

PERCEPTION

Where is ...? Learning and Utilizing Motion Patterns of Persons with Mobile Robots

Grzegorz Cielniak^{†‡}

Maren Bennewitz[†]

Wolfram Burgard[†]

[†]Department of Computer Science, University of Freiburg, 79110 Freiburg, Germany*

[‡]Department of Technology, Örebro University, 70182 Örebro, Sweden

Abstract

Whenever people move through their environments they do not move randomly. Instead, they usually follow specific trajectories or motion patterns corresponding to their intentions. Knowledge about such patterns may enable a mobile robot to robustly keep track of persons in its environment or to improve its obstacle avoidance behavior. This paper proposes a technique for learning collections of trajectories that characterize typical motion patterns of persons. Data recorded with laser-range finders is clustered using the expectation maximization algorithm. Based on the result of the clustering process we derive a Hidden Markov Model (HMM). This HMM is able to estimate the current and future positions of multiple persons given knowledge about their intentions. Experimental results obtained with a mobile robot using laser and vision data collected in a typical office building with several persons illustrate the reliability and robustness of the approach. We also demonstrate that our model provides better estimates than an HMM directly learned from the data.

1 Introduction

Whenever mobile robots are designed to operate in populated environments, they need to be able to perceive the people in their neighborhood and to adapt their behavior according to the activities of the people. Knowledge about typical motion behaviors of persons can be used in several ways to improve the behavior of a robot since it may provide better estimates about current positions of persons as well as allow better prediction of future locations.

In this paper we present an approach for learning probabilistic motion patterns of persons. We use the EM-algorithm [10] to simultaneously cluster trajectories belonging to the same motion behavior and to learn the characteristic motions of this behavior. We apply our technique to data

recorded with laser-range finders. Furthermore, we demonstrate how the learned models can be used to predict positions of persons.

Recently, a variety of service robots has been developed that have been designed to operate in populated environments. These robots for example, have been deployed in hospitals [6], museums [4], office buildings [1], and department stores [5] where they perform various services e.g., deliver, educate, entertain [14] or assist people [13, 8, 12]. A variety of techniques has been developed that allows a robot to track people in its vicinity [15, 7]. Additionally, several authors have used models of people's motions to adapt the behavior of a mobile platform according to predicted movements [18, 17, 9]. These approaches, however, assume that a motion model is given. They provide no means to learn the parameters of the motion behavior of persons. Bui et al. proposed an Abstract Hidden Markov Model (AHMM) to predict people's motion [3]. They assume that the goals and subgoals a person might have (i.e. locations the person aims to approach) are given. Our approach, in contrast, is able to learn the intentions and to automatically derive the parameters of the underlying HMM. The technique described in this paper is an extension of the approach recently proposed by Bennewitz et al. [2]. We describe how to learn the intentions of persons and how to derive an HMM from the corresponding motion patterns. This Hidden Markov Model allows the robot to maintain a belief about the current location of a person.

The paper is organized as follows. The next section introduces our approach to learn motion patterns from observed trajectories and describes how we generate Hidden Markov Models to predict motions of persons. In Section 3 we present several experiments illustrating the robustness of our approach for estimating the positions of single and multiple persons using laser and vision data with a mobile robot. We also give results indicating that our models provide better estimates than Hidden Markov Models directly learned from the observations.

2 Learning Motion Patterns

When people perform their everyday activities in their environment they do not move permanently. They usually stop at several locations and stay there for a certain period of time, depending on what activity they are currently carrying out. Accordingly, we assume that the input to our algorithm is a

*This work has partly been supported by the German Science Foundation (DFG) under contract number SFB/TR8-03 and by the EC under contract numbers IST-2000-29456 and HPMT-CT-00251.

collection of trajectories $d = \{d_1, \dots, d_N\}$ between resting places. The output is a number of different types of motion patterns $\theta = \{\theta_1, \dots, \theta_M\}$ a person might exhibit in its natural environment. Each trajectory d_i consists of a sequence $d_i = \{x_{i,1}, x_{i,2}, \dots, x_{i,T_i}\}$ of positions $x_{i,t}$. Accordingly, $x_{i,1}$ is the first position of the person when it starts leaving a resting area and x_{i,T_i} is the destination. The task of the algorithm described in this section is to cluster these trajectories into different motion behaviors and finally to derive an HMM from the resulting clusters.

Throughout this paper we assume that all trajectories have the same length T , where T is chosen as the maximum length of all trajectories. A trajectory d_i of length $T_i < T$ is extended by linear interpolation. The learning algorithm described below operates solely on this information and therefore does not take into account the velocities of the persons during the learning phase. In our experiments, we never found evidence that the walking speed of a person depends on its intention. However, one way to incorporate velocities is to introduce further dimensions to the state variables. The necessary changes to our clustering algorithm are then straightforward.

2.1 Motion Patterns

We begin with the description of our model of motion patterns, which is subsequently estimated from data using EM. Within this paper we assume that a person engages in M different types of motion patterns. A motion pattern denoted as θ_m where $1 \leq m \leq M$ is represented by K probability distributions $p(x | \theta_{m,k})$. For each $\theta_{m,k}$ the probability distribution $p(x | \theta_{m,k})$ is computed based on $\beta = \lceil T/K \rceil$ subsequent positions on the trajectories. Accordingly, $p(x | \theta_{m,k})$ specifies the probability that the person is at location x after $[(k-1) \cdot \beta + 1; k \cdot \beta]$ steps given that it is engaged in motion pattern m . Thus, we calculate the likelihood of a trajectory d_i under the m -th motion pattern θ_m as

$$p(d_i | \theta_m) = \prod_{t=1}^T p(x_{i,t} | \theta_{m, \lceil t/\beta \rceil}). \quad (1)$$

2.2 Expectation Maximization

In essence, our approach seeks to identify a model θ that maximizes the likelihood of the data. To define the likelihood of the data under the model θ , it will be useful to introduce a set of correspondence variables denoted as c_{im} . Here i is the index of the trajectory d_i , and m is the index of the motion pattern θ_m . Each correspondence c_{im} is a binary variable. It is 1 if and only if the i -th trajectory corresponds to the m -th motion pattern. If we think of a motion pattern as a specific motion activity a person might be engaged in, then c_{im} is 1 if person was engaged in motion activity m in trajectory i .

In the sequel, we will denote the set of all correspondence variables for the i -th data item by c_i , that is, $c_i = \{c_{i1}, \dots, c_{iM}\}$. For any data item i the fact that exactly one of its correspondence variable is 1 leads to $\sum_{m=1}^M c_{im} = 1$.

Throughout this paper we assume that each motion pattern is represented by K Gaussian distributions with a fixed standard deviation σ . The goal is to find the set of motion

patterns which has the highest data likelihood. EM is an algorithm that iteratively maximizes expected data likelihood by optimizing a sequence of lower bounds. In particular it generates a sequence of models denoted as $\theta^{[1]}, \theta^{[2]}, \dots$ of increasing data likelihood. The standard method is to use a so-called Q -function which depends on two models, θ and θ' . In accordance with [2] this Q -function is factored as follows:

$$Q(\theta' | \theta) = \sum_{i=1}^N \left(T \cdot M \cdot \ln \frac{1}{\sqrt{2\pi}\sigma} - \frac{1}{2\sigma^2} \cdot \sum_{t=1}^T \sum_{m=1}^M E[c_{im} | \theta, d] \|x_{i,t} - \mu'_{m, \lceil t/\beta \rceil}\|^2 \right). \quad (2)$$

The sequence of models is then given by calculating

$$\theta^{[j+1]} = \underset{\theta'}{\operatorname{argmax}} Q(\theta' | \theta^{[j]}) \quad (3)$$

starting with some initial model $\theta^{[0]}$. Whenever the Q -function is continuous as in our case, the EM algorithm converges at least to a local maximum.

In particular, the optimization involves two steps: calculating the expectations $E[c_{im} | \theta^{[j]}, d]$ given the current model $\theta^{[j]}$, and finding the new model $\theta^{[j+1]}$ that has the maximum expected data log likelihood under these expectations. The first of these two steps is typically referred to as the E-step (short for: expectation step), and the latter as the M-step (short for: maximization step).

To calculate the expectations $E[c_{im} | \theta^{[j]}, d]$ we apply Bayes' rule, obeying independence assumptions between different data trajectories:

$$\begin{aligned} E[c_{im} | \theta^{[j]}, d] &= p(c_{im} | \theta^{[j]}, d) = p(c_{im} | \theta^{[j]}, d_i) \\ &= \eta p(d_i | c_{im}, \theta^{[j]}) p(c_{im} | \theta^{[j]}) \\ &= \eta' p(d_i | \theta^{[j]}_m), \end{aligned} \quad (4)$$

where the normalization constants η and η' ensure that the expectations sum up to 1 over all m . If we combine (1) and (4) utilizing the fact that the distributions are represented by Gaussians we obtain:

$$E[c_{im} | \theta^{[j]}, d_i] = \eta' \prod_{t=1}^T e^{-\frac{1}{2\sigma^2} \|x_{i,t} - \mu^{[j]}_{m, \lceil t/\beta \rceil}\|^2}. \quad (5)$$

Finally, the M-step calculates a new model $\theta^{[j+1]}$ by maximizing the expected likelihood. Technically, this is done by computing for every motion pattern m and for each probability distribution $p(x | \theta_{m,k})$ a new mean $\mu^{[j+1]}_{m,k}$. We thereby consider the expectations $E[c_{im} | \theta^{[j]}, d]$ computed in the E-step:

$$\mu^{[j+1]}_{m,k} = \frac{1}{\beta} \cdot \sum_{t=(k-1)\cdot\beta+1}^{k\cdot\beta} \frac{\sum_{i=1}^N E[c_{im} | \theta^{[j]}, d] x_{i,t}}{\sum_{i=1}^N E[c_{im} | \theta^{[j]}, d]}. \quad (6)$$

2.3 Estimating the Number of Model Components

Since in general the correct number of motion patterns is not known in advance, we need to determine this quantity during the learning phase. If the number of motion patterns is wrong, we can distinguish two different situations. First, if there are too few motion patterns there must be trajectories, that are not explained well by any of the current motion patterns. On the other hand, if there are too many motion patterns then there must be trajectories that are explained well by different model components. Thus, whenever the EM algorithm has converged, we check whether the overall data likelihood can be improved by increasing or decreasing the number of model components. To limit the model complexity, during the evaluation we use a penalty term that depends on the number of model components (see [2]). This avoids that our algorithm learns a model that overfits the data, which in the worst case is a model with one motion pattern for every single trajectory. If the maximum number of iterations is reached or if the overall evaluation cannot be improved after increasing and decreasing the model complexity our algorithms stops and returns the model with the best value found so far. In most of the experiments carried out with different data sets our approach correctly clustered the trajectories into the corresponding categories.

2.4 Laser-based Data Acquisition

The EM-based learning procedure has been implemented for data acquired with laser-range finders. To acquire the data we used several laser-range scanners which were installed in the environment so that the relevant parts of the environment were covered. First, to identify persons in the laser data our system extracts features which are local minima in the range scans that come from the legs of persons. Additionally, it considers changes in consecutive scans to more reliably identify the moving people. To keep track of a person, we use a Kalman filter. The state x_r of a person at time step r is represented by a vector $[x, y, \delta x, \delta y]^T$. Whereas x and y represent the position of the person, the terms δx and δy represent the velocity of the person in x - and y -direction. Accordingly, the prediction is carried out by the equation:

$$x_{r+1}^- = \begin{bmatrix} 1 & 0 & t_r & 0 \\ 0 & 1 & 0 & t_r \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} x_r \quad (7)$$

where t_r is the time elapsed between the measurement z_{r+1} and z_r . Usually, sensors only give the position of an object. Since the laser range sensor does not provide the velocities δx and δy , which are also part of our state space, the measurement matrix projects onto the first two components of the state space. Accordingly, the predicted measurement at step $r + 1$ is:

$$z_{r+1}^- = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} x_{r+1}^- \quad (8)$$

In a second step we identify the resting places and perform a segmentation of the data into different slices in which the person moves. Finally, we compute the trajectories, i.e. the

sequence of positions covered by the person during that motion. When computing these trajectories, we ignore positions which lie closer than 15cm to each other.

2.5 Deriving Hidden Markov Models from Learned Intentions

Once the intentions of the persons have been learned, we can easily derive Hidden Markov Models to estimate their positions. To achieve this, we distinguish two types of nodes. The first class are the initial and final nodes that correspond to the resting places. To connect these nodes we introduce so-called intermediate nodes which lie on the learned motion patterns. In our current system we use a sequence of L_m intermediate nodes $\nu_m^1, \dots, \nu_m^{L_m}$ for each intention θ_m . The intermediate nodes are distributed over θ_m such that the distance between two consecutive nodes is $\Delta_\nu = 50\text{cm}$. Given this equidistant distribution of the sub-nodes and assuming a constant speed v with standard deviation σ_v of the person, the transition probabilities of this HMM depend on the length Δ_t of the time interval between consecutive updates of the HMM as well as on v and σ_v . In our current system, this value is set to $\Delta_t = 0.5\text{secs}$. Accordingly, we compute the probability that the person will be in node ν'_m given it is currently in ν_m and given that the time Δ_t has elapsed as:

$$p(\nu'_m | \nu_m, \Delta_t) = \int_{\nu'_m - \frac{\Delta_\nu}{2}}^{\nu'_m + \frac{\Delta_\nu}{2}} \mathcal{N}(\nu_m + v \cdot \Delta_t, \sigma_v, x) dx. \quad (9)$$

Here $\mathcal{N}(\nu_m + v \cdot \Delta_t, \sigma_v, x)$ is the value of the Gaussian with mean $\nu_m + v \cdot \Delta_t$ and standard deviation σ_v at position x . The transition probabilities for the resting places are computed based on a statistics about the average time period which elapses before the person starts this particular motion behavior after staying in the corresponding resting area.

Please note that the resulting model can be regarded as a two-level Abstract Hidden Markov Model [3]. Whereas the higher-level goals of this AHMM correspond to the resting places of the person, the lower-level goals are the nodes along the paths to the high-level goals.

2.6 An Application Example

To see how our EM-based learning procedure works in practice please consider Figure 1. In this example, a model for nine trajectories with three different intentions has to be learned. The leftmost image shows the initial model (the means of the three model components are indicated by circles). In the next two images one can see the evolution of the model components. The fourth image shows the model components after convergence of the EM algorithm. As can be seen, the trajectories are approximated quite well by the corresponding motion patterns. Finally, the rightmost picture shows the HMM derived from these motion patterns. The different resting places are indicated by rectangles and numbers.

3 Experimental Results

The technique described above has been implemented and evaluated using data acquired in an unmodified office environment. The experiments described in this section are de-

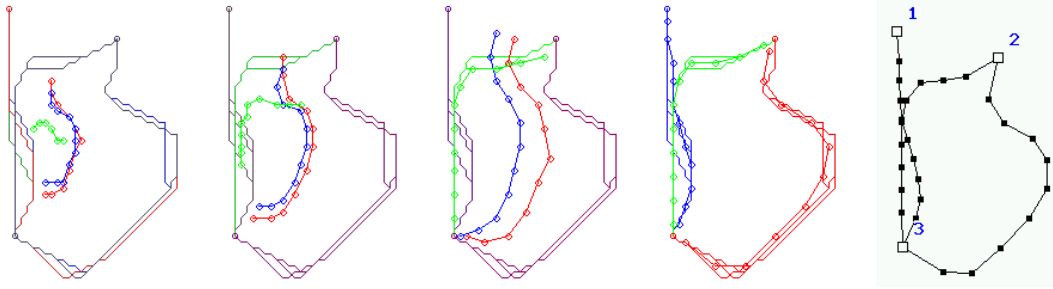


Figure 1: Trajectories of three different intentions and evolution of the motion patterns during the EM algorithm (images 1-4) and resulting HMM (rightmost image).

