sequence of images containing the objects that are represented with a parametric motion model, parameters of the motion model are required to be estimated in successive frames. Tracking could be difficult due to the potential variability such as partial or full occlusions of objects, appearance changes caused by the variation of object poses or illumination conditions, as well as distractions from background clutter.

One principle challenge for visual tracking is to develop an accurate and effective model representation. The variability in visual environments usually results in a multimodal state space probability distribution. The Kalman filter [Brown, 1983; Rohr, 1994], a classical choice employed in tracking work, is restricted to representing unimodal probability distributions. Switching linear dynamic systems (SLDS) [Pavlovic and Rehg, 2000] and their equivalents [Shumway and Stoffer, 1991; Kim, 1994] have been used to describe dynamic processes. Intuitively, a complex dynamic system is represented with a set of linear models controlled by a switching variable. Joint probabilistic data association (JPDA) [Bar-Shalom and

Fortmann, 1988] and multiple hypothesis tracking (MHT) [Cox and Hingorani, 1996] techniques, which represent multimodal distributions by constructing data association hypotheses, can be cast in the framework of SLDS as well. Moreover, Monte Carlo methods such as the Condensation algorithm [Isard and Blake, 1996] support multimodal probability densities with sample based representation. By retaining only the peaks of the probability density, relatively fewer samples are required in the work of Cham and Rehg [1999]. A switching model framework of the Condensation algorithm is also proposed by Isard and Blake [1998].

On the other hand, the measurement process is another essential issue to deal with the potential variability. Measurements are not readily available from image sequences in visual tracking. Even an accurate tracking model may have a poor performance if the measurements are too noisy. Parametric models can be used to characterize appearance changes of target regions [Hager and Belhumeur, 1998]. In the work of Galvin et al. [1999], two virtual snakes, a background and a foreground snake for each object, are generated to resolve the occlusion when two objects intersect. Rasmussen and Hager [2001] describe a joint measurement process for tracking multiple objects enumerating all possible occlusion relationships. The measurement with respect to the most possible occlusion relationship is determined from the current frame. Moreover, layered approach [Wang and Adelson, 1994; Ayer and Sawhney, 1995; Jovic and Frey, 2001; Tao et al., 2002] is an efficient way to represent multiple moving objects. A moving object is characterized by a coherent motion model over its support region.

In this paper, the idea of switching hypothesized measurements (SHM), which results in a SHM model supporting multimodal distributions, is proposed to handle the potential variability in visual tracking. The approach acquires a set of hypothesized measurements for different occlusion hypotheses at each time instant. Comparing with the above mentioned state space models, the SHM approach switches among a set of hypothesized measurements rather than switches among a set of models. Two computationally efficient filtering algorithms are derived for jointly tracking multiple objects. Both the occlusion relationship and state of the objects are estimated from the history of measurements.

2 Model

2.1 Hypothesized Measurement

For a hidden state sequence $\{\mathbf{z}_k\}$ ($k \in \mathbf{N}$), the objective of online tracking is to recursively estimate $\mathbf{z}_k$ from the set of all available measurements $\mathbf{y}_{1:k} = \{\mathbf{y}_i\}_{1 \leq i \leq k}$ up to time k . For a certain complex system, the estimation may be influenced by a mode or switching state sequence $\{s_k\}$ as well, with $s_k \in \{1, 2, \dots, L\}$ ($L \in \mathbf{N}$). Specifically, the mode switching originates from the measurement process in our work. The notion of a measurement is extended to a set of L hypothesized measurements $\mathbf{y}_k = (\mathbf{y}_{k,1}, \mathbf{y}_{k,2}, \dots, \mathbf{y}_{k,L})$ at each time instant. Each $\mathbf{y}_{k,j}$ ($1 \leq j \leq L$) is called a hypothesized measurement since it is obtained by assuming that the switching state s_k is j at time k .

To illustrate the idea of hypothesized measurement, consider the measurement process for jointly tracking two objects, e.g. a rectangle and a circle, in an image sequence $\{g_k\}$. To deal with occlusions between the two objects when measuring the k th frame g_k , the switching state s_k is introduced to describe the depth ordering at time k . $s_k \in \{1, 2\}$, where s_k equals 1 if the rectangle is in front of the circle, and 2 if the circle is in front of the rectangle. The hypothesized measurement $\mathbf{y}_{k,j}$ ($1 \leq j \leq 2$) is denoted as $(\mathbf{y}_{k,j}^{(1)}, \mathbf{y}_{k,j}^{(2)})^T$, where $\mathbf{y}_{k,j}^{(1)}$ is the measurement for the rectangle, and $\mathbf{y}_{k,j}^{(2)}$ is the measurement for the circle under the j th hypothesis.

Under the hypothesis of $s_k = 1$, i.e. the circle is occluded by the rectangle at time k , the rectangle should be measured first to acquire $\mathbf{y}_{k,1}^{(1)}$. Then the observed rectangle is masked in the image. The occluded area of the circle is ignored and only the visible region is matched normally to get $\mathbf{y}_{k,1}^{(2)}$. Similarly, under the hypothesis of $s_k = 2$, i.e. the rectangle is occluded by the circle, the circle should be matched first to get $\mathbf{y}_{k,2}^{(2)}$, then the masked image is used to measure $\mathbf{y}_{k,2}^{(1)}$. Thus, the occlusion will not affect the measurement result. It is obvious that both hypothesized measurements support the condition of nonocclusion since different depth orderings of nonoverlapping objects are visually equivalent. The probabilities of the hypotheses should be equal in the case of nonocclusion.

Unfortunately, the occlusion relationship is not given before hand. The objective of our SHM approach is to estimate both the switching state and the hidden state from the history of the hypothesized measurements.

2.2 Linear SHM Model for Joint Tracking

For joint tracking of M ($M \in \mathbf{N}$) objects in the scene, the switching state s_k represents the occlusion relationship at time k . $s_k \in \{1, 2, \dots, L\}$, and $L = M!$. We assume the switching state follows a first order Markov chain with the following transition probability,

$$p(s_{k+1} = i | s_k = j) = \alpha_{ij}, \quad (1)$$

with $\sum_i \alpha_{i,j} = 1$. The hidden state $\mathbf{z}_k$ is denoted as

$(\mathbf{z}_k^{(1)}, \mathbf{z}_k^{(2)}, \dots, \mathbf{z}_k^{(M)})^T$, with $\mathbf{z}_k^{(m)}$ ($1 \leq m \leq M$) being the state for the m th object at time k . For a linear process with Gaussian noise, the hidden state transition model becomes

$$\mathbf{z}_{k+1} = \mathbf{F}\mathbf{z}_k + \mathbf{n}, \quad (2a)$$

$$p(\mathbf{z}_{k+1} | \mathbf{z}_k) = N(\mathbf{z}_{k+1}; \mathbf{F}\mathbf{z}_k, \mathbf{Q}), \quad (2b)$$

where $\mathbf{F} \in \mathbf{R}^{n_z} \times \mathbf{R}^{n_z}$ is the state transition matrix, $\mathbf{n}$ is a zero-mean Gaussian noise with covariance matrix $\mathbf{Q}$, and $N(\mathbf{z}; \mathbf{m}, \mathbf{\Sigma})$ is a Gaussian density with argument $\mathbf{z}$, mean $\mathbf{m}$, and covariance $\mathbf{\Sigma}$.

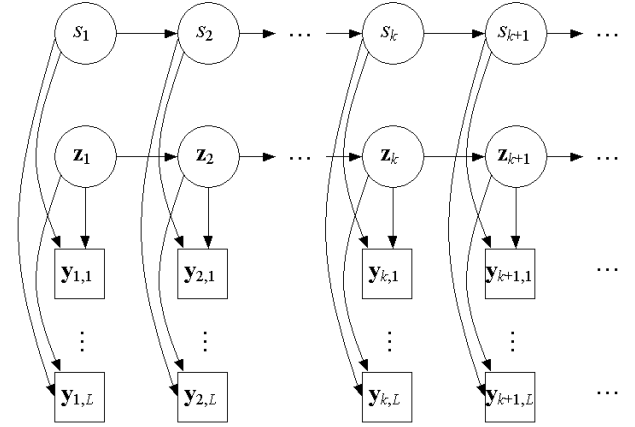


Figure 1: Bayesian network representation of the SHM model

Given the switching state s_k at time k , the corresponding hypothesized measurement $\mathbf{y}_{k,s_k}$ could be considered as a proper measurement centering on the hidden state, while every other $\mathbf{y}_{k,j}$ for $j \neq s_k$ is an improper measurement generated under a wrong assumption. The improper measurement should be weakly influenced by the hidden state $\mathbf{z}_k$ and have a large variance. To simplify the computation, we assume a normal distribution for a proper measurement and a uniform distribution for an improper measurement. The measurement model is simplified as

$$\mathbf{y}_{k,j} = \begin{cases} \mathbf{z}_k + \mathbf{v}_{k,j}, & \text{if } j = s_k, \\ \mathbf{w}, & \text{otherwise,} \end{cases} \quad (3a)$$

$$p(\mathbf{y}_{k,j} | s_k, \mathbf{z}_k) = \begin{cases} N(\mathbf{y}_{k,j}; \mathbf{z}_k, \mathbf{R}_{k,j}), & \text{if } j = s_k, \\ \text{a constant,} & \text{otherwise,} \end{cases} \quad (3b)$$

where $\mathbf{v}_{k,j}$ is a zero-mean Gaussian noise with covariance matrix $\mathbf{R}_{k,j}$, and $\mathbf{w}$ is a uniformly distributed noise, whose density is a small positive constant. For the measurement of M objects, $\mathbf{y}_{k,j}$ is denoted as $(\mathbf{y}_{k,j}^{(1)}, \mathbf{y}_{k,j}^{(2)}, \dots, \mathbf{y}_{k,j}^{(M)})^T$, and $\mathbf{v}_{k,j}$ is written as $(\mathbf{v}_{k,j}^{(1)}, \mathbf{v}_{k,j}^{(2)}, \dots, \mathbf{v}_{k,j}^{(M)})^T$. Given the current state, the conditional independence among the hypothesized measurements is assumed to make the model computationally efficient.

$$\begin{aligned}
p(\mathbf{y}_k | s_k = j, \mathbf{z}_k) &= p(\mathbf{y}_{k,1}, \mathbf{y}_{k,2}, \dots, \mathbf{y}_{k,L} | s_k = j, \mathbf{z}_k) \\
&= p(\mathbf{y}_{k,j} | s_k = j, \mathbf{z}_k) \prod_{l \neq j} p(\mathbf{y}_{k,l} | s_k = j, \mathbf{z}_k) \\
&\propto N(\mathbf{y}_{k,j}; \mathbf{z}_k, \mathbf{R}_{k,j}). \tag{4}
\end{aligned}$$

The SHM model can be represented by a dynamic Bayesian network shown in figure 1.