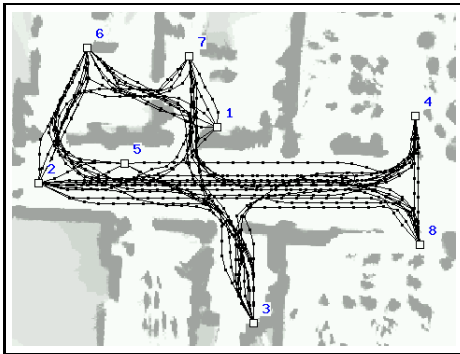


Figure 2: Hidden Markov Model derived from learned intentions.

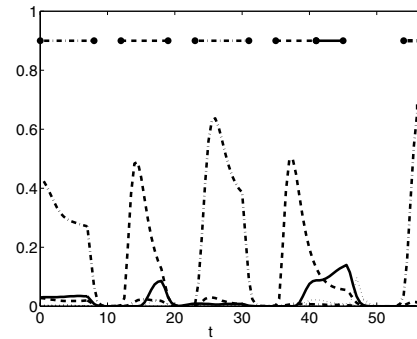


Figure 3: Evolution of the probability for a person of being in different resting areas over the time.

signed to illustrate that the approach can learn complex motion behaviors in a typical office environment. We furthermore demonstrate that the resulting models can be used to robustly estimate the positions of persons. Additionally, we compare the performance of the models learned by our algorithm to that of a standard HMM. Finally, we present an extension which allows the system to deal with multiple persons.

3.1 Learning Intentions in an Office Environment

To evaluate our approach, we applied it to data recorded over two hours in our office environment. During the acquisition phase the average speed of the person was $v=107$ cm/sec with a standard deviation $\sigma_v=25$ cm/sec. From the resulting data our system extracted 129 trajectories which were successfully clustered into 49 different intentions. The resulting Hidden Markov Model is shown in Figure 2.

3.2 Tracking a Single Person

To analyze the applicability of the learned HMM for the prediction of the locations of a person, we used our mobile robot Albert, which is a B21r platform equipped with a laser range scanner. While the robot was moving along the corridor of our department with speed up to 40cm/sec, its task was to maintain a belief about the position of the person.

To incorporate observations into the HMM we apply the

recursive Bayesian update scheme:

$$p(\nu \mid z_1, \dots, z_R) = \alpha \cdot p(z_R \mid \nu) \cdot p(\nu \mid z_1, \dots, z_{R-1}) \quad (10)$$

Thereby the likelihood $p(z_r \mid \nu)$ of an observation z_r given the state ν is computed using a Gaussian distribution which depends on both, the variance in the current estimate of the tracking system and the variance σ used during the learning of the intentions.

Figure 3 plots for different resting areas the probability that the person stays in this particular place. Whereas the x-axis represents the individual time steps, the y-axis indicates the probability. The graph also includes the ground truth, which is indicated by the corresponding horizontal line-pattern at the .9 level. As can be seen from the figure, the system can reliably determine the current position of the person. During this experiment it predicted the correct place of the person in 93% of the time.

3.3 A Comparison to Standard HMMs

The second experiment is designed to demonstrate that an HMM that takes into account the people's intentions allows a better prediction than a standard HMM that is directly generated from the observed trajectories of the persons and that does not take into account the clustered trajectories. To evaluate the performance of the two different approaches we chose two motion patterns from those depicted in Figure 2. The first pattern is the one leading from resting place 7 via

the office containing resting place 6 to the staying area 2. The second one is the motion pattern between the places 6 and 5. We defined a standard HMM over the possible states of the person in the $\langle x, y, dx, dy \rangle$ space where x and y were discretized in 15cm patches; dx and dy encode 9 possible incremental moves per cell. The transition probabilities were learned from the trajectories corresponding to both motion patterns by counting. We randomly chose a position along the trajectories of both patterns as the observed position of the person. The states of the HMM were initialized according to the observation model (see Section 3.2). After convergence of the HMM we measured the likelihood of the final destination. We compared this value to those obtained by the HMM generated by our algorithm for the trajectories corresponding to these two intentions. We repeated this experiment for different locations along the trajectories of both patterns and determined the average probability of the true goal location. Whereas we obtained an average of .74 with our model, the corresponding value of the standard HMM is .56. This illustrates that our model leads to better results and that the standard independence assumption of HMMs is generally not justified in this application domain. Please note that similar observations have been reported by Murphy [11]. In contrast to a standard HMM our model automatically chooses the transitions that correspond to the actual intention of the person.

3.4 Estimating the Locations of Multiple Persons

The final experiment described in this section is designed to illustrate that our models can also be used to maintain beliefs about multiple persons. The major difference to the situation with a single person is that we have to be able to represent the beliefs for the individual persons and to detect multiple persons in the observations. In our current system we learn an individual HMM for every person.

To track multiple persons in the range scans, we apply independent Kalman filters, one for each feature. To solve the data association problem, we apply a nearest neighbor approach, i.e. we update a filter using the observation z_{r+1} that is closest to z_{r+1}^- . New filters are introduced for observations from which all predictions are too far away. Furthermore, filters are removed if no corresponding feature can be found for one second.

We also need to be able to identify a person in order to appropriately update the belief about the location of that person. To achieve this we additionally employ the vision system of our robot. To identify a person, we proceed as follows: Every time the laser-based people tracker detects a person in the field of view of the camera, an image is collected and following three steps are applied:

1. *Segmentation*: The size of a rectangular area of the image containing the person is determined.
2. *Feature extraction*: We compute a color histogram for the area selected in the previous step.
3. *Database matching*: To determine the likelihood of a particular person, we compare the histogram computed in step 2 to all prototypes existing in the database.

To determine the area in the image corresponding to a feature detected by the laser tracking system, we rely on an accu-

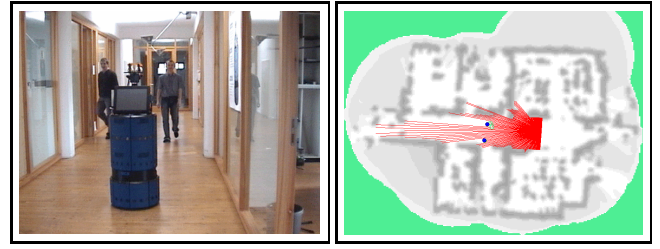


Figure 4: Typical scene with two persons walking along the corridor (left image) and corresponding estimate of the laser-based people tracking system (right image).

rate calibration between the camera and the laser and we use a perspective projection to map the 3D position of the person in world coordinates to 2D image coordinates. Whereas color histograms are robust with respect to translation, rotation, scale and to any kind of geometric distortions they are sensitive to varying lighting conditions. To handle this problem we consider the HSV (Hue-Saturation-Value) color space. In this color model the intensity factor can be separated so that its influence is reduced. In our current system we simply ignore this factor. Throughout all our experiments we could not find any evidence that this factor negatively affected the performance of the system. The image database is created beforehand. For each person it contains one histogram which is built from 20 images.

To compare a given query histogram I with a prototype M in the database we use normalized intersection norm $H(I, M)$ [16]. This quantity can be computed as:

$$H(I, M) = \frac{\sum_{j=1}^n \min(I_j, M_j)}{\sum_{j=1}^n M_j}, \quad (11)$$

where I and M are color histograms both having n bins. One advantage of this norm is that it also allows to compare partial views, i.e. when the person is close to the camera and only a part of it is visible.

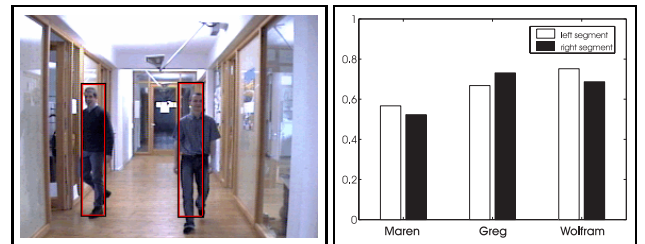


Figure 5: Segmentation of the two persons from the image grabbed with the camera of the robot (left image) and similarity of these segments to the data base prototypes (right image).

To incorporate the similarity measure provided by the vision system into the HMM of the person π , we simply multiply the likelihoods provided by the laser tracking system with the similarity measure $H(I_s, M_\pi)$ of the query histogram I_s for the segment s and the data base prototype M_π for person

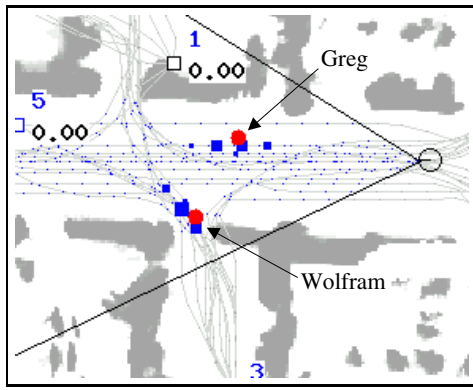


Figure 6: Resulting posterior after incorporating the two segments shown in Figure 5 into the belief over Wolfram's position.

π . If the current estimate of the laser tracker is not in the field of view of the camera we simply update the HMM for all persons as we do in the case in which we track a single person only.

As an application example consider the situation depicted in the left image of Figure 4. In this particular situation two persons (Wolfram and Greg) are walking along the corridor within the perceptual field of the robot. The right image of Figure 4 shows the estimate of the laser-based people tracking system at the same point in time. The corresponding image obtained with the robot's camera is shown in the left image of Figure 5. Also shown there are the two segments of the image that correspond to the two persons detected with the laser. The right image of this figure plots the similarities of the two segments to the individual prototypes stored in the data base. Finally, Figure 6 depicts the HMM for Wolfram (who is the left person in Figure 5). As can be seen, the probabilities indicated by the size of the rectangles are slightly higher for the states that correspond to Wolfram's true location. Throughout this experiment the robot was able to predict the correct location of the persons in 79% of all cases.

4 Conclusions

In this paper we have presented a method for learning and utilizing motion behaviors of persons. Our approach applies the EM-algorithm to cluster trajectories recorded with laser range sensors into a collection of motion patterns, each corresponding to a possible intention of a person. From these motion patterns we automatically derive an HMM that can be used to predict the positions of persons in their environment.

Our approach has been implemented and applied successfully to trajectories recorded in a typical office environment. Practical experiments demonstrate that our method is able to learn typical motion behaviors of persons and to reliably use them for state estimation. The experiments have been carried out using a mobile robot equipped with a laser-range sensor and a vision system. We have furthermore presented experiments indicating that standard HMMs directly learned from the same input data are less predictive than our models.

References

- [1] H. Asoh, S. Hayamizu, I. Hara, Y. Motomura, S. Akaho, and T. Matsui. Socially embedded learning of office-conversant robot Jijo-2. In *Proc. of the Int. Joint Conference on Artificial Intelligence (IJCAI)*, 1997.
- [2] M. Bennewitz, W. Burgard, and S. Thrun. Using EM to learn motion behaviors of persons with mobile robots. In *Proc. of the Int. Conference on Intelligent Robots and Systems (IROS)*, 2002.
- [3] H. Bui, S. Venkatesh, and G. West. Tracking and surveillance in wide-area spatial environments using the Abstract Hidden Markov Model. *Intl. J. of Pattern Rec. and AI*, 2001.
- [4] W. Burgard, A.B. Cremers, D. Fox, D. Hähnel, G. Lakemeyer, D. Schulz, W. Steiner, and S. Thrun. Experiences with an interactive museum tour-guide robot. *Artificial Intelligence*, 114(1-2), 1999.
- [5] H. Endres, W. Feiten, and G. Lawitzky. Field test of a navigation system: Autonomous cleaning in supermarkets. In *Proc. of the Int. Conference on Robotics & Automation (ICRA)*, 1998.
- [6] S. King and C. Weiman. Helpmate autonomous mobile robot navigation system. In *Proc. of the SPIE Conference on Mobile Robots*, pages 190–198, Boston, MA, November 1990. Volume 2352.
- [7] B. Kluge, C. Köhler, and E. Prassler. Fast and robust tracking of multiple moving objects with a laser range finder. In *Proc. of the IEEE Int. Conference on Robotics & Automation (ICRA)*, 2001.
- [8] G. Lacey and K. Dawson-Howe. The application of robotics to a mobility aid for the elderly blind. *Journal of Robotics and Autonomous Systems (RAS)*, 23:245–252, 1998.
- [9] S. M. Lavalle, H. H. Gonzalez-Banos, G. Becker, and J.-C. Latombe. Motion strategies for maintaining visibility of a moving target. In *Proc. of the IEEE Int. Conference on Robotics & Automation (ICRA)*, 1997.
- [10] G.J. McLachlan and T. Krishnan. *The EM Algorithm and Extensions*. Wiley Series in Probability and Statistics, 1997.
- [11] K. P. Murphy. *Dynamic Bayesian Networks: Representation, Inference, and Learning*. PhD thesis, University of California, Berkeley, 2002.
- [12] N. Roy, G. Baltus, D. Fox, F. Gemperle, J. Goetz, T. Hirsch, D. Magaritis, M. Montemerlo, J. Pineau, Schulte J., and S. Thrun. Towards personal service robots for the elderly. In *Proc. of the Workshop on Interactive Robotics and Entertainment*, 2000.
- [13] C. Schaeffer and T. May. Care-o-bot - a system for assisting elderly or disabled persons in home environments. In *Assistive technology on the threshold of the new millenium*. IOS Press, Amsterdam, 1999.
- [14] R.D. Schraft and G. Schmierer. *Service Robots*. Springer Verlag, 1998.
- [15] D. Schulz, W. Burgard, D. Fox, and A.B. Cremers. Tracking multiple moving targets with a mobile robot using particle filters and statistical data association. In *Proc. of the Int. Conference on Robotics & Automation (ICRA)*, 2001.
- [16] M. Swain and D. Ballard. Color indexing. *International Journal of Computer Vision*, 7(1), 1991.
- [17] S. Tadokoro, M. Hayashi, Y. Manabe, Y. Nakami, and T. Takamori. On motion planning of mobile robots which co-exist and cooperate with human. In *Proc. of the Int. Conference on Intelligent Robots and Systems (IROS)*, 1995.
- [18] Q. Zhu. Hidden Markov model for dynamic obstacle avoidance of mobile robot navigation. *IEEE Transactions on Robotics and Automation*, 7(3), 1991.

An Extension of the ICP Algorithm for Modeling Nonrigid Objects with Mobile Robots

Dirk Hähnel^{†‡}, Sebastian Thrun[†], Wolfram Burgard^{†‡}

[†]Carnegie Mellon University, School of Computer Science, PA, USA,

[‡]University of Freiburg, Department of Computer Science, Germany

Abstract

The iterative closest point (ICP) algorithm [2] is a popular method for modeling 3D objects from range data. The classical ICP algorithm rests on a rigid surface assumption. Building on recent work on nonrigid object models [5; 16; 9], this paper presents an ICP algorithm capable of modeling nonrigid objects, where individual scans may be subject to local deformations. We describe an integrated mathematical framework for simultaneously registering scans and recovering the surface configuration. To tackle the resulting high-dimensional optimization problems, we introduce a hierarchical method that first matches a coarse skeleton of scan points, then adapts local scan patches. The approach is implemented for a mobile robot capable of acquiring 3D models of objects.

1 Introduction

In recent years, there has been a flurry of work on acquiring 3D models from range data. The classical setting involves a range sensor (e.g., a 3D range camera or a stereo vision system) used to acquire range images of the target object from multiple vantage points. The problem of integrating multiple range scans into a 3D model is commonly known as *scan registration*. Most state-of-the-art implementations are based on the popular *iterative closest point* algorithm [2]. The topic has received significant attention in fields as diverse as computer vision [12; 9] and medical imaging [7], large-scale urban modeling [14], and mobile robotics [10; 8; 15].

ICP aligns range scans by alternating a step in which closest points are identified, and a step by which the optimal translation and rotation of scans relative to each other is computed. In doing so, most of them are making a rigid object assumption: range scans, if aligned correctly, must be spatially consistent with each other.

Many objects are deformable. For example, people change shape, as do trees, pillows, and so on. A natural research goal is therefore to extend ICP to accommodate local object transformations. Following recent work primarily found in the medical imaging literature [5; 16; 6; 3], we propose an approach suited for scan registration and 3D modeling of non-rigid objects that can efficiently deal with hundreds of thou-

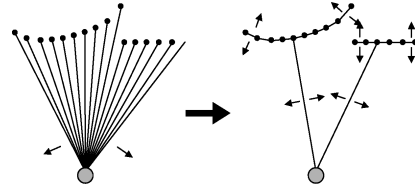


Figure 1: The essential idea: Rather assuming that the relation of measurement coordinates are fixed relative to the location from which the measurement was taken, the approach proposed here constrains the relation of measurement coordinates in a soft way. The exact configuration of a scan is calculated while scans are registered to each other.

sands of variables. To accommodate local deformations, our approach transforms scans into Markov random fields, where nearby measurements are linked by a (nonlinear) potential function. All links are soft. They can be bent, but bending them incurs a penalty. Figure 1 illustrates this transformation: rigid links between the measurement coordinates and the robot sensor are replaced by soft links between adjacent measurement points. The resulting problem of scan registration under these soft constraints becomes a high-dimensional optimization problem with orders of magnitude more variables involved than in regular ICP. We show how to solve this problem via Taylor series expansion (linearization), and then propose a coarse-to-fine hierarchical optimization technique for carrying out the optimization efficiently.

Our new algorithm is applied to the problem of learning 3D models of non-stationary objects with a mobile robot. We describe an implemented robot system that utilizes a model differencing technique similar to the one described in [1] to segment scans. By acquiring views of the target objects from multiple sides, our approach enables a robot to acquire a 3D model of a non-stationary object.

2 Scan Registration

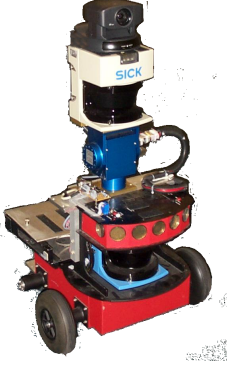
This section describes a variant of the well-known *iterative closest point* algorithm (ICP) [2] for *rigid* objects. The algorithm alternates two phases, one in which nearest points is identified, and one in which the distance between all pairs of nearest points is minimized.

2.1 Scans

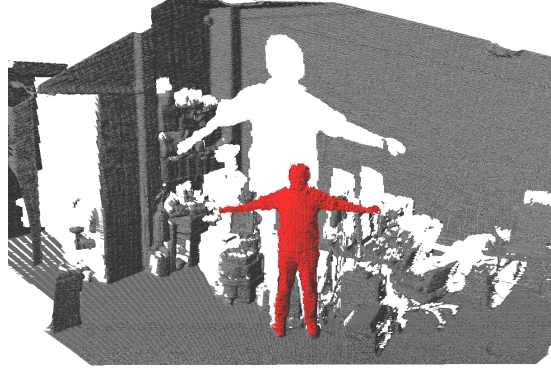
The input to the ICP algorithm is a set of 3D scans denoted

$$\mathbf{d} = \mathbf{d}_1, \mathbf{d}_2, \dots \quad (1)$$

(a) mobile robot with scanner



(b) scene acquired by scanner



(c) result of model differencing



Figure 2: (a) 3D range scan acquired by mobile robot shown in (b). (c) Scene from which the scan in (a) is extracted through background differencing.

Each such scan $\mathbf{d}_k$ consists of a collection of 1D range measurements, arranged as a 3D “range image”:

$$\mathbf{d}_k = d_{1k}, d_{2k}, \dots \quad (2)$$

Figure 2c shows a typical scan acquired by a robot. Further below, we will denote the horizontal angle of the k -th range measurement by α_k and the vertical one by β_k . In our experimental setup, scans are obtained by a SICK laser range finder mounted on a tilt unit shown in Figure 2a.

The problem of scan registration can be formulated as the problem of recovering the vantage points from which the scans were taken. In our approach, scans are taken from a mobile robot; hence each vantage point is described by three variables: its x - y location in Cartesian coordinates and its orientation γ :

$$\mathbf{x}_k = (x_k, y_k, \gamma_k)^T \quad (3)$$

Here $\mathbf{x}_k$ denotes the vantage point from which scan $\mathbf{d}_k$ was acquired. The set of all vantage points will be denoted $\mathbf{x}$. Recovering the vantage points $\mathbf{x}$ is equivalent to registering the scans if one vantage point is (arbitrarily) defined to be $\mathbf{x}_1 = (0, 0, 0)^T$.

2.2 Measurement Model

Scans are registered in world coordinates. To do so, measurements d_{ik} must be mapped into 3D world coordinates. This is achieved by a projective function π , which takes as an argument a range measurement and a vantage point and returns as its output the corresponding coordinate in 3D world coordinates:

$$\pi(d_{ik}, \mathbf{x}_k) = \begin{pmatrix} x_k + d_{ik} \cos(\gamma_k + \alpha_i) \sin \beta_i \\ y_k + d_{ik} \sin(\gamma_k + \alpha_i) \sin \beta_i \\ z + d_{ik} \cos \beta_i \end{pmatrix} \quad (4)$$

Here α_i and β_i are the orientation of the range measurement d_{ik} relative to the sensor. The variable z is the generic height of the sensor which in our robot system is fixed, but is easily generalized to variable heights. For brevity, we will sometimes write π_{ik} instead of $\pi(d_{ik}, \mathbf{x}_k)$.

We define the quality of a pairwise scan registration by the probability representing the likelihood $p(d_{ik})$ of a range measurement under a fixed (hypothetical) registration:

$$p(d_{ik} | \mathbf{d}_l, \mathbf{x}_k, \mathbf{x}_l) = \begin{cases} \max_j |2\pi\Sigma|^{-\frac{1}{2}} \exp \left\{ -\frac{1}{2} (\pi_{ik} - \pi_{jl})^T \Sigma^{-1} (\pi_{ik} - \pi_{jl}) \right\} & \text{if } \pi_{ik} \in \mathcal{F}_l \\ \text{const.} & \text{if } \pi_{ik} \notin \mathcal{F}_l \end{cases} \quad (5)$$

This likelihood distinguishes two cases. The first case models the noise in range perception by a Gaussian with a sensor-specific covariance Σ (usually a diagonal matrix). The measurement error under this Gaussian is given by the distance between the point π_{ik} under consideration, and the point j in scan $\mathbf{d}_l$ that maximizes this Gaussian. As is easily to be seen, this point π_{jl} minimizes the Mahalanobis distance to π_{ik} . Thus, π_{jl} is simply the point “nearest” to π_{ik} in the scan $\mathbf{d}_l$ under Σ^{-1} .

However, finding the nearest point only makes sense if π_{ik} falls within the perceptual range of scan $\mathbf{d}_l$. If π_{ik} is occluded, it might be perfectly well-explained by an object not detectable from the vantage point $\mathbf{x}_l$. This is captured by the second case in (5), which applies when π_{ik} lies outside the free space of scan l . The free space of scan l is denoted $\mathcal{F}_l$. It is defined as the region between the robot and the detected objects. If π_{ik} lies outside this region, the measurement probability is assumed to be uniform. The value of this uniform depends on the range of occluded space, but it plays no role in the optimization to come; hence we leave it unspecified.

2.3 Registration as Likelihood Maximization

The goal of scan registration is to determine the vantage points $\mathbf{x}$ that maximize the joint likelihood of the scans $\mathbf{d}$ and $\mathbf{x}$:

$$p(\mathbf{d}, \mathbf{x}) = p(\mathbf{d} | \mathbf{x}) p(\mathbf{x}) \quad (6)$$

The first term of this product is obtained by calculating the product over all individual measurement likelihoods, assuming noise independence in each measurement:

$$p(\mathbf{d} | \mathbf{x}) = \prod_k \prod_i \prod_{l \neq k} p(d_{ik} | \mathbf{d}_l, \mathbf{x}_k, \mathbf{x}_l) \quad (7)$$

The term $p(\mathbf{x})$ in (6) is the *prior* on the individual vantage points (and hence on the registration). The prior of the point $\mathbf{x}_k$ is expressed as a Gaussian with mean $\bar{\mathbf{x}}_k$ and covariance matrix Ψ :

$$\begin{aligned} p(\mathbf{x}) &= \prod_k p(\mathbf{x}_k) \\ &= \prod_k |2\pi\Psi|^{-\frac{1}{2}} \exp \left\{ -\frac{1}{2} (\mathbf{x}_k - \bar{\mathbf{x}}_k)^T \Psi^{-1} (\mathbf{x}_k - \bar{\mathbf{x}}_k) \right\} \end{aligned} \quad (8)$$

In our robot system, this prior is obtained from the robot’s 2D localization routines, supplied by a public domain software package [11].

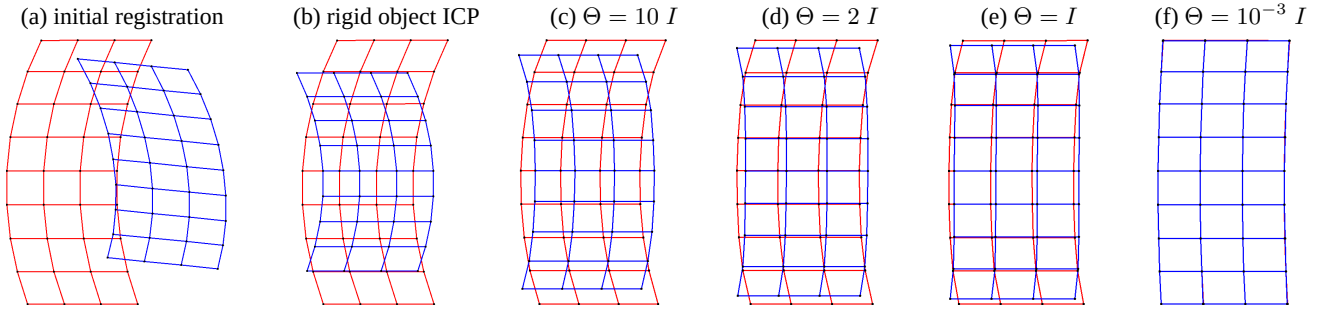


Figure 3: Illustration in 2D: (a) initial configurations of two scans in red and blue, (b) result of optimal registration, (c-f) result of recovering the object configuration by our new adjustable shape technique.

The negative logarithm of the joint likelihood (6) is given by the following sum:

$$\begin{aligned} G &= -\log p(\mathbf{d}, \mathbf{x}) \\ &= \text{const.} + \frac{1}{2} \sum_k [(\mathbf{x}_k - \bar{\mathbf{x}}_k)^T \Psi^{-1} (\mathbf{x}_k - \bar{\mathbf{x}}_k) \\ &\quad + \sum_i \sum_{l \neq k} \min_j (\pi_{ik} - \pi_{jl})^T \Sigma^{-1} (\pi_{ik} - \pi_{jl})] \end{aligned} \quad (9)$$

Scans are registered by minimizing the sum, which is equivalent to maximizing the likelihood function.

Unfortunately, minimizing G is not possible *in closed form* for the following three reasons: (1) the exact terms in the sum depend on the occlusion constraint which is a function of the vantage points $\mathbf{x}$; (2) the result of the minimization over j involves a discrete search for a nearest neighbor, whose outcome is again a function of $\mathbf{x}$; (3) the projection functions π are nonlinear in the points $\mathbf{x}$, and the resulting nonlinear optimization problem is hard.

2.4 Optimization Procedure

ICP-style algorithms minimize the negative log-likelihood by calculating a sequence of vantage points (registrations):

$$\mathbf{x}_k^{[0]}, \mathbf{x}_k^{[1]}, \mathbf{x}_k^{[2]}, \dots \quad (10)$$

The first set of points are obtained from the prior (e.g., the robot odometry): $\mathbf{x}_k^{[0]} = \bar{\mathbf{x}}_k$. The $(n+1)$ -th vantage points are computed from the n -th ones by the following five step algorithm:

Step 1. The set of occluded points is determined for the n -th points $\mathbf{x}_k^{[n]}$. This step involves the calculation of a relative orientation of a point π_{ik} to scan $\mathbf{d}_l$ via $\mathbf{x}_l$. It furthermore involves a range comparison with the corresponding measurement in scan $\mathbf{d}_l$, to determine if π_{ik} is occluded relative to $\mathbf{d}_l$.

Step 2. The minimization in (9) is carried out by determining the closest measurement j in each scan relative to each point π_{ik} :

$$l_{ikl}^{[n]} = \min_j (\pi_{ik} - \pi_{jl})^T \Sigma^{-1} (\pi_{ik} - \pi_{jl}) \quad (11)$$

For $\Sigma = \text{const.} \cdot I$, this calculation is equivalent to finding the closest points (hence the name of the algorithm).

Step 3. The projection function π is linearized to obtain a quadratic objective function. This is achieved by the Taylor series expansion:

$$\pi(d_{ik}, \mathbf{x}_k) \approx \pi_{ik}^{[n]} + \mathbf{J}_{ik}^{[n]} (\mathbf{x}_k - \mathbf{x}_k^{[n]}) \quad (12)$$

Here $\mathbf{J}_{ik}^{[n]}$ is the Jacobian (gradient) of π at $\mathbf{x}_k^{[n]}$, which is directly obtained from the definition of π in (4):

$$\mathbf{J}_{ik}^{[n]} = \begin{pmatrix} 1 & 0 & -d_{ik} \sin(\gamma_k + \alpha_i) \sin \beta_i \\ 0 & 1 & d_{ik} \cos(\gamma_k + \alpha_i) \sin \beta_i \\ 0 & 0 & 0 \end{pmatrix} \quad (13)$$

With this approximation, $G^{[n]}$ (the negative log-likelihood function G in the n -th iteration) becomes quadratic in $\mathbf{x}_k$ and is of the form

$$\begin{aligned} G^{[n]} &= \text{const.} + \frac{1}{2} (\mathbf{x} - \bar{\mathbf{x}})^T \Psi^{-1} (\mathbf{x} - \bar{\mathbf{x}}) \\ &\quad + \frac{1}{2} (\mathbf{A}^{[n]} \mathbf{x} - \mathbf{c}^{[n]})^T \tilde{\Gamma}^{-1} (\mathbf{A}^{[n]} \mathbf{x} - \mathbf{c}^{[n]}) \end{aligned} \quad (14)$$

Here $\mathbf{A}^{[n]}$ is a matrix, $\mathbf{c}^{[n]}$ a vector, and $\mathbf{x}$ the vector of all vantage points (the unknowns). The various sums in (9) are simply integrated into $\mathbf{A}^{[n]}$ and $\mathbf{c}^{[n]}$, and $\tilde{\Sigma}$ generalizes Σ to the full space of all points $\mathbf{x}$:

$$\tilde{\Sigma}^{-1} = \begin{pmatrix} \Sigma^{-1} & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \Sigma^{-1} \end{pmatrix}^{-1} \quad (15)$$

Step 4. The x - y -values of the vantage points $\mathbf{x}^{[n+1]}$ are now obtained through a closed-form solution of minimizing $G^{[n]}$. This solution is obtained by setting the first derivative of $G^{[n]}$ with respect to $\mathbf{x}$ to zero (derivation omitted, see [13]):

$$\begin{aligned} \mathbf{x}^{[n+1]} &= (\Psi^{-1} + \mathbf{A}^{[n]T} \Sigma^{-1} \mathbf{A}^{[n]})^{-1} (\Psi^{-1} \bar{\mathbf{x}} + \mathbf{A}^{[n]T} \Sigma^{-1} \mathbf{c}^{[n]}) \end{aligned} \quad (16)$$

In the original ICP algorithm [2], this step is implemented by calculating the “center of mass” of each scan and shifting the scans accordingly (which is computationally simpler); our approach also accommodates a prior $\bar{\mathbf{x}}$ on $\mathbf{x}$.