3 Method

3.1 Measurement

Multiple, occluding objects are modeled using layer representation. Layers are indexed by $m = 1, 2, \dots, M$, with layer 1 being the layer that is closest to the camera and layer m being behind layer 1, 2, ..., $m-1$. There is one object in each layer. The number of all occlusion relationship hypotheses (or depth ordering permutations) is $L = M!$. Each permutation is tagged with a index j ($1 \leq j \leq L$).

Under each permutation hypothesis, the object in the front layer 1 should be measured first from the image g_k at time k . Then the object in layer 2 can be matched from the masked image, and so on. At last, the object in layer M can be measured. Occluded points are not matched when measuring the objects. Measurement results of nonoverlapping objects should be equivalent for different depth ordering permutations. Given the reference image g_r ($r < k$), the measurement is based on minimizing the mean of squared intensity differences between the current image and the reference region. Under the j th hypothesis, $\mathbf{y}_{k,j}^{(m)}$ is the hypothesized measurement of the m th object, and $e_{k,j}^{(m)}$ is the corresponding squared difference mean at time k . The vector $(e_{k,j}^{(1)}, e_{k,j}^{(2)}, \dots, e_{k,j}^{(M)})^T$ is written as $\mathbf{e}_{k,j}$. The covariance matrix $\mathbf{R}_{k,j}$ is obtained by assuming that the components of the measurement noise are uncorrelated to each other, and the variances is proportional to the corresponding squared difference mean.

3.2 SHM Filter

From a Bayesian perspective, the online tracking problem is to recursively calculate the posterior state space distribution. Given the measurement data up to time k , the probability density function (pdf) of the state is expressed as

$$\begin{aligned}
p(s_k = j, \mathbf{z}_k | \mathbf{y}_{1:k}) &= p(s_k = j | \mathbf{y}_{1:k}) p(\mathbf{z}_k | s_k = j, \mathbf{y}_{1:k}) \\
&= \beta_{k,j} N(\mathbf{z}_k; \mathbf{m}_{k,j}, \mathbf{P}_{k,j}), \tag{5}
\end{aligned}$$

where $p(s_k = j | \mathbf{y}_{1:k})$ is denoted as $\beta_{k,j}$, with $\sum_j \beta_{k,j} = 1$, and

the conditional density $p(\mathbf{z}_k | s_k = j, \mathbf{y}_{1:k})$ is modeled as a normal distribution $N(\mathbf{z}_k; \mathbf{m}_{k,j}, \mathbf{P}_{k,j})$ under each switching state hypothesis. Thus the pdf $p(\mathbf{z}_k | \mathbf{y}_{1:k})$ is a mixture of L Gaussians.

At time $k+1$, the set of hypothesized measurements $\mathbf{y}_{k+1}$ becomes available, and it is used to update $\{\beta_{k,j}, \mathbf{m}_{k,j}, \mathbf{P}_{k,j}\}_{1 \leq j \leq L}$ to $\{\beta_{k+1,i}, \mathbf{m}_{k+1,i}, \mathbf{P}_{k+1,i}\}_{1 \leq i \leq L}$ via Bayes' rule.

$$\begin{aligned}
p(s_{k+1}, s_k, \mathbf{z}_{k+1} | \mathbf{y}_{1:k+1}) &= \frac{1}{p(\mathbf{y}_{k+1} | \mathbf{y}_{1:k})} p(\mathbf{y}_{k+1} | s_{k+1}, \mathbf{z}_{k+1}) p(s_{k+1}, s_k, \mathbf{z}_{k+1} | \mathbf{y}_{1:k}) \\
&\propto p(\mathbf{y}_{k+1} | s_{k+1}, \mathbf{z}_{k+1}) p(s_{k+1}, s_k, \mathbf{z}_{k+1} | \mathbf{y}_{1:k}). \tag{6}
\end{aligned}$$

In principle, the filtering process has three stages: prediction, update, and collapsing [Murphy, 1998]. From (1), (2), (4) and (6), the approximate inference algorithm can be derived in a similar way as that in a Gaussian sum filter [Anderson and Moore, 1979].

$$\begin{aligned}
\beta_{k+1,i} &= p(s_{k+1} = i | \mathbf{y}_{1:k+1}) \\
&= \frac{\sum_j \alpha_{i,j} \beta_{k,j} N(\mathbf{y}_{k+1,i}; \mathbf{m}_{k+1|k,j}, \mathbf{S}_{k+1,i|j})}{\sum_i \sum_j \alpha_{i,j} \beta_{k,j} N(\mathbf{y}_{k+1,i}; \mathbf{m}_{k+1|k,j}, \mathbf{S}_{k+1,i|j})}, \tag{7a}
\end{aligned}$$

$$\begin{aligned}
p(\mathbf{z}_{k+1} | s_{k+1} = i, s_k = j, \mathbf{y}_{1:k+1}) &\approx N(\mathbf{z}_{k+1}; \mathbf{m}_{k+1,i|j}, \mathbf{P}_{k+1,i|j}), \tag{7b} \\
p(\mathbf{z}_{k+1} | s_{k+1} = i, \mathbf{y}_{1:k+1}) &\approx \sum_j \beta_{k+1,i|j} N(\mathbf{z}_{k+1}; \mathbf{m}_{k+1,i|j}, \mathbf{P}_{k+1,i|j})
\end{aligned}$$

$$\approx N(\mathbf{z}_{k+1}; \mathbf{m}_{k+1,i}, \mathbf{P}_{k+1,i}), \tag{7c}$$

where

$$\begin{aligned}
\mathbf{m}_{k+1|k,j} &= \mathbf{F} \mathbf{m}_{k,j}, \\
\mathbf{P}_{k+1|k,j} &= \mathbf{F} \mathbf{P}_{k,j} \mathbf{F}^T + \mathbf{Q}, \\
\mathbf{S}_{k+1,i|j} &= \mathbf{P}_{k+1|k,j} + \mathbf{R}_{k+1,i}, \\
\mathbf{K}_{k+1,i|j} &= \mathbf{P}_{k+1|k,j} \mathbf{S}_{k+1,i|j}^{-1}, \\
\mathbf{m}_{k+1,i|j} &= \mathbf{m}_{k+1|k,j} + \mathbf{K}_{k+1,i|j} (\mathbf{y}_{k+1,i} - \mathbf{m}_{k+1|k,j}), \\
\mathbf{P}_{k+1,i|j} &= \mathbf{P}_{k+1|k,j} - \mathbf{K}_{k+1,i|j} \mathbf{P}_{k+1|k,j}, \\
\beta_{k+1,i|j} &= \frac{\alpha_{i,j} \beta_{k,j} N(\mathbf{y}_{k+1,i}; \mathbf{m}_{k+1|k,j}, \mathbf{S}_{k+1,i|j})}{\sum_j \alpha_{i,j} \beta_{k,j} N(\mathbf{y}_{k+1,i}; \mathbf{m}_{k+1|k,j}, \mathbf{S}_{k+1,i|j})}, \\
\mathbf{m}_{k+1,i} &= \sum_j \beta_{k+1,i|j} \mathbf{m}_{k+1,i|j}, \\
\mathbf{P}_{k+1,i} &= \sum_j \beta_{k+1,i|j} [\mathbf{P}_{k+1,i|j} + \\
&\quad (\mathbf{m}_{k+1,i|j} - \mathbf{m}_{k+1,i})(\mathbf{m}_{k+1,i|j} - \mathbf{m}_{k+1,i})^T].
\end{aligned}$$

The state at time $k+1$ is estimated as

$$\begin{aligned}
\hat{s}_{k+1} &= \arg \max_i p(s_{k+1} = i | \mathbf{y}_{1:k+1}) = \arg \max_i \beta_{k+1,i}, \tag{8a} \\
\hat{\mathbf{z}}_{k+1} &= \arg \max_{\mathbf{z}_{k+1}} p(\mathbf{z}_{k+1} | s_{k+1} = \hat{s}_{k+1}, s_k = \hat{s}_k, \mathbf{y}_{1:k+1}) \\
&= \mathbf{m}_{k+1, \hat{s}_{k+1} | \hat{s}_k}. \tag{8b}
\end{aligned}$$

From (7) it can be seen that the computation of the SHM filter is slightly more complex than that of multiple Kalman filters or Gaussian sum filters.

3.3 Fast SHM Filter

When occlusion is the main factor in the potential variability of joint tracking, we can assume that the measurement noise under the true occlusion hypothesis is small. When the noise becomes zero, the measurement model can be simplified as

$$p(\mathbf{y}_{k,j} | s_k, \mathbf{z}_k) = \begin{cases} \delta(\mathbf{y}_{k,j} - \mathbf{z}_k), & \text{if } j = s_k, \\ \text{a constant, otherwise,} \end{cases} \quad (9)$$

where $\delta(\cdot)$ is the Dirac delta function. Consider the minimized mean of squared differences as a part of the hypothesized measurement, so that the definition of measurement can be generalized as

$$\mathbf{y}'_{k,j} = (\mathbf{y}_{k,j}, \mathbf{e}_{k,j}), \quad \mathbf{y}'_k = (\mathbf{y}'_{k,1}, \dots, \mathbf{y}'_{k,L}). \quad (10)$$