Step 5. Because of the linearization, only Cartesian coordinates are actually updated in Step 4; the orientation coordinates remain unmodified. For this reason, the original ICP literature introduced a separate step for calculating orientations. This is achieved by a singular value decomposition (SVD) step, in which the correlation between residual errors in the scan is determined, and each scan is rotated so as to minimize these error correlations. Details on this step can be found in the ICP literature [2].

All steps are iterated until a convergence criterion is reached. Satisfactory registrations are usually obtained within the first three to four iterations. If the total number of scans is small (e.g., less than 10), the computationally most expensive

step is the determination of the closest points in Step 2. This step is usually implemented efficiently by representing scans through kd-trees [4].

Figure 3a-b illustrate the result of scan registration in 2D. The initial configuration in Figure 3a is transformed into the one shown in Figure 3b, which is the one that minimizes the squared distance (maximizes the likelihood). Clearly, both scans are incompatible in shape. Pure registration techniques are unable to handle such shape deformations, but the technique presented in the next section is.

3 Recovering the Surface Configuration of Nonrigid Objects

The key idea for extending ICP to nonrigid objects was already discussed in the introduction to this paper, and is highlighted in Figure 1. Technically, it involves two modifications: First, the static relationship between points π_{ik} and the corresponding vantage points $\mathbf{x}_k$ is replaced by nonrigid links between adjacent points. These links can be bent (but at a probabilistic penalty), to accommodate nonrigid surfaces. Second, and as consequence of this, the optimization now involves the determination of the location of all points π_{ik} , in addition to the robot poses $\mathbf{x}_k$. This optimization problem is much higher dimensional, and we will discuss a hierarchical optimization technique for tackling it efficiently. A key characteristic of the approach proposed here is that it fits neatly into the ICP methodology above: Again, under appropriate linearization the target function is quadratic, and estimates are obtained just as in (16).

3.1 Links

The definition of links between pairs of adjacent points makes it necessary to augment the notion of a measurement point. In particular, our approach associates an (imaginary) coordinate system with each node. The origin of each coordinate system is the familiar coordinate π_{ik} , and its orientation is specified by three Euler angles (an alternative formulation may use quaternions):

$$\mathbf{r}_{ik} = (\phi_{ik}, \theta_{ik}, \psi_{ik})^T \quad (17)$$

The orientation is initialized arbitrarily; e.g. $\mathbf{r}_{ik} = (0, 0, 0)^T$. (The result of the optimization is invariant with respect to this initialization.) A link is now given by the affine coordinate transformations among the coordinate systems of adjacent measurements. Each link possesses six parameters, three for rotation (denoted $\Delta\mathbf{r}_{i \rightarrow j, k}$) and three for translation (denoted $\Delta\pi_{i \rightarrow j, k}$). They are calculated as follows:

$$\begin{aligned} \Delta\mathbf{r}_{i \rightarrow j, k} &= \mathbf{r}_{jk} - \mathbf{r}_{ik} \\ \Delta\pi_{i \rightarrow j, k} &= R_z(-\psi_{ik}) \cdot R_y(-\theta_{ik}) \cdot R_x(-\phi_{ik}) \begin{pmatrix} x_{jk} - x_{ik} \\ y_{jk} - y_{ik} \\ z_{jk} - z_{ik} \end{pmatrix} \end{aligned} \quad (18)$$

Here the R 's are the rotation matrices around the three coordinate axes. Links enable us to recover a node's coordinates from any of its neighbors:

$$\begin{aligned} \pi_{jk} &= \underbrace{\pi_{ik} + R_x(\phi_{ik}) \cdot R_y(\theta_{ik}) \cdot R_z(\psi_{ik}) \Delta\pi_{i \rightarrow j, k}}_{=: \hat{\pi}_{i \rightarrow j, k}} \\ \mathbf{r}_{jk} &= \underbrace{\mathbf{r}_{ik} + \Delta\mathbf{r}_{i \rightarrow j, k}}_{=: \hat{\mathbf{r}}_{i \rightarrow j, k}} \end{aligned} \quad (19)$$

To model nonrigid surfaces, our approach allows for violation of these link constraints. This is obtained by introducing the following Gaussian potentials for each link

$$h_{i \rightarrow j, k} =; |2\pi\Theta|^{-\frac{1}{2}} \exp \left\{ -\frac{1}{2} \begin{pmatrix} \pi_{jk} - \hat{\pi}_{i \rightarrow j, k} \\ \mathbf{r}_{jk} - \hat{\mathbf{r}}_{i \rightarrow j, k} \end{pmatrix}^T \Theta^{-1} \begin{pmatrix} \pi_{jk} - \hat{\pi}_{i \rightarrow j, k} \\ \mathbf{r}_{jk} - \hat{\mathbf{r}}_{i \rightarrow j, k} \end{pmatrix} \right\}$$

Here Θ defines the strength of the link (the resulting structure is a Markov random field [17]).

3.2 Target Function

The negative logarithm of these potentials, summed over all links, is given by the following function H (constant omitted):

$$H = \frac{1}{2} \sum_{i \rightarrow j, k} \begin{pmatrix} \pi_{jk} - \hat{\pi}_{i \rightarrow j, k} \\ \mathbf{r}_{jk} - \hat{\mathbf{r}}_{i \rightarrow j, k} \end{pmatrix}^T \Theta^{-1} \begin{pmatrix} \pi_{jk} - \hat{\pi}_{i \rightarrow j, k} \\ \mathbf{r}_{jk} - \hat{\mathbf{r}}_{i \rightarrow j, k} \end{pmatrix} \quad (20)$$

As in the scan registration problem, all these terms are non-linear in the node coordinates π_{ik} and the orientations $\mathbf{r}_{ik}$. To obtain a closed-form solution for the resulting equation system, the link function is linearized via a Taylor series expansion:

$$\begin{pmatrix} \pi_{jk} \\ \mathbf{r}_{jk} \end{pmatrix} \approx \begin{pmatrix} \pi_{jk}^{[n]} \\ \mathbf{r}_{jk}^{[n]} \end{pmatrix} + K_{ik, ik}^{[n]} \begin{pmatrix} \pi_{ik} - \pi_{ik}^{[n]} \\ \mathbf{r}_{ik} - \mathbf{r}_{ik}^{[n]} \end{pmatrix} \quad (21)$$

where $\pi_{ik}^{[n]}$ and $\mathbf{r}_{ik}^{[n]}$ specify the coordinate system for the node ik in the n -th iteration of the optimization. The matrix $K_{ik, ik}^{[n]}$ is a Jacobian matrix of dimension six by six, which is obtained as the derivative of the functions (19).

By the same logic by which G can be approximated by a quadratic function in the scan registration problem, substituting our approximation back into the definition of H gives us a quadratic function in all variables π_{ik} and $\mathbf{r}_{ik}$. This function can be written in the form

$$H = \frac{1}{2} \left[\mathbf{B}^{[n]} \begin{pmatrix} \pi \\ \mathbf{r} \end{pmatrix} - \mathbf{f}^{[n]} \right]^T \tilde{\Theta}^{-1} \left[\mathbf{B}^{[n]} \begin{pmatrix} \pi \\ \mathbf{r} \end{pmatrix} - \mathbf{f}^{[n]} \right] \quad (22)$$

where π is the vector of all coordinate system origins, $\mathbf{r}$ the vector of all Euler angles, and $\mathbf{B}^{[n]}$ is a matrix and $\mathbf{f}^{[n]}$ a vector. Calculating $\mathbf{B}^{[n]}$ and $\mathbf{f}^{[n]}$ is involved but mathematically straightforward.

3.3 Optimization Procedure

Our new version of ICP now minimizes the combined target function $G + H$, which is again quadratic in all parameters ($\mathbf{x}$, π , and $\mathbf{r}$). By doing so, it simultaneously recovers the scan registration and the surface configuration of the object. The solutions for $\mathbf{x}$ and π are completely analogous to the one in (16):

$$\begin{pmatrix} \pi^{[n+1]} \\ \mathbf{r}^{[n+1]} \end{pmatrix} = (\mathbf{B}^{[n]T} \mathbf{B}^{[n]})^{-1} \mathbf{B}^{[n]T} \mathbf{f}^{[n]} \quad (23)$$

The global orientation is still optimized by a single global SVD as above. Our new augmented optimization leads to relative adjustments between measurement points, in which the links play the role of soft constraints.

This is illustrated in Figure 3c-f, for different values for Θ . As the various diagrams illustrate, scans are deformed to improve their match. The degree of the deformation depends on the value of Θ , which defines the rigidity of the surface. Figure 3c-f illustrates that our approach succeeds in locally rotating and even rescaling the model.

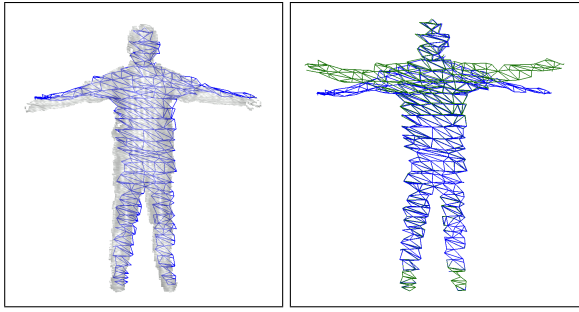


Figure 4: Example of a thinned graph superimposed to the original scan (left) and before and after adjustment (right). Thinning is necessary to perform the optimization efficiently.

3.4 Efficient Variable Resolution Optimization

The main problem with the approach so far is its enormous complexity. The number of variables involved in the optimization is orders of magnitude larger than in scan registration. This is because the target function H is a function of all measurement points π and orientations $\mathbf{r}$, whereas G has only the vantage points $\mathbf{x}$ as its arguments. The matrix $\mathbf{B}^{[n]}$ in (23) is, thus, a (sparse) matrix with many thousand dimensions.

To tackle such problems efficiently the optimization is reduced to a sequence of nested optimization problems. In a first step, scans are analyzed for connected components (regions without large disparities); links exist only between connected components in each scan; hence H factors naturally into different subproblems for different connected components. Next, the resulting scan patches are thinned. Thinning proceeds by identifying a small number of representative landmark measurements that are approximately equally spaced. This computation is performed by stipulating a grid over the scan (in workspace coordinates), and selecting measurements closest to the center points of each grid cell. An all point shortest path search then associates remaining measurements with landmark measurements. The optimization is first performed for the thinned scan; after the landmark scans are localized (and the corresponding coordinate transformation are computed), the remaining measurements are optimized locally, in groups corresponding to individual landmark measurements. Smoothness is attained by using multiple landmark measurements as boundary conditions in this optimization. Figure 4 shows an examples of a thinned graph, for which the optimization can be carried out in seconds.

4 Setup and Experimental Results

Our approach was implemented using a mobile robot, in an attempt to acquire 3D models of non-stationary objects. In a

	start	loop #1	loop #2	loop #3
<i>moving arms</i>				
scan 1	2.3266	0.8993	0.7986	-
scan 2	2.5320	0.8704	0.8001	-
<i>stretched body</i>				
scan 1	1.9369	1.2915	1.2008	1.1975
scan 2	2.5087	1.2964	1.2220	1.2120

Table 1: Average distance to the closest points to the matched model *after* scan registration. The decrease of this distance measures the improvement of the model through local surface deformations.

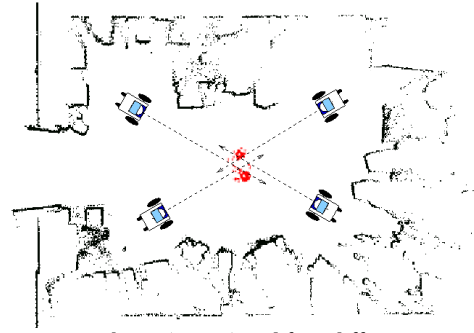


Figure 5: 2D map, object (center) and four different vantage points.

technique adopted from [11], we first acquired a map of the environment (see Figure 5). Non-stationary objects were detected through differencing of scans, using the robot’s localization routines to get a rough estimate of this pose. Figure 2b illustrates the segmentation process. Red scans are retained while the black scans are assumed to correspond to the background and are henceforth discarded.

Figure 6 illustrates one iteration of the algorithm in all essential steps, using data acquired by the robot. Results for matching three scans with different postures are shown in Figure 7(a-b). While the standard registration procedure leads to a model with six arms, our approach correctly deforms the scan to arrive at an improved model, with two arms. A similar result is shown in Figure 7(c-g), which shows three raw scans on the left, followed by the result of (rigid) scan registration and the result of our approach. Another example is the chair in Figure 8(a-d) scanned in different heights. The standard registration will lead to multiple feet, our approach correctly aligns them. Table 1 shows the cumulative distance between points in the nearest neighbor calculation. The value marked as “start” is the result of an initial registration phase, reflecting the remaining distances under the rigid body assumption. All other columns correspond to further iterations of our algorithm as it adjusts the shape of the scans. This result illustrates numerically the integrity of the result is indeed improved by iterate the process.

5 Discussion

This paper proposed a technique for simultaneous scan registration and scan deformation for modeling nonrigid objects. The deformation was made possible through the definition of (soft) links between neighboring scan points, whose configuration was calculated during registration. To tackle the resulting optimization problem efficiently, we described a hierarchical optimization techniques that operated on thinned graphs. Experimental results obtained using a mobile robot illustrated the viability of this approach.