Assume that $\mathbf{e}_{k,j}$ is independent on $\mathbf{y}_{k,j}$ and $\mathbf{z}_k$, and the posterior density of the squared difference mean is of exponential distribution for each object (More accurate expression could be derived using the χ^2 distribution.) under the true hypothesis, the pdf of $\mathbf{e}_{k,j}$ is factorized as

$$p(\mathbf{e}_{k,j} | s_k, \mathbf{z}_k) = p(\mathbf{e}_{k,j} | s_k) = \prod_m p(e_{k,j}^{(m)} | s_k), \quad (11a)$$

$$p(e_{k,j}^{(m)} | s_k) \begin{cases} \propto \exp[-e_{k,j}^{(m)}], & \text{if } j = s_k. \\ \text{is a constant, otherwise.} \end{cases} \quad (11b)$$

The generalized measurement model now becomes

$$\begin{aligned} & p(\mathbf{y}'_k | s_k = j, \mathbf{z}_k) \\ &= p(\mathbf{y}'_{k,j} | s_k = j, \mathbf{z}_k) \prod_{l \neq j} p(\mathbf{y}'_{k,l} | s_k = j, \mathbf{z}_k) \\ &\propto p(\mathbf{y}_{k,j} | s_k = j, \mathbf{z}_k) p(\mathbf{e}_{k,j} | s_k = j, \mathbf{z}_k) \\ &\propto \eta_{k,j} \delta(\mathbf{y}_{k,j} - \mathbf{z}_k), \end{aligned} \quad (12)$$

where $\eta_{k,j} = \prod_m \exp[-e_{k,j}^{(m)}]$.

At time k , the conditional pdf of the state is expressed as

$$p(s_k = j, \mathbf{z}_k | \mathbf{y}'_{1:k}) = \beta_{k,j} \delta(\mathbf{z}_k - \mathbf{y}_{k,j}), \quad (13)$$

where $\beta_{k,j}$ is $p(s_k = j | \mathbf{y}'_{1:k})$, and $p(\mathbf{z}_k | s_k = j, \mathbf{y}'_{1:k})$ equals $\delta(\mathbf{z}_k - \mathbf{y}_{k,j})$. From appendix A, the filtering algorithm is

$$\begin{aligned} \beta_{k+1,i} &= p(s_{k+1} = i | \mathbf{y}'_{1:k+1}) \\ &= \frac{\sum_j \alpha_{i,j} \beta_{k,j} \eta_{k+1,i} N(\mathbf{y}_{k+1,i}; \mathbf{Fy}_{k,j}, \mathbf{Q})}{\sum_i \sum_j \alpha_{i,j} \beta_{k,j} \eta_{k+1,i} N(\mathbf{y}_{k+1,i}; \mathbf{Fy}_{k,j}, \mathbf{Q})}. \end{aligned} \quad (14)$$

The state at time k is estimated as

$$\hat{s}_k = \arg \max_j p(s_k = j | \mathbf{y}'_{1:k}) = \arg \max_j \beta_{k,j}, \quad (15a)$$

$$\hat{\mathbf{z}}_k = \arg \max_{\mathbf{z}_k} p(\mathbf{z}_k | s_k = \hat{s}_k, \mathbf{y}'_{1:k}) = \mathbf{y}_{k, \hat{s}_k}. \quad (15b)$$

The result $\{\mathbf{y}_{k, \hat{s}_k}\}$ can then be plugged into a Kalman filter to achieve improved performance. Such a SHM-Kalman filter keeps the multimodality of the SHM model and has attractive computation requirement. In addition, the collapsing stage is not necessary in the fast SHM filter.

4 Implementation

In practice, we use the second order (constant velocity) model. The hidden state transition function is

$$\begin{pmatrix} \mathbf{z}_{k+2} \\ \mathbf{z}_{k+1} \end{pmatrix} = \begin{pmatrix} 2\mathbf{I} & -\mathbf{I} \\ \mathbf{I} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \mathbf{z}_{k+1} \\ \mathbf{z}_k \end{pmatrix} + \begin{pmatrix} \mathbf{n} \\ \mathbf{0} \end{pmatrix}, \quad (16)$$

where $\mathbf{z}_k$ is the tracked entity (e.g. position and orientation). The hidden state and state transition matrix can be correspondingly defined. The switching state transition probability is set as

$$\alpha_{i,j} = \begin{cases} 1 - \lambda, & \text{if } i = j, \\ \frac{\lambda}{L - 1}, & \text{otherwise,} \end{cases} \quad (17)$$

where λ is a small positive value (0.1 in this paper) so that two successive switching states are more likely to be of the same label. At the beginning, the reference image g_r is set as the initial image g_0 . When there is a high confidence in nonocclusion, the reference image can be adaptively updated. The objects are assumed to be separated from each other in g_0 . The initial $\beta_{0,j}$, $\mathbf{m}_{0,j}$, and $\mathbf{P}_{0,j}$ should be equal for different j because of nonocclusion. $\beta_{0,j} = p(s_0 = j) = \frac{1}{L}$.

The initial mean $\mathbf{m}_{0,j}$ is set as a zero vector. The initial covariance matrix $\mathbf{P}_{0,j}$ is set as diagonal with small variances since the initialization is assumed to be accurate. When an object is totally occluded by the other objects, no points of the target region will be matched. The estimation is based on the result of time k using the state transition function when no visible region of the object is expected at time $k+1$.

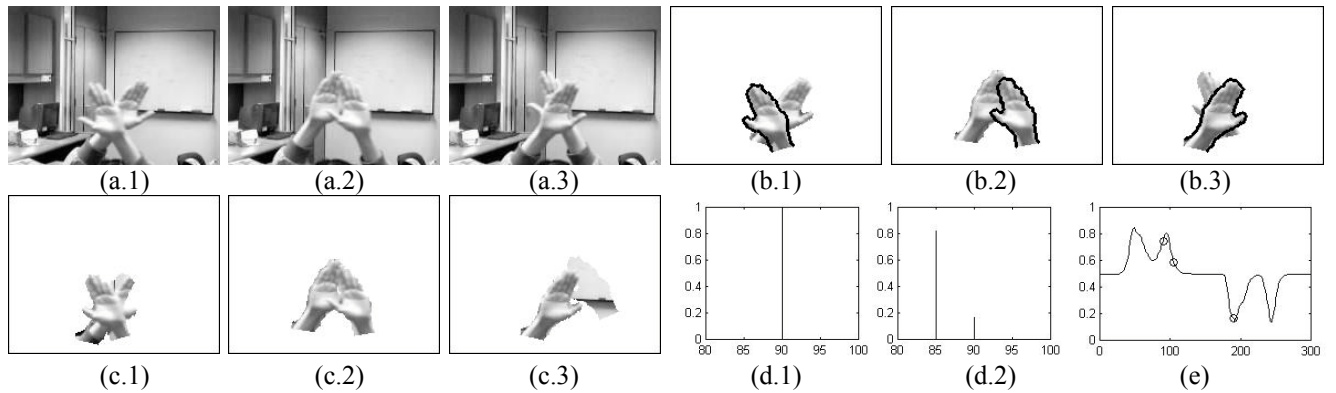


Figure 2: Results of tracking the two hands of a person

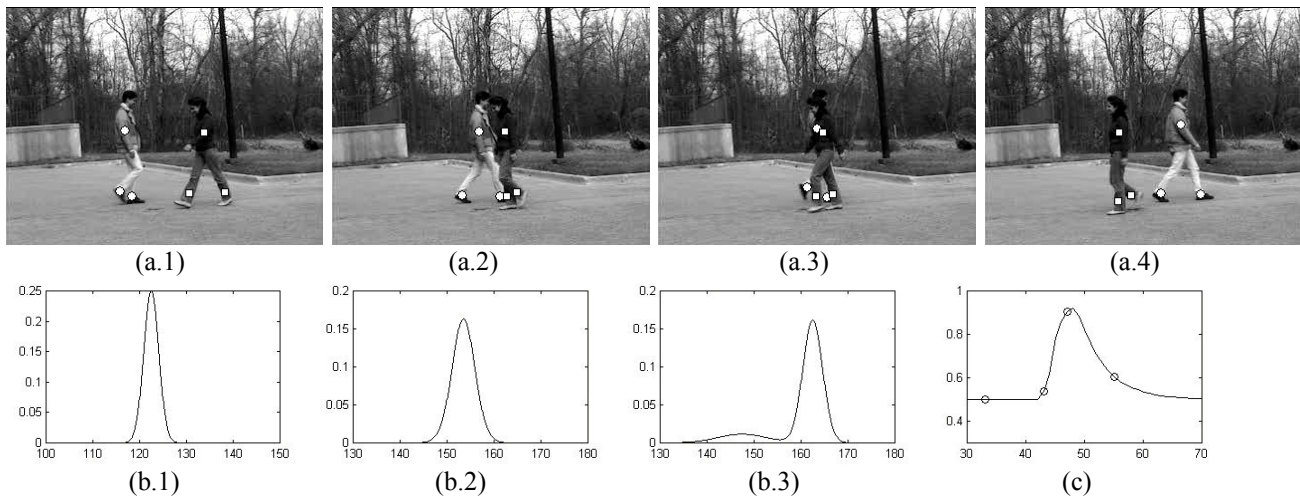


Figure 3: Results of tracking the four shanks of two persons

5 Results and Discussion

Test results of two video sequences are shown in our experiments. The state is the position and orientation, and the second order model is employed. Each measurement is a translation and rotation.

Figure 2 shows the tracking of two hands of a person as they cross several times in an image sequence. Figure 2a shows the three frames of the sequence. Appearance variation of the hands due to pose changes can be seen. Figure 2b and 2c demonstrate the tracking efficacy of the fast SHM filter versus the Kalman filter. The fast SHM filter successfully tracks both hands under different occlusion relationships. In figure 2b, one hand is drawn in black contour when the detected depth order indicates that it is in front of the other hand. The Kalman filter has a similar performance when occlusions are not severe, but poor under heavy occlusions. In figure 2c.3, the distraction from background clutter causes the Kalman tracker to fail. The normalized posterior distributions for the vertical position of the left hand in figure 2a.2 and 2a.3 are shown in figure 2d.1 and 2d.2. When the occlusion is not severe, measurements under the two hypotheses are the same, and the distribution is unimodal. Under heavy occlusions, the distribution becomes multimodal since the two hypothesized measurements tend to be different. The measurement under true hypothesis matches the hand correctly, while the measurement under false hypothesis is distracted by background clutter. Figure 2e shows the probabilities of the first occlusion hypothesis (the left hand being in the front) over the first 300 frames. The probabilities for the three frames in figure 2a are circled. The probabilities of the two hypotheses are equal in the nonoverlapping cases, while the probability of the true hypothesis becomes dominant under occlusions.