There are many problems in object modeling that this paper does not address, but whose inclusion shall be the subject of future research. For example, the present segmentation approach is somewhat simplistic: It will fail if more than one non-stationary object appears in the scene. The approach requires deformations to be small, and the target object may not move very far during acquisition. Objects are not sub-segmented. This will cause difficulties when components of objects are adjacent to different other components, or missing entirely, which can happen if components are combined and

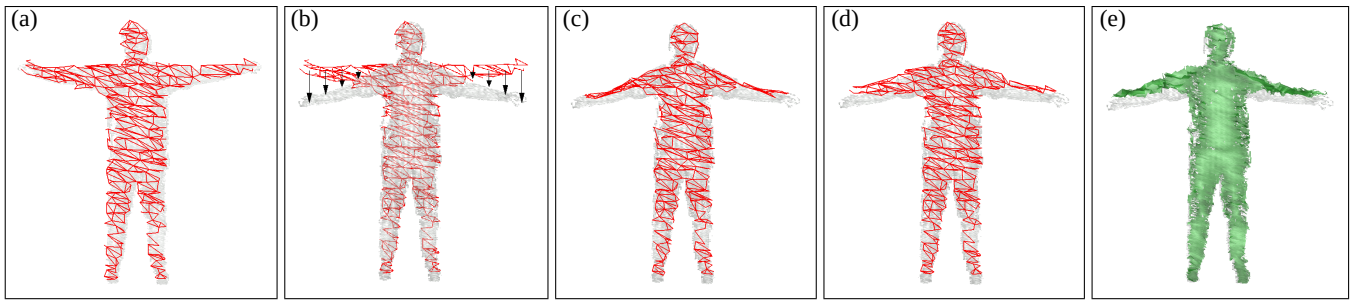


Figure 6: (a) Thinning, (b) nearest neighbor search, (c) optimization (first iteration), (d) optimization (second iteration), (e) optimization of remaining nodes.

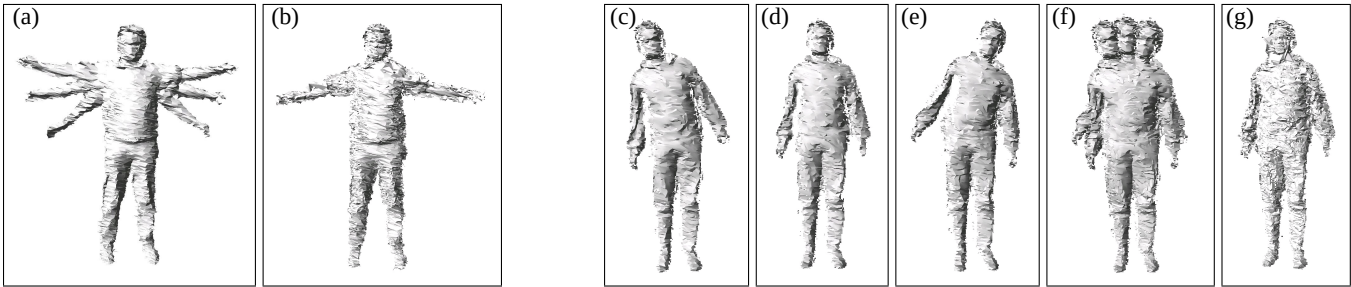


Figure 7: (a) Optimal scan registration with rigid object assumption, (b) corresponding result for our approach. (c-e) three scans, (f) results of optimal scan registration and (g) our approach.

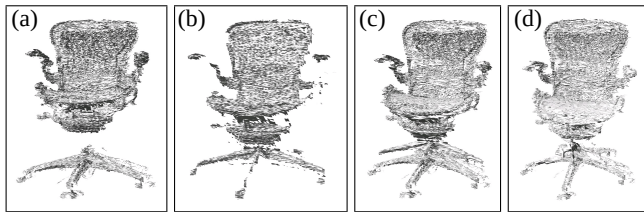


Figure 8: Adjustable chair: Scans of the chair in high position (a), scan from the chair in low position (b), the resulting model of the chair with multiple feet (c), and the resulting model of the chair with the transformed scan (d).

cannot be separated to find the corresponding part. A final direction of future research involves the integration into advanced techniques for between-object data association when modeling multiple objects in non-stationary environments [1].

References

- [1] D. Anguelov, R. Biswas, D. Koller, B. Limketkai, S. Sanner, and S. Thrun. Learning hierarchical object maps of non-stationary environments with mobile robots. *UAI-02*.
- [2] P. Besl and N. McKay. A method for registration of 3d shapes. *PAMI* 14(2), 1992.
- [3] M. Bakircioglu, S. C. Joshi, M. I. Miller. Landmark matching on brain surfaces via large deformation diffeomorphisms on the sphere. *SPIE Medical Imaging*, 1999.
- [4] L. Breiman, J. H. Friedman, R. A. Ohlsen, and C. J. Stone. Classification and regression trees. 1984.
- [5] H. Chui, A. Rangarajan. A new point matching algorithm for non-rigid registration. *CVPR*, 2000.
- [6] M. K. Chung, K. J. Worsley, S. Robbins, P. Paus, J. Taylor, J. N. Giedd, J. L. Rapoport, A. C. Evans. Deformation-Based Surface Morphometry with an Application to Gray Matter Deformation. *NeuroImage*, 18, 2003.
- [7] J. Feldmar, N. Ayache, and F. Betting. 3D-2D projective registration of free-form curves and surfaces. *CVIU*, 65(3), 1997.
- [8] J.-S. Gutmann and K. Konolige. Incremental mapping of large cyclic environments. *CIRA-00*, 2000.
- [9] E. G. Huot, H. M. Yahia, I. Cohen, I. Herlin. Surface Matching with Large Deformations and Arbitrary Topology: A Geodesic Distance Evolution Scheme on a 3-Manifold. *ECCV*, 2000.
- [10] F. Lu and E. Milios. Globally consistent range scan alignment for environment mapping. *Autonomous Robots*, 4, 1997.
- [11] M. Montemerlo, N. Roy, and S. Thrun. Carnegie mellon robot navigation toolkit, 2002.
- [12] S. Rusinkiewicz and M. Levoy. Efficient variants of the ICP algorithm. *3DIM-01*.
- [13] G. Strang. *Introduction to Linear Algebra*. Wellesley-Cambridge Press, 1998.
- [14] S. Teller, M. Antone, Z. Bodnar, M. Bosse, S. Coorg, M. Jethwa, and N. Master. Calibrated, registered images of an extended urban area. *CVPR-01*.
- [15] S. Thrun. A probabilistic online mapping algorithm for teams of mobile robots. *IJRR*, 20(5), 2001.
- [16] Y. Wang, L. H. Staib. Shape-Based 3D Surface Correspondence Using Geodesics and Local Geometry. *CVPR*, 2000.
- [17] G. Winkler. *Image Analysis, Random Fields, and Dynamic Monte Carlo Methods*. Springer, 1995.

People Tracking with Anonymous and ID-Sensors Using Rao-Blackwellised Particle Filters

Dirk Schulz, Dieter Fox, and Jeffrey Hightower

Dept. of Computer Science & Engineering
University of Washington
Seattle, WA

Abstract

Estimating the location of people using a network of sensors placed throughout an environment is a fundamental challenge in smart environments and ubiquitous computing. Id-sensors such as infrared badges provide explicit object identity information but coarse location information while anonymous sensors such as laser range-finders provide accurate location information only. Tracking using both sensor types simultaneously is an open research challenge. We present a novel approach to tracking multiple objects that combines the accuracy benefits of anonymous sensors and the identification certainty of id-sensors. Rao-Blackwellised particle filters are used to estimate object locations. Each particle represents the association history between Kalman filtered object tracks and observations. After using only anonymous sensors until id estimates are certain enough, id assignments are sampled as well resulting in a fully Rao-Blackwellised particle filter over both object tracks and id assignments. Our approach was implemented and tested successfully using data collected in an indoor environment.

1 Introduction

Accurate and reliable tracking of people using sensors placed throughout an environment is a fundamental problem relevant to several research communities. Knowing the locations of people is of critical importance for research investigating high-level state estimation, plan recognition, and learning of human activity patterns for applications such as work flow enhancement and health monitoring.

Over the years, many location estimation approaches have been introduced using sensors such as cameras, laser range-finders, infrared and ultrasound sensors, and wireless networking infrastructure [Hightower and Borriello, 2001]. A crucial aspect of these sensors is whether they provide explicit information about the identity of a person. *Anonymous sensors* such as radar, reflective ultrasound transducers, and scanning laser range-finders provide accurate location and appearance information, but do not provide explicit identity information. *Id-sensors* like infrared and ultrasound badge systems do provide explicit object identity information, but

with relatively coarse location information [Want *et al.*, 1992; Priyantha *et al.*, 2000]. Various techniques have been proposed for tracking with multiple anonymous sensors or multiple id-sensors, but the problem of integrating anonymous and id sensor information has not been addressed so far. In this paper we present an approach that combines the accuracy benefits of anonymous sensors with the identification certainty of id-sensors.

Our approach uses Rao-Blackwellised particle filters to efficiently estimate the locations and identities of multiple objects. Each particle represents a history of associations between object tracks and observations. For each particle, the individual objects are tracked using Kalman filters. Since the initial id uncertainty makes a sample-based representation of id assignments extremely inefficient, our approach starts by tracking objects using only anonymous sensors and efficiently representing estimates over object id's by keeping track of sufficient statistics. Once the id estimates are certain enough, the approach switches to sampling id assignments as well resulting in a fully Rao-Blackwellised particle filter over both object tracks and id assignments. When applied to anonymous sensors only, our method results in a new Rao-Blackwellised approach to multi-hypothesis tracking, which has gained substantial attention in the target tracking community [Bar-Shalom and Li, 1995].

This paper is organized as follows: Section 2 clarifies the problem. Section 3 then presents our Rao-Blackwellised particle filter approach to tracking multiple objects using only anonymous sensor information and Section 4 extends the approach to incorporate id-sensors. Our implementation and experimental results are presented in Section 5, followed by a discussion.

2 Problem Description

Figure 1 illustrates the problem of tracking multiple people with anonymous and id-sensors. The solid and dotted lines are the trajectories of person A and B, respectively. In the beginning, the identity of the two people is not known. As they walk, the anonymous sensor observes their locations frequently. Since the people are far enough apart, their positions can be tracked reliably using the anonymous sensor. However, until they reach id-sensor areas 3 and 4, both trajectories have the same probability of belonging to either person A or B. Hence there are two different hypotheses for the id's of the two

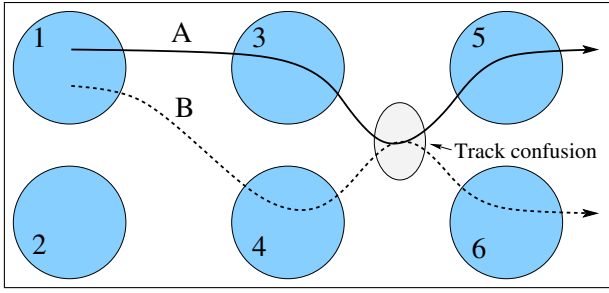


Figure 1: Example scenario: The shaded circles indicate areas covered by id-sensors such as infrared receivers. When a person wearing a badge enters such an area, the corresponding sensor issues a reading indicating the id of the person. Since these sensors provide no information about the person’s location within the area, two people in the same area can not be distinguished. Not shown is an additional anonymous sensor such as a laser range-finder. This sensor provides accurate information at a high rate about the locations of people, but no information about their id’s.

trajectories. After passing through the coverage of id-sensors 3 and 4, the ambiguity is resolved and both trajectories’ id’s are determined. Then, after the paths cross there is confusion about the continuation of the two tracks. When the people leave the light gray area the anonymous sensor can not determine which observations to associate with which trajectory. This problem is known as the *data association* problem in the multitarget tracking community [Bar-Shalom and Li, 1995]. Were there no id-sensors, it would be impossible to resolve this ambiguity. In our scenario, the ambiguity can be resolved as soon as the people reach the areas covered by id-sensors 5 and 6. To do so, however, it is necessary to maintain the hypotheses for both possible track continuations, A going down and B going up, or B going down and A going up.

The use of a combination of anonymous and id-sensors requires us to solve two types of data association problems. The first problem is the classic multitarget tracking problem of assigning anonymous observations to object tracks. This data association problem has to be solved at each point in time, resulting in $m!^k$ possible associations for tracks of length k involving m people. Fortunately, the probability distributions over these anonymous assignments are typically highly peaked, thereby allowing an efficient, sample-based representation of assignments. The second problem is the one of estimating the id of individual objects/tracks. As with anonymous assignments there are $m!$ possible assignments of id’s to tracks. Fortunately, the number of id assignments does not increase over time since the identity of objects does not change. However, due to the low spatial resolution of id-sensors, the posterior over id assignments is almost uniform during early stages of the estimation process (see Figure 1). For such “flat” distributions, a sample-based representation requires in the order of $m!$ samples, which is certainly not feasible for online tracking.

Figure 2 shows the graphical model for this tracking problem. Here, time is indexed by subscripts and the current time is denoted by k . $x_k := \{x_k^1, x_k^2, \dots, x_k^m\}$ are the current positions of the m people being tracked. Following standard notation in the tracking community, observations are denoted by

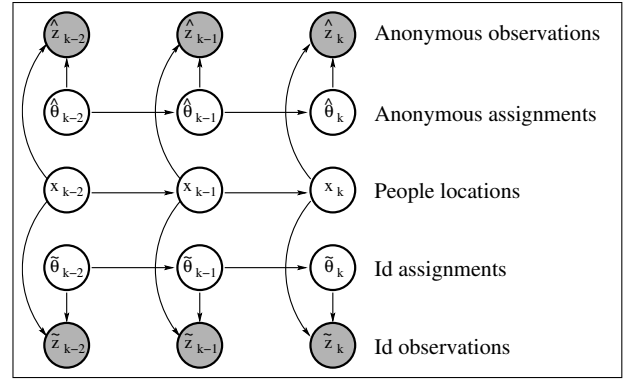


Figure 2: Graphical model for multi-object tracking with anonymous and id-sensors. x_k is the state vector describing the locations of the individual objects at time k . Objects generate anonymous observations $\hat{z}_k$ and id observations $\tilde{z}_k$. The assignments of individual observations to objects are given by the hidden nodes $\hat{\theta}_k$ and $\tilde{\theta}_k$. Rao-Blackwellised particle filters *sample* assignments $\hat{\theta}_k$ and $\tilde{\theta}_k$ and solve the state updates analytically, using Kalman filters conditioned on the samples.

z_k and the complete sequence of observations up to time k is given by $z_{1:k}$ (we use $\hat{\cdot}$ and $\tilde{\cdot}$ to distinguish anonymous from id observations). The associations between object tracks x_k^i and observations are given by assignment matrices $\hat{\theta}_k$ and $\tilde{\theta}_k$. For instance, $\hat{\theta}_k(i, j)$ is one if $\hat{\theta}_k$ assigns anonymous observation $\hat{z}_k^j$ to object x_k^i , and zero otherwise. The observations only depend on the current object positions and assignments.

The goal of tracking is to estimate the posterior over the state x_k based on all sensor information available up to time k . A factored representation of this tracking problem allows us to use Rao-Blackwellised particle filters (RBPF) which sample assignments and track the objects using a bank of m Kalman filters for each sample [Doucet *et al.*, 2000]. However, due to the large number of anonymous assignments and the “flat” distributions over id assignments, a straightforward implementation of RBPFs would require a prohibitive number of samples. In the next section we describe how to efficiently sample anonymous assignments over time. The resulting algorithm, MHT-RBPF (multi-hypothesis tracking RBPF), can track multiple objects using anonymous sensor information. Then, in section Section 4, we will describe how to extend MHT-RBPF to incorporate id-sensors.