Figure 3 shows the results of jointly tracking the four shanks of a man and a woman as they cross. The man's right shank has been totally occluded in the sequence. There

should be totally $4! = 24$ hypotheses if we directly apply the SHM filter in section 3.2. To reduce the computation, two reasonable assumptions are made to prune less plausible hypotheses. Firstly, one's legs can not simultaneously occlude and be occluded by the other's legs. Secondly, the occlusion relationship between the man and woman can be determined from their bodies. Thus the whole tracking procedure is divided into three trackers. The first one tracks the two bodies of the walkers. According to the detected occlusion relationship, the two shanks of the person in the front are then tracked. At last, the shanks of the other person are tracked in the masked image. Figure 3a shows the results for the four frames of the sequence (circles are marked on the man's body and shanks, and rectangles are marked on the woman). Figure 3b shows the posterior distributions for the horizontal position of the occluded body in figure 3a.1-3a.3. When the two bodies are separated, the density is unimodal and of a small variance. The density variance increases when occlusions occur. It becomes multimodal under heavy occlusions. Figure 3c shows the probabilities of the woman's body being in the front. The probabilities for the four frames in figure 3a are circled.

Under realistic environments, it is understandable that comparing with the other hypothesized measurements, the measurement under the true occlusion hypothesis usually shows more regularity and has smaller variances (or squared difference means). Thus, the true information (the switching state and the hidden state) could be enhanced through the propagation. In addition, the acquirement of multiple measurements helps decrease the information loss (e.g. caused by background clutter) during the measurement process.

6 Conclusion

This paper proposes a SHM model for state space representation of dynamic systems and derives two efficient filtering algorithms. Our joint tracking approach explicitly

reasons about the occlusion relationships. The occlusion relationship is quantitatively analyzed throughout the propagation. The information can be used for reference update and further analysis. Moreover, experimental results show that our method helps handle appearance changes and distractions.

The SHM model discusses the measurement switching in dynamic systems. It is complementary to the idea of model switching in [Ghahramani and Hinton, 1998]. Our future study is to effectively combine these two ideas in visual tracking. Furthermore, the SHM model is generally applicable to various dynamic processes in which there are multiple alternative measurement methods.

A Inference in Fast SHM Filter

The predictive density of the state is

$$\begin{aligned}
p(s_{k+1} = i, s_k = j, \mathbf{z}_{k+1} | \mathbf{y}'_{1:k}) \\
&= \int p(s_{k+1} = i, \mathbf{z}_{k+1} | s_k = j, \mathbf{z}_k) p(s_k = j, \mathbf{z}_k | \mathbf{y}'_{1:k}) d\mathbf{z}_k \\
&= p(s_{k+1} = i | s_k = j) p(s_k = j | \mathbf{y}'_{1:k}) \cdot \\
&\quad \int p(\mathbf{z}_{k+1} | \mathbf{z}_k) p(\mathbf{z}_k | s_k = j, \mathbf{y}'_{1:k}) d\mathbf{z}_k \\
&= \alpha_{i,j} \beta_{k,j} \int N(\mathbf{z}_{k+1}; \mathbf{F}\mathbf{z}_k, \mathbf{Q}) \delta(\mathbf{z}_k - \mathbf{y}_{k,j}) d\mathbf{z}_k \\
&= \alpha_{i,j} \beta_{k,j} N(\mathbf{z}_{k+1}; \mathbf{F}\mathbf{y}_{k,j}, \mathbf{Q}). \quad (18)
\end{aligned}$$

After receiving the measurement $\mathbf{y}'_{k+1}$, the posterior density is updated using (6).

$$\begin{aligned}
p(s_{k+1} = i, s_k = j, \mathbf{z}_{k+1} | \mathbf{y}'_{1:k+1}) \\
&\propto p(\mathbf{y}'_{k+1} | s_{k+1} = i, \mathbf{z}_{k+1}) p(s_{k+1} = i, s_k = j, \mathbf{z}_{k+1} | \mathbf{y}'_{1:k}) \\
&\propto \alpha_{i,j} \beta_{k,j} \eta_{k+1,i} \delta(\mathbf{y}_{k+1,i} - \mathbf{z}_{k+1}) N(\mathbf{z}_{k+1}; \mathbf{F}\mathbf{y}_{k,j}, \mathbf{Q}) \\
&= \alpha_{i,j} \beta_{k,j} \eta_{k+1,i} \delta(\mathbf{z}_{k+1} - \mathbf{y}_{k+1,i}) N(\mathbf{y}_{k+1,i}; \mathbf{F}\mathbf{y}_{k,j}, \mathbf{Q}). \quad (19)
\end{aligned}$$

The probability of the switching state is updated as

$$\begin{aligned}
\beta_{k+1,i} &= p(s_{k+1} = i | \mathbf{y}'_{1:k+1}) \\
&= \sum_j \int p(s_{k+1} = i, s_k = j, \mathbf{z}_{k+1} | \mathbf{y}'_{1:k+1}) d\mathbf{z}_{k+1} \\
&\propto \eta_{k+1,i} \sum_j \alpha_{i,j} \beta_{k,j} N(\mathbf{y}_{k+1,i}; \mathbf{F}\mathbf{y}_{k,j}, \mathbf{Q}). \quad (20)
\end{aligned}$$

Since $\sum_i \beta_{k+1,i} = 1$, $\beta_{k+1,i}$ can be calculated by normalizing.

$$\beta_{k+1,i} = \frac{\sum_j \alpha_{i,j} \beta_{k,j} \eta_{k+1,i} N(\mathbf{y}_{k+1,i}; \mathbf{F}\mathbf{y}_{k,j}, \mathbf{Q})}{\sum_i \sum_j \alpha_{i,j} \beta_{k,j} \eta_{k+1,i} N(\mathbf{y}_{k+1,i}; \mathbf{F}\mathbf{y}_{k,j}, \mathbf{Q})}.$$

References

- [Anderson and Moore, 1979] B. D. O. Anderson and J. B. Moore, *Optimal filtering*, Prentice-Hall, 1979.
- [Ayer and Sawhney, 1995] S. Ayer and H. S. Sawhney, Layered representation of motion video using robust maximum-likelihood estimation of mixture models and MDL encoding, In *Proc. International Conf. Computer Vision*, pp. 777–784, 1995.
- [Bar-Shalom and Fortmann, 1988] Y. Bar-Shalom and T. E. Fortmann, *Tracking and data association*, Academic Press, 1988.
- [Brown, 1983] R. G. Brown, *Introduction to random signal analysis and Kalman filtering*, John Wiley & Sons, 1983.
- [Cham and Rehg, 1999] T.-J. Cham and J. M. Rehg, A multiple hypothesis approach to figure tracking, In *Proc. Conf. Computer Vision and Pattern Recognition*, vol. 2, pp. 239–245, 1999.
- [Cox and Hingorani, 1996] I. J. Cox and S. L. Hingorani, An efficient implementation of Reid’s multiple hypothesis tracking algorithm and its evaluation for the purpose of visual tracking, *IEEE Trans. Patt. Anal. Mach. Intel.*, vol. 18, pp. 138–150, 1996.
- [Galvin et al., 1999] B. Galvin, B. McCane, and K. Novins, Virtual snakes for occlusion analysis, In *Proc. Conf. Computer Vision and Pattern Recognition*, vol. 2, pp. 294–299, 1999.
- [Ghahramani and Hinton, 1998] Z. Ghahramani and G. E. Hinton, Variational learning for switching state-space models, *Neural Computation*, vol. 12, pp. 963–996, 1998.
- [Hager and Belhumeur, 1998] G. D. Hager and P. N. Belhumeur, Efficient region tracking with parametric models of geometry and illumination, *IEEE Trans. Patt. Anal. Mach. Intel.*, vol. 20, pp. 1025–1039, 1998.
- [Isard and Blake, 1998] M. Isard and A. Blake, A mixed-state Condensation tracker with automatic model-switching, In *Proc. International Conf. Computer Vision*, pp. 107–112, 1998.
- [Isard and Blake, 1996] M. Isard and A. Blake, Contour tracking by stochastic propagation of conditional density, In *Proc. European Conf. Computer Vision*, pp. 343–356, 1996.
- [Jojic and Frey, 2001] N. Jojic and B. J. Frey, Learning flexible sprites in video layers, In *Proc. Conf. Computer Vision and Pattern Recognition*, vol. 1, pp. 199–206, 2001.
- [Kim, 1994] C.-J. Kim, Dynamic linear models with Markov-switching, *Journal of Econometrics*, vol. 60, pp. 1–22, 1994.
- [Murphy, 1998] K. P. Murphy, Learning switching Kalman filter models, Technical Report 98-10, Compaq Cambridge Research Lab, 1998.
- [Pavlovic and Rehg, 2000] V. Pavlovic and J. M. Rehg, Impact of dynamic model learning on classification of human motion, In *Proc. Conf. Computer Vision and Pattern Recognition*, vol. 1, pp. 788–795, 2000.
- [Rasmussen and Hager, 2001] C. Rasmussen and G. D. Hager, Probabilistic data association methods for tracking complex visual objects, *IEEE Trans. Patt. Anal. Mach. Intel.*, vol. 23, pp. 560–576, 2001.
- [Rohr, 1994] K. Rohr, Towards model-based recognition of human movements in image sequences, *Computer Vision, Graphics, and Image Processing: Image Understanding*, vol. 59, pp. 94–115, 1994.
- [Shumway and Stoffer, 1991] R. H. Shumway and D. S. Stoffer, Dynamic linear models with switching, *Journal of the American Statistical Association*, vol. 86, pp. 763–769, 1991.
- [Tao et al., 2002] H. Tao, H. S. Sawhney, and R. Kumar, Object tracking with Bayesian estimation of dynamic Layer representations, *IEEE Trans. Patt. Anal. Mach. Intel.*, vol. 24, pp. 75–89, 2002.
- [Wang and Adelson, 1994] J. Y. A. Wang and E. H. Adelson, Representing moving images with layers, *IEEE Trans. Image Processing*, vol. 3, pp. 625–637, 1994.