3 MHT-RBPF: Rao-Blackwellised Particle Filters for Multi-hypothesis Tracking

Since the assignments between observations and objects/tracks are not given, we need to estimate the posterior over both object states x_k and assignments $\hat{\theta}_{1:k}$. This joint posterior can be factorized by conditioning the state x_k on the assignments $\hat{\theta}_{1:k}$:

$$p(x_k, \hat{\theta}_{1:k} | \hat{z}_{1:k}) = p(x_k | \hat{\theta}_{1:k}, \hat{z}_{1:k}) p(\hat{\theta}_{1:k} | \hat{z}_{1:k}). \quad (1)$$

The key idea of Rao-Blackwellised particle filters is to compute (1) by *sampling* assignments from $p(\hat{\theta}_{1:k} | \hat{z}_{1:k})$ and then

computing the state x_k conditioned on each sample. More specifically, each sample represents a history of data associations $\hat{\theta}_{1:k}$ and is annotated with a bank of m Kalman filters, one for each tracked person. The Kalman filters are conditioned on the data associations provided by the sample and thus can be updated efficiently using standard Kalman filter update rules for known data association [Bar-Shalom and Li, 1995]. The Kalman filter estimates, or tracks, $t_k = \langle \mu_k^i, \Sigma_k^i \rangle_{i=1, \dots, m}$ are represented by the mean and covariance of the persons' locations.

RBPf generate assignments incrementally by maintaining sample sets containing assignment histories distributed according to the posterior given by the rightmost term in (1). More specifically, at time k a sample set $S_k = \{s_k^{(\iota)}, w_k^{(\iota)} \mid 1 \leq \iota \leq N\}$ contains N weighted samples, where each sample $s_k^{(\iota)} = \langle t_k^{(\iota)}, \hat{\theta}_{1:k}^{(\iota)} \rangle$ consists of a history of assignments and the current position estimates for the m objects. The generic RBPf algorithm generates a set S_k from the previous sample set S_{k-1} and an observation $\hat{z}_k$ by first generating new assignments $\hat{\theta}_k$ distributed according to the posterior $p(\hat{\theta}_{1:k} \mid \hat{z}_k)$. Each such assignment specifies which observations in $\hat{z}_k$ belong to which object track. The final step consists of updating the Kalman filter tracks of each sample using the observations assigned to them by the sample [Doucet *et al.*, 2000].

3.1 Importance Sampling with Lookahead

The efficiency of RBPfs strongly depends on the number of samples needed to represent the posterior $p(\hat{\theta}_{1:k} \mid \hat{z}_{1:k})$. In this section we will devise an efficient algorithm for generating such assignments/samples. Due to the sequential nature of the estimation process, samples must be generated from the assignments of the previous time step. The posterior at time k is given by

$$\begin{aligned} p(\hat{\theta}_{1:k} \mid \hat{z}_{1:k}) &= p(\hat{\theta}_k \mid \hat{\theta}_{1:k-1}, \hat{z}_{1:k}) p(\hat{\theta}_{1:k-1} \mid \hat{z}_{1:k}) \\ &= \frac{p(\hat{z}_k \mid \hat{\theta}_{1:k}, \hat{z}_{1:k-1}) p(\hat{\theta}_k \mid \hat{\theta}_{1:k-1}, \hat{z}_{1:k-1})}{p(\hat{z}_k \mid \hat{\theta}_{1:k-1}, \hat{z}_{1:k-1})} p(\hat{\theta}_{1:k-1} \mid \hat{z}_{1:k}) \quad (2) \\ &= \frac{p(\hat{z}_k \mid \hat{\theta}_k, t_{k-1}) p(\hat{\theta}_k \mid t_{k-1})}{p(\hat{z}_k \mid t_{k-1})} p(\hat{\theta}_{1:k-1} \mid \hat{z}_{1:k}). \quad (3) \end{aligned}$$

Here, (2) follows from Bayes rule, and (3) follows from (2) since the position tracks t_{k-1} of the objects are sufficient statistics for the previous observations $\hat{z}_{1:k-1}$ and assignments $\hat{\theta}_{1:k-1}$.

Unfortunately, in most cases it is impossible to sample directly from (3). The approach most commonly used in particle filters is to evaluate (3) from right to left in a three stage process [Doucet *et al.*, 2001]: First, draw samples $s_{k-1}^{(\iota)}$ from the previous sample set using the importance weights, then draw for each such sample a new sample from the predictive distribution $p(\hat{\theta}_k \mid t_{k-1}^{(\iota)})$, and finally weight these samples proportional to the observation likelihood $p(\hat{z}_k \mid \hat{\theta}_k^{(\iota)}, t_{k-1}^{(\iota)})$. The last step, importance sampling, adjusts for the fact that samples are not drawn from the actual target distribution. This approach has two main sources of inefficiency. First, the samples of the previous sample set are distributed according

to $p(\hat{\theta}_{1:k-1} \mid \hat{z}_{1:k-1})$ and not the desired $p(\hat{\theta}_{1:k-1} \mid \hat{z}_{1:k})$, the rightmost term of the target distribution (3) [Morales-Menéndez *et al.*, 2002]. The second source of inefficiency is that sampling from the predictive distribution in the second sampling step can be very inefficient if the observation likelihood $p(\hat{z}_k \mid \hat{\theta}_k, t_{k-1})$ is highly peaked compared to the predictive distribution $p(\hat{\theta}_k \mid t_{k-1})$ [Pitt and Shephard, 1999]. The second problem is extremely severe in our context since the predictive distribution for assignments is virtually uniform while the posterior is typically concentrated on a small set of assignments [Bar-Shalom and Li, 1995].

MCMC Sampling from the optimal distribution

Let us first discuss how to address the second problem, *i.e.* the problem of drawing samples from the posterior distribution (3), *given* the previous sample set. Conditioned on a specific assignment $\hat{\theta}_{1:k-1}^{(\iota)}$, the optimal sampling distribution follows from (3) as

$$p(\hat{\theta}_k \mid \hat{\theta}_{1:k-1}^{(\iota)}, \hat{z}_{1:k}) \propto p(\hat{z}_k \mid \hat{\theta}_k, t_{k-1}^{(\iota)}) p(\hat{\theta}_k \mid t_{k-1}^{(\iota)}). \quad (4)$$

It is possible to efficiently generate samples from this distribution using Markov Chain Monte Carlo (MCMC) techniques [Gilks *et al.*, 1996]. We apply a version of the Metropolis-Hastings algorithm that has been adapted specifically to the data association problem [Dellaert *et al.*, 2003]. Here, the idea of Metropolis-Hastings is to sample states from an ergodic Markov chain with the posterior as stationary distribution. Such a Markov chain is constructed by first choosing a candidate for the next state s' given the current state s according to a proposal distribution $q(s' \mid s)$. This state transition is accepted with probability

$$\alpha(s, s') = \min(1, \frac{\pi(s') q(s \mid s')}{\pi(s) q(s' \mid s)}), \quad (5)$$

where $\pi(s)$ is the intended stationary distribution. In our case the states are the possible assignments $\hat{\theta}_k$ and $\pi(s)$ is the optimal sampling distribution (4). The efficiency of the Metropolis-Hastings method strongly depends on the choice of the proposal distribution q . We use an efficient approach called smart chain flipping. Smart chain flipping permutes the assignments of a subset of the objects on each transition, where the actual choice already takes the individual assignment likelihoods into account. This approach has been shown to result in improved mixing rates on assignment problems [Dellaert *et al.*, 2003].

Assignment lookahead

As mentioned above, another source of inefficiency is the fact that the previous sample set does not consider the most recent observation. This problem can be greatly reduced by using the most recent observation to re-weight the samples of the previous sample set [Pitt and Shephard, 1999; Morales-Menéndez *et al.*, 2002]. The weight for a sample $s_{k-1}^{(\iota)}$ is given by dividing the target distribution $p(\hat{\theta}_{1:k-1} \mid \hat{z}_{1:k})$ by the sampling distribution $p(\hat{\theta}_{1:k-1} \mid \hat{z}_{1:k-1})$, which can be shown to result in

$$\hat{w}_{k-1}^{(\iota)} = \frac{p(\hat{\theta}_{1:k-1}^{(\iota)} \mid \hat{z}_{1:k})}{p(\hat{\theta}_{1:k-1}^{(\iota)} \mid \hat{z}_{1:k-1})} \propto w_{k-1}^{(\iota)} p(\hat{z}_k \mid t_{k-1}^{(\iota)}). \quad (6)$$

1.	Inputs: $S_{k-1} = \{\langle s_{k-1}^{(\iota)}, w_{k-1}^{(\iota)} \rangle \mid \iota = 1, \dots, N\}$, observation $\hat{z}_k$	
2.	$S_k := \emptyset$	// Initialize
3.	for $\iota := 1, \dots, N$ do	// Generate lookahead assignments using most recent observation
4.	Generate M samples $\hat{\theta}_k^{(\iota, m)}$ proportional to $p(\hat{z}_k \mid \hat{\theta}_k^{(\iota, m)}, t_{k-1}^{(\iota)}) p(\hat{\theta}_k^{(\iota, m)} \mid t_{k-1}^{(\iota)})$ using MCMC	
5.	for $\iota := 1, \dots, N$ do	// Update importance weights based on average probability of lookahead assignments
6.	$\hat{w}_{k-1}^{(\iota)} \propto \frac{w_{k-1}^{(\iota)}}{M} \sum_{m=1 \dots M} p(\hat{z}_k \mid \hat{\theta}_k^{(\iota, m)}, t_{k-1}^{(\iota)}) p(\hat{\theta}_k^{(\iota, m)} \mid t_{k-1}^{(\iota)})$	
7.	for $\iota := 1, \dots, N$ do	// Sample $s_{k-1}^{(\iota)}$ using updated weights and draw $s_k^{(\iota)}$ from corresponding set
8.	Sample $s_{k-1}^{(\iota)} = \langle t_{k-1}^{(\iota)}, \hat{\theta}_{1:k-1}^{(\iota)} \rangle$ from S_{k-1} with probability proportional to the updated importance weights $\hat{w}_{k-1}^{(\iota)}$.	
9.	Draw an assignment $\hat{\theta}_k^{(\iota)}$ from the corresponding Markov chain generated in step 4.	
10.	Update the position estimates $t_k^{(\iota)}$ using Kalman filter updates with $\hat{z}_k$, $t_k^{(\iota)}$, and $\hat{\theta}_k^{(\iota)}$	
11.	$s_k^{(\iota)} := \langle t_k^{(\iota)}, \hat{\theta}_{1:k}^{(\iota)} \rangle$; $S_k := S_k \cup \{\langle s_k^{(\iota)}, \frac{1}{N} \rangle\}$	
12.	return S_k	

Table 1: MHT-RBPF algorithm.

That is, the importance weight of a sample is given by the ability of the tracks associated with the sample to *predict* the next observation. Accurate computation of the right term in (6) requires summation over all possible next assignments $\hat{\theta}_k$, as done by [Morales-Menéndez *et al.*, 2002]. Since in our case the number of assignments can be prohibitively large, we estimate (6) using the samples generated in the MCMC step. Unfortunately, computing $p(\hat{z}_k \mid t_{k-1}^{(\iota)})$ is equivalent to computing the normalization factor of a Markov chain, which is not possible in general [Gilks *et al.*, 1996]. In our case, however, we do not need the absolute value of the normalization constant for each chain, but only the value *relative* to the normalizers of the other Markov chains. Since the distributions (4) have similar shapes for all chains (they are highly peaked), we can estimate the relative normalization constants by the average probabilities of the samples in the different Markov chains.

More specifically, let M samples be drawn from each Markov chain. Let $\hat{\theta}_k^{(\iota, m)}$ denote the m -th sample drawn from the Markov chain associated with sample $s_{k-1}^{(\iota)}$. Then the updated weight $\hat{w}_{k-1}^{(\iota)}$ of this sample is given by

$$\hat{w}_{k-1}^{(\iota)} \propto \frac{w_{k-1}^{(\iota)}}{M} \sum_{m=1 \dots M} p(\hat{z}_k \mid \hat{\theta}_k^{(\iota, m)}, t_{k-1}^{(\iota)}) p(\hat{\theta}_k^{(\iota, m)} \mid t_{k-1}^{(\iota)}), \quad (7)$$

where proportionality is such that all weights sum up to one.

MHT-RBPF Algorithm

The algorithm is summarized in Table 1. In step 4, new samples are generated from the previous sample set. The average probability of these samples is used to estimate the lookahead/predictive weights of each sample of the previous set (step 6). This step also involves a normalization so that the weights sum up to one. In step 8, a sample is drawn from the previous sample set S_{k-1} . Then, for each sample drawn from S_{k-1} , we draw an assignment $\hat{\theta}_k^{(\iota)}$ from the posterior re-using the samples generated in the Markov chains in step 4. Step 10 updates the actual position estimates for the individual objects, using the corresponding assignment $\hat{\theta}_k^{(\iota)}$.

4 Tracking with Anonymous and ID-Sensors

In principle, MHT-RBPF's can be readily extended to include id-sensors. Instead of only sampling anonymous assignments, it is possible to sample both anonymous and id assignments. Such a straightforward extension, however, results in an infeasible increase in the number of samples needed during early stages of the estimation process. This has two reasons. First, each hypothesis (sample) of MHT-RBPF has $m!$ possible assignments of id's to object tracks. Second, due to the low spatial resolution of id-sensors, the posterior over id assignments is initially very uncertain and sample-based representations of such flat distributions are inherently inefficient. To overcome these difficulties we instead use a two-stage estimation process. During the initial stage, only anonymous sensors are used for object tracking while the id-sensors are simply used to estimate the identity of the different objects. Once these estimates are certain enough, the process moves into the full Rao-Blackwellisation phase, during which both anonymous and id assignments are sampled. The two phases are discussed below.

Identity estimation phase

During this phase only anonymous sensors are used to track the objects. The id-sensors are used to estimate the identity of the different objects. More specifically, for each hypothesis of the MHT-RBPF, there are $m!$ possible assignments $\tilde{\theta}_k$ of identities to tracks. In order to avoid estimating distributions over this potentially too large number of assignments, we only keep track of sufficient statistics that allow us to recover distributions over assignments. Such sufficient statistics are given by the m^2 individual assignments $\tilde{\theta}_k(i, j)$ of id's j to tracks t_k^i . The probabilities of these individual assignments can be updated recursively using the most recent id observation:

$$p(\tilde{\theta}_k(i, j) \mid \hat{1}_{1:k}, z_{1:k}) \propto p(\tilde{z}_k^j \mid t_k^i) p(\tilde{\theta}_{k-1}(i, j) \mid \hat{1}_{1:k-1}, z_{1:k-1}) \quad (8)$$

Here, $\tilde{z}_k^j$ is an id observation corresponding to person j . To determine the assignment probabilities, we have to normalize these values by considering all possible assignments:

$$p(\tilde{\theta}_k = \tilde{\theta}') = \frac{\prod_{(i,j) \in \tilde{\theta}'} \tilde{\theta}'(i, j) p(\tilde{\theta}_k(i, j))}{\sum_{\tilde{\theta}''} \prod_{(i,j) \in \tilde{\theta}''} \tilde{\theta}''(i, j) p(\tilde{\theta}_k(i, j))} \quad (9)$$

Again, $\tilde{\theta}'(i, j)$ is one if $\tilde{\theta}'$ assigns id j to object track i , and zero otherwise. The computational complex computation of (9) can be avoided by *sampling* id assignments using Metropolis-Hastings, based on the individual values $\tilde{\theta}_k(i, j)$. This approach works identical to the method used to sample anonymous assignments in the MHT-RBPF algorithm.

To summarize, in the identity estimation phase, each sample $s^{(i)}$ consists of m Kalman filters and an $m \times m$ matrix $\tilde{\theta}_k$ representing the sufficient statistics $\tilde{\theta}_k(i, j)$ of the id assignments. Whenever needed, the posterior over id assignments can be computed using MCMC sampling. The id estimation stage is ended as soon as the posterior over id assignments is sufficiently peaked to allow an accurate representation with a reasonable number of samples. Currently, we estimate this condition by determining the average number of hypotheses in the Markov chains run for id assignments.

Full Rao-Blackwellisation phase

Once the id's are estimated accurately enough, we begin estimating the joint posterior of both anonymous and id assignments. This posterior is given by:

$$p(\hat{\theta}_{1:k}, \hat{\theta}_{1:k} | \hat{z}_{1:k}, \hat{z}_{1:k}) \propto p(\hat{z}_k | \hat{\theta}_k, t_{k-1}) p(\hat{\theta}_k | \hat{\theta}_{k-1}, t_{k-1}) \cdot p(\hat{\theta}_k | t_{k-1}) p(\hat{\theta}_k | \hat{\theta}_{k-1}, t_{k-1}) p(\hat{\theta}_{1:k-1} | \hat{z}_{1:k}, \hat{z}_{1:k}). \quad (10)$$

Note that id assignments are sampled only once during the complete estimation process, since after an id assignment is sampled, the identities of the attached object tracks are fixed. From then on, id observations serve two purposes: First they provide information of object positions, and, second, they provide information for weighting anonymous hypotheses, thereby improving the object estimates considerably.

5 Experiments

To validate our approach we captured trace logs of six people simultaneously walking around the cubical areas of the office environment outlined in Figure 3. Each person was wearing a small id-sensor track-pack consisting of two infrared badges and an ultrasound badge. Infrared and ultrasound receivers were installed throughout the ceiling. The entire scene was continually observed by two wall-mounted laser range-finders scanning at chest height just above the cubical partitions. The duration of the log was 10 minutes, during which the individual people moved between 230 meter and 410 meter. In this challenging data log, the paths of people frequently crossed each other and there were situations in which up to 4 people were occluded by others. To validate our tracking algorithm quantitatively, we carried out a series of experiments based on this data log and on additional simulation runs (see www.cs.washington.edu/robotics/people-tracking/ for visualizations).

Tracking ability

The path of the six people as estimated by our system is shown in Figure 3. This result was obtained with an MHT-RBPF using 1000 samples and Markov chains of length 100. With this setting, the algorithm was able to reliably track the six people if no lookahead was used. After determining these parameters, we carried out trial runs for different variants of our algorithm. A run was considered successful if at the end the

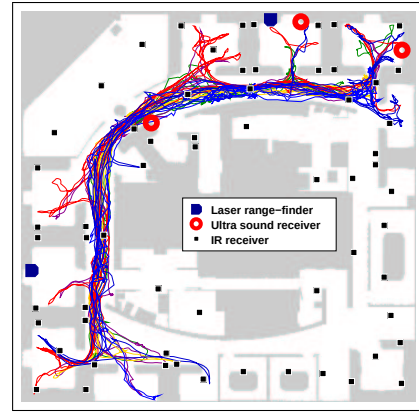


Figure 3: Outline of the Intel Research Lab Seattle. The environment is equipped with ceiling mounted ultrasound and infrared receivers. Cubical partitions are half-height (about 1.3 meters high). The two laser range-finders scan at chest height just above the partitions. Also shown are the paths of the six people as estimated by our system.

sample set contained at least one hypothesis with correct locations and identities. Each method was tested on 10 trial runs using the real data set. Without identity estimation phase, the algorithm was never able to successfully complete the data set. Next, we tested the approach when not switching to the full Rao-Blackwellisation phase, *i.e.* the algorithm remained in the identity estimation phase. Without lookahead, the update times of the approach were prohibitively large (more than 10 times real time). The lookahead resulted in significant speedup and the results are shown in the first row of the table below. Finally, we tested our two-phase approach with and without lookahead. The algorithms were able to successfully track the complete data in most cases, as can be seen in the first column. The other three columns give average time, standard deviation, and maximum time per update in seconds. The results demonstrate that both the lookahead and the two-stage process improve the performance of the tracking algorithm.

Method	Succ.	Avg.	Std.	Max.
Id estimation w. look.	10	0.062	0.125	2.9
Two-phase RBPF	10	0.154	0.170	1.5
Two-phase RBPF w. look.	9	0.036	0.060	1.2

In another set of experiments, we compared our MCMC based assignment generation to a deterministic sampling scheme, as used in traditional MHT algorithms. Here, assignments are enumerated in decreasing order wrt. their likelihood [Cox and Hingorani, 1996]. Note that generating assignments by decreasing likelihood results in estimates that are strongly biased towards more likely assignments. We performed extensive tests using real data and data simulating 20 people and found that our approach is slightly more efficient while achieving the same robustness. Since such results depend on implementational details, we did not consider them significant enough. However, the fact that smart chain flipping works at least as good as ranked assignments in practice is very encouraging, since the MCMC approach results in much less biased estimates.

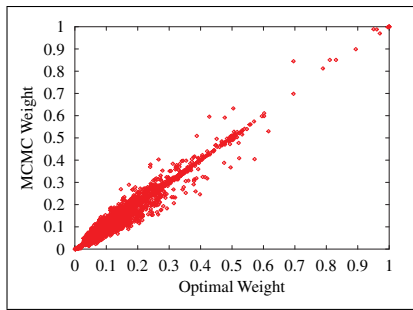


Figure 4: Optimal RBPF sample weights plotted against the sample weights computed using relative MCMC normalizers.

MCMC weight estimation

In Section 3.1, we introduced an approach to using the most recent observation in order to weight samples *before* drawing them for the next time step. So far it is not clear whether this estimation based on the probabilities of samples drawn from the Markov chains is a valid approximation. To test the quality of this approximation, we computed the optimal predictive weights by enumerating all possible assignments of the next time step. Figure 4 shows the MCMC sample weights computed in (7) plotted against the optimal weights. The similarity to $y = x$ suggests that our method of estimating Markov chain normalizers results in accurate weight estimates.

6 Conclusions and Future Work

We have presented a solution to the problem of tracking multiple people using a combination of anonymous and id sensors. The approach inherits the advantages of both sensor types, thereby being able to accurately track people and estimate their identity. Our technique uses Rao-Blackwellised particle filters to make the estimation problem tractable. We introduced several improvements to the vanilla particle filter. First, the estimation process is separated into two stages, a first stage of identity estimation and a second stage of full Rao-Blackwellisation. A second improvement is in using the most recent observation *before* sampling from the previous sample set. In contrast to [Morales-Menéndez *et al.*, 2002], the state space of our problem can become too large to allow an accurate estimate of the predictive quality of samples. Therefore, we estimate this quality using Markov chains generating samples distributed according to the posterior. We demonstrate the robustness of our approach in a challenging experiment involving six people walking through a confined office environment. We also show that our MCMC prediction is an accurate approximation of the optimal weighting function.

The approach introduced in this paper is just the first step towards a reliable and efficient tracking system. Currently, the transition between the identity estimation stage and the full Rao-Blackwellisation stage is based on a simple heuristic, namely the average number of different assignments in the Markov chains. We intend to replace this measure by a more fundamental approach such as the overall complexity of the distribution [Fox, 2002]. Another source of potential improvement lies in the handling of hypotheses. In our current system, the number of hypotheses grows extremely fast whenever several people are close to each other. We intend to overcome this

problem by clustering people into groups for which we do not attempt to estimate the individual id's, as already introduced in a similar context by [Rosencrantz *et al.*, 2003]. Finally, the recovery from tracking failures is another important issue for future research. Especially in the full Rao-Blackwellisation phase, the current approach can not recover from losing the correct id hypothesis, since id assignments do not change over time.

Acknowledgments

This work has partly been supported by the National Science Foundation under grant number IIS-0093406, and by DARPA's SDR Programme (contract number NBCHC020073). We are also very grateful for the help of various people at the Intel Research Lab Seattle for collecting the data needed for our experimental evaluation.

References

- [Bar-Shalom and Li, 1995] Y. Bar-Shalom and X.-R. Li. *Multitarget-Multisensor Tracking: Principles and Techniques*. Yaakov Bar-Shalom, 1995.
- [Cox and Hingorani, 1996] I. J. Cox and S. L. Hingorani. An efficient implementation of reids multiple hypothesis tracking algorithm and its evaluation for the purpose of visual tracking. *IEEE Transactions on PAMI*, 18(2), 1996.
- [Dellaert *et al.*, 2003] F. Dellaert, S.M. Seitz, C. Thorpe, and S. Thrun. EM, MCMC, and chain flipping for structure from motion with unknown correspondence. *Machine Learning*, 50(1-2), 2003.
- [Doucet *et al.*, 2000] A. Doucet, J.F.G. de Freitas, K. Murphy, and S. Russell. Rao-Blackwellised particle filtering for dynamic bayesian networks. In *Proc. of the Conference on Uncertainty in Artificial Intelligence (UAI)*, 2000.
- [Doucet *et al.*, 2001] A. Doucet, N. de Freitas, and N. Gordon, editors. *Sequential Monte Carlo in Practice*. Springer-Verlag, New York, 2001.
- [Fox, 2002] D. Fox. KLD-sampling: Adaptive particle filters. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14 (NIPS)*, Cambridge, MA, 2002. MIT Press.
- [Gilks *et al.*, 1996] W.R. Gilks, S. Richardson, and D.J. Spiegelhalter, editors. *Markov Chain Monte Carlo in Practice*. Chapman and Hall/CRC, 1996.
- [Hightower and Borriello, 2001] J. Hightower and G. Borriello. Location systems for ubiquitous computing. *Computer*, 34(8), 2001. IEEE Computer Society Press.
- [Morales-Menéndez *et al.*, 2002] R. Morales-Menéndez, N. de Freitas, and D. Poole. Real-time monitoring of complex industrial processes with particle filters. In *Advances in Neural Information Processing Systems 15*, 2002.
- [Pitt and Shephard, 1999] M. K. Pitt and N. Shephard. Filtering via simulation: auxiliary particle filters. *Journal of the American Statistical Association*, 94(446), 1999.
- [Priyantha *et al.*, 2000] N.B. Priyantha, A. Chakraborty, and H. Balakrishnan. The cricket location-support system. In *Proceedings of MOBICOM 2000*, Boston, MA, 2000. ACM Press.
- [Rosencrantz *et al.*, 2003] M. Rosencrantz, G. Gordon, and S. Thrun. Locating moving entities in dynamic indoor environments with teams of mobile robots. In *Second International Joint Conference on Autonomous Agents and Multiagent Systems*, 2003.
- [Want *et al.*, 1992] R. Want, A. Hopper, V. Falcao, and J. Gibbons. The active badge location system. *ACM Transactions on Information Systems*, 10(1), 1992.

PLANNING

Factored Planning

Eyal Amir and Barbara Engelhardt

Computer Science Division, University of California at Berkeley
Berkeley, CA 94720-1776, USA
{eyal,bee}@cs.berkeley.edu

Abstract

We present a general-purpose method for dynamically factoring a planning domain, whose structure is then exploited by our generic planning method to find sound and complete plans. The planning algorithm's time complexity scales linearly with the size of the domain, and at worst exponentially with the size of the largest subdomain and interaction between subdomains. The factorization procedure divides a planning domain into subdomains that are organized in a tree structure such that interaction between neighboring subdomains in the tree is minimized. The combined planning algorithm is sound and complete, and we demonstrate it on a representative planning domain. The algorithm appears to scale to very large problems regardless of the black box planner used.

1 Introduction

Many planning algorithms use independencies or loose interactions between components in the planning domain to find plans more efficiently. For example, hierarchical planners divide a goal into subgoals (high-level operators) using a decomposition of the domain into loosely interacting parts (e.g., [Knoblock, 1990; Erol *et al.*, 1994]). Planning is done at each level separately, and later the subplans are pieced together to build a valid plan. Other planners that use domain decomposition (e.g., [Lansky and Getoor, 1995]) are not limited to domains with explicit hierarchical structure, but require backtracking across subdomains, and often replan for subgoals that already have valid plans. In general, domain-decomposition planning does not scale well because backtracking between subdomains can dominate the complexity of the search, and the domain decomposition is often ad hoc.

In this paper we present an approach to planning that scales to very large domains by taking advantage of domain structure. This approach is composed of two procedures: factoring and planning. The factoring procedure partitions a planning domain into loosely interacting subdomains that are organized in a tree structure. Our factoring procedure uses decomposition algorithms from graph theory, and our contribution in this matter is in translating the planning-problem decomposition task into a graph decomposition task. After factoring,

our planning procedure finds plans for multiple subgoals in each of the subdomains separately, using a generic black box planner. It searches over possible plans using complex action descriptors from each of the subdomains to form a plan for the overall goal. The planning procedure uses dynamic programming principles, and backtracking occurs only within a subdomain as part of the black box planner.

We prove that our planning procedure runs in time linear in the number of subdomains and takes time that is at most exponential in the size of the largest subdomain and the number of dependencies between subdomains. The type of factoring that we select is justified by this complexity result. We also prove that the combined algorithm is sound and complete, and that it can be applied to solve any planning problem using any generic black box planner for planning within subdomains. The complexity is upper bounded by the complexity of the black box planner on the unpartitioned domain.

We implemented and tested our planning algorithm on a simple domain to guide further development. We created two implementations, one with the IPP planning system [Koehler and Hoffmann, 2000] and one with the FF planner [Hoffmann and Nebel, 2001]. We compared the results of our algorithm with those of IPP and FF, and have shown that for a single domain our results scale much better than these planners alone. The example validates our analytical results and shows that our planner's performance scales linearly with the size of the domain for this problem, motivating further development.

2 Factored Planning

In this section we present an algorithm for planning with a partitioned planning domain, where the subdomains are arranged in a rooted tree. Each subdomain corresponds to a subset of the fluents, or single states, in the domain and a subset of the actions which use only those fluents. Neighboring subdomains may share fluents and actions, and the ideal partition has the smallest number of shared fluents. We analyze this algorithm computationally and present a procedure for finding a tree partition with close-to-optimal properties.

2.1 Partitioned Planning Problems

We restrict ourselves to partitioned planning problems that are described using a simple propositional action language. A *partitioned domain description* is a labeled graph that describes a set of subdomains and the connections between the

subdomains. Formally, a partitioned domain description is a labeled graph $G(\mathbb{D}, E, l)$ with

- $\mathbb{D} = \{D_i\}_{i \leq m}$ vertices: for all i , $D_i = \langle P_i, \mathcal{A}_i \rangle$ is a planning domain description with
 - P_i a set of propositional *fluents* (features of our domain that may change value over time),
 - $\mathcal{A}_i$ a set of action (operator) definitions over P_i .
- Edges E over $\mathbb{D}$, and
- A labeling l of the edges in E . For every $D_u, D_v \in \mathbb{D}$, $l(D_u, D_v)$ is a subset of fluents from $\bigcup_{i \leq m} P_i$.

The label $l(D_u, D_v)$ includes at least the fluents shared between D_u, D_v (i.e., $P_u \cap P_v$), but it may include other (e.g. global) fluents. We use the convention that *action* A_i refers to the action definition (also called *action schema*) and that a_i denotes the instantiation of the action in a plan. We say that action a_i is in subdomain D_u iff it is defined in $\mathcal{A}_u$.

We specify the preconditions and effects of actions in $\mathcal{A}_i$ using the simple language of situation calculus (with propositional fluents), and allow actions to have conditional effects. We use the simple compilation technique of [Reiter, 2001] to provide monotonic semantics for this language (and avoid the frame problem). All of our results apply to the PDDL language [Fox and Long, 2002] as well. PDDL is the representation language used in our example (Section 3) to model the domains, the subdomains, and the plans.

2.2 Planning Algorithm

Here we present our planning algorithm, *PartPlan* (Figure 2), which finds plans in a partitioned domain. We use the following notation. For P a set of propositional fluents, $\mathcal{L}(P)$ is the language of P , i.e., the set of propositional sentences that can be built with P .

High-Level Overview

PartPlan processes a partitioned domain tree iteratively from the leaves to the root (the subdomain that holds the goal of *PartPlan*). It selects a leaf subdomain, tries to build plans for each of a set of possible preconditions and goals (Step 2b), records the successful attempts, adds them as new (macro-like) actions to this leaf’s single parent subdomain (Step 2c), and removes that leaf subdomain from the graph.

The result of planning in each subdomain is not a complete plan, but rather a compilation of capabilities, which contain both subdomain actions and capabilities of descendant subdomains. Generally, every subdomain’s *capabilities* are those plans that it can find (using its own actions and capabilities of its neighbors) that affect the fluents that it shares with its parent subdomain in the tree. Capabilities of a subdomain are shared by sending a message to the parent domain containing a complex action describing the capability.

When the tree is reduced to the single subdomain (after the iterative processing and removal of leaves) that contains the problem goal, we perform planning in that node to achieve the goal condition, based on the capabilities from the (eliminated) child subdomains (Step 3). Then, the plan found by this subdomain is expanded into a plan that achieves the overall problem goal, using the actions of this subdomain along

with the reported capabilities of the children in the tree. Every capability that appears in this plan is expanded into a subplan by the subdomain that reported it. Those subplans can themselves contain capabilities that are then expanded, and the process continues until the plan contains only the original domain actions.

The Simple Algorithm is Incomplete

The view presented above leaves the exact nature of planning in each subdomain unspecified. It suggests, roughly, that planning in each subdomain takes the form of finding a plan that starts from some initial state of local fluents and reaches a specified end state of local fluents. Unfortunately, limiting ourselves to this kind of message from children subdomains to parents is incomplete, as demonstrated in the following example (see Figure 1 for an illustration).

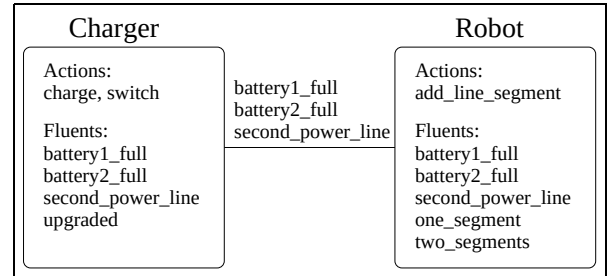


Figure 1: Subdomains for upgrading a battery charger.

Consider a scenario in which we have a charger that can charge batteries and a robot that can connect power cables together and that uses two batteries. Initially, every *charge* action of the charger charges exactly one battery, so two *charge* actions are required to charge both of the robot’s batteries. However, if the charger is connected to an additional power source, then a single *charge* action charges two batteries at once. Our task is to upgrade the charger so that it can charge two batteries at once. For this, the robot needs to build a power line that is long enough to reach the charger, and the charger needs to switch to using both sources. Thus, building this line requires multiple charges interleaved with adding lines. A plan that solves this problem needs to charge the robot’s batteries, add a line segment, charge again, add another line segment, charge again, add the final line segment, and switch the charger to the new mode.

We want our factored-planning approach to apply to any given planning problem and be complete (i.e., find a plan if there is one) regardless of the plan partition. We will show that there is a partitioning of this problem so that simple messages of the form “if X holds, then I can make Y hold” are not sufficient in a directed tree to find a complete plan. Messages of the form “if X_0 holds, then I can cause Y_0 ; if you subsequently make X_1 hold, then I can cause Y_1 to hold” are required for domains with interleaved interactions as in this example. These complex messages can be extended for larger numbers of interleaved preconditions and effects, and the algorithm we present allows that type of message.

One possible partitioning of the example above is presented in Figure 1. On the left-hand side we include the

actions *charge* and *switch*, and the features *battery1_full*, *battery2_full*, *second_power_line*, and *upgraded*. The action *charge* has the following effects and preconditions:

- *charge* causes *battery1_full*.
- If *battery1_full* or *upgraded*, then *charge* causes *battery2_full*.

Action *switch* has the following effects and preconditions:

- If *second_power_line*, then *switch* causes *upgraded*.

On the right-hand subdomain we include the action *add_line_segment*, and the features *battery1_full*, *battery2_full*, *second_power_line*, *one_segment*, and *two_segments*. The action *add_line_segment* has the following effects and preconditions:

- If *battery1_full* $\wedge$ *battery2_full*, then *add_line_segment* causes *one_segment* $\wedge$ $\neg$ *battery1_full* $\wedge$ $\neg$ *battery2_full*.
- If *one_segment*, then *add_line_segment* causes *two_segments*.
- If *two_segments*, then *add_line_segment* causes *second_power_line*.

Now, consider all the possible *simple* messages, i.e., messages of the form “if *X*, then I can cause *Y* to hold”. We require each of those *X* and *Y* to be a combination of fluent values for fluents that are shared between the two subdomains. In our case, those shared fluents are *battery1_full*, *battery2_full*, and *second_power_line*.

Assume that we chose the root node in our partition to be the left-hand side subdomain. The only non-trivial message that the right-hand side can send is

- If *battery1_full* $\wedge$ *battery2_full*, then I can cause $\neg$ *battery1_full* $\wedge$ $\neg$ *battery2_full*.

We would like to find a plan in the root subdomain (the left-hand side subdomain) that uses these messages and that results in the charger being upgraded. Note that this is not possible with only simple messages.

One way to resolve this problem is to allow more complex messages. Particularly, in our example, we can allow the following message to be sent:

If *battery1_full* $\wedge$ *battery2_full*,
then I can cause $\neg$ *battery1_full* $\wedge$ $\neg$ *battery2_full* and
if subsequently *battery1_full* $\wedge$ *battery2_full*,
then I can cause $\neg$ *battery1_full* $\wedge$ $\neg$ *battery2_full* and
if subsequently *battery1_full* $\wedge$ *battery2_full*,
then I can cause *second_power_line*. (1)

If our planner on the left-hand subdomain can handle such complex actions, then it can use such a message to find a plan that succeeds in upgrading the charger. Indeed allowing this kind of messages makes our planning algorithm complete. This is consistent with results achieved by [Amir and McIlraith, 2000; McIlraith and Amir, 2001] for inference in first-order logic, and is also consistent with [Amir, 2002] that presented results for projection in factored domains.

An alternative to using such long messages is to include internal (i.e., not shared) fluents in shorter messages. However, this has the effect of breaking the encapsulation and structure

that we strive to employ, reducing our problem to that of traditional planning on the entire domain and forcing backtracking across different subdomains.

Detailing the Planning Algorithm

Procedure *PartPlan* is presented in Figure 2 and its subroutines are presented in Figures 3 and 4. *PartPlan* is given a partitioned planning domain $G(\mathbb{D}, E, l)$, $\mathbb{D} = \{D_i\}_{i \leq m}$, a set of initial conditions $I = \{I_i\}_{i \leq m}$, indexed by subdomain such that I_i is the initial state for the fluents in P_i , a goal state condition Q in $\mathcal{L}(P_t)$ for root node D_t , and search parameters k (interactions) and d (depth). It returns a plan for achieving the specified goal conditions in the given domain, if such a plan exists under the search constraints. Notice that we show goals that can be represented in a single subdomain. We illustrate how to get around this restriction in Section 2.5.

```

PROCEDURE PartPlan( $G(\mathbb{D}, E, l)$ ,  $I, Q, k, d$ )
 $\mathbb{D} = \{D_i\}_{i \leq m}$  a partitioned planning domain with  $G$  a tree,  $I = \{I_i\}_{i \leq m}$  a set of initial conditions, indexed by subdomain,  $Q$  a goal in  $\mathcal{L}(P_t)$  ( $t \leq m$ ), and  $k, d$  parameters.

1. Set  $W \leftarrow \mathbb{D}$ ,  $G_W \leftarrow G$ . Orient the tree  $G_W$  such that it is rooted in  $t$  (i.e., every  $v \in W$ ,  $v \neq t$ , is a descendant of  $t$ ).
2. Do until  $W = \{t\}$  (i.e., until  $W$  includes only the root):
    (a) Take  $D_v \in W$  a leaf in  $G_W$ , and let  $D_u$  be its parent.
    (b) For every  $\{\text{pre}_i\}_{i < k}$ ,  $\{\text{eff}_i\}_{i \leq k}$  truth assignments to  $l(D_u, D_v)$ :
         $\langle \pi, j \rangle \leftarrow \text{SinglePlan}(D_v, I_v, \{\text{pre}_i\}_{i < k}, \{\text{eff}_i\}_{i \leq k}, d)$ 
        If  $\pi \neq \text{nil}$ ,
            call  $\text{StorePlan}(\pi, T, v, \{\text{pre}_i\}_{i \leq j}, \{\text{eff}_i\}_{i \leq j}, \text{eff}_k)$ .
    (c) Call  $\text{SendMessage}(v, T, A_u)$ .
    (d) Remove  $D_v$  from  $G_W$ , and set  $W \leftarrow W \setminus \{D_v\}$ .
3. Let  $\langle \pi, j \rangle \leftarrow \text{SinglePlan}(D_t, I_t, \{Q\}, d)$ .
    If  $\pi = \text{nil}$ , return “FAILURE for  $d, k$ ”.
4. Do until  $W = \mathbb{D}$  (reconstructing  $G_W$ ):
    (a) Select  $D_u \in W$  and  $D_v \in \mathbb{D} \setminus W$  neighbors in  $G$ .
    (b)  $\pi \leftarrow \text{ExpandPlan}(\pi, D_u, D_v)$ ;
    (c)  $W \leftarrow W \cup \{D_v\}$ .
5. Return  $\pi$ .

```

Figure 2: Planning with Subdomains

We discuss two of the subroutines of *PartPlan* in detail. The first, *SinglePlan*, is presented in Figure 3. It determines whether there is a plan with at most d actions in subdomain D_v that starts with the state described by the initial state I_v and arrives at a state that satisfies eff_k . We require the plan to have $k - 1$ preconditions interleaved within k effects, corresponding to the number of interleaved conditions in the message (e.g., in (1), the message sent in our example above, $k = 4$). To find such an interleaving message the planner uses *fluent-setting actions*, which set the fluents in $l(D_u, D_v)$ to a truth assignment in one of $\{\text{pre}_i\}_{i < k}$. They correspond to (yet undetermined) action sequences that can be performed in ancestor (or *cousin*) subdomains. If the parent subdomain wishes to use the result of this planning, it will need to find a sequence of actions that replaces these *fluent-setting actions*.

Every fluent-setting action, b_i , can be used at most once, and all of $b_1, \dots, b_{i-1}$ must have been used before this action. Also, prior to using this fluent-setting action, eff_{i-1} must hold. This condition corresponds to the state that is required by the other subdomain's complex action as a precondition for the subsequent complex action.

SUBROUTINE *SinglePlan*($D_v, I_v, \{\text{pre}_i\}_{i < k}, \{\text{eff}_i\}_{i \leq k}, d$)
 $D_v = \langle P_v, E_v \rangle$ a domain description, I_v an initial condition formula for D_v , $\{\text{pre}_i\}_{i \leq k-1}$ (preconditions) and $\{\text{eff}_i\}_{i \leq k}$ (effects) are state description formulae in some set of fluents $P \subseteq P_v$, d the depth of planning explored.

1. Let $P' \leftarrow P_v$, $E' \leftarrow E_v$ and $I' \leftarrow I_v$.
2. Add the fluents $\{did(b_i)\}_{0 < i < k}$ to P' .
3. For every $0 < i < k$ add a new action definition to E' (for $i = 1$ omit $did(b_{i-1})$):

$$\text{Poss}(b_i, s) \Leftrightarrow \text{Holds}(did(b_{i-1}) \wedge \neg did(b_i) \wedge \text{eff}_{i-1}, s)$$

$$\text{Poss}(b_i, s) \Rightarrow \text{Holds}(\text{pre}_i \wedge did(b_i), \text{res}(b_i, s))$$
4. Add the following initial condition to I' :

$$\text{Holds}(\neg did(b_1) \wedge \dots \wedge \neg did(b_{k-1}), S_0)$$
5. Search for a plan in P', E', I' that achieves eff_k with at most d actions. If found plan π , then take $j = \max\{i \mid b_i \in \pi\} \cup \{0\}$ and return $\langle \pi, j \rangle$. Otherwise, return nil.

SUBROUTINE *StorePlan*($\pi, T, v, \{\text{pre}_i\}_{i < j}, \{\text{eff}_i\}_{i \leq j}, \text{eff}_k$)
 P a plan, T a table, v a subdomain index, $\{\text{pre}_i\}_{i < k}$, $\{\text{eff}_i\}_{i \leq k}$ and eff_k are state description formulae.

1. Set $T(v, \{\text{pre}_i\}_{i \leq j}, \{\text{eff}_i\}_{i \leq j}, \text{eff}_k) \leftarrow \pi$.
2. For all $j' < j$,
 (a) Let $\pi_{j'}$ be the initial part of π up to $b_{j'}$ (not including).
 (b) Set $T(v, \{\text{pre}_i\}_{i < j'}, \{\text{eff}_i\}_{i < j'}, \text{eff}_{j'}) \leftarrow \pi_{j'}$.

Figure 3: Planning in a single subdomain, and storing a plan.

Subroutine *SendMessage* (Figure 4) takes a plan found by subdomain D_v (stored in table T by subroutine *StorePlan*) and adds a new complex action definition in the parent subdomain, D_u , whose initial state and goal are fluents found in D_v . This new action contains complex action definitions from child subdomains and actions local to D_u . Plans execute sequentially to ensure that the preconditions for complex actions are satisfied at the appropriate times.

2.3 Properties of *PartPlan*

In this section we prove the completeness and soundness of *PartPlan* and analyze its computational behavior. We pursue the intuition that our planning algorithm can find a plan more efficiently if the number of interactions between subdomains is small. The maximum shared fluents across the partition determines the time complexity of the algorithm.

Definition 2.1 (Width of a Plan) Let $G(\{D_i\}_{i \leq m}, E, l)$ be a partitioned planning domain tree, and $\pi = \langle a_1, \dots, a_l \rangle$ a sequence of actions. The width of π for G is the largest k

SUBROUTINE *SendMessage*(v, T, D_u)
 v a node from which messages are sent, T a table containing plans, D_u a domain for part u .

1. For every table entry $T(v, \langle \text{pre}_i \rangle_{i \leq j}, \langle \text{eff}_i \rangle_{i \leq j}, \text{eff}_k)$ ($j \leq k$), create a complex action symbol $c_{\langle \text{pre}_i \rangle_{i \leq j}, \langle \text{eff}_i \rangle_{i \leq j}, \text{eff}_k}$.
2. For every table entry $T(v, \langle \text{pre}_i \rangle_{i \leq j}, \langle \text{eff}_i \rangle_{i \leq j}, \text{eff}_k)$ ($j \leq k$), add the following new complex action definition to $\mathcal{A}_u$:

$$\text{Poss}(c, s) \Leftrightarrow \text{Holds}(\text{pre}_j \wedge (\text{last}(v) = c'), s)$$

$$\text{Poss}(c, s) \Rightarrow \text{Holds}(\text{eff}_k \wedge (\text{last}(v) = c), \text{res}(c, s))$$

$$c = c_{\langle \text{pre}_i \rangle_{i \leq j}, \langle \text{eff}_i \rangle_{i \leq j}, \text{eff}_k}, c' = c_{\langle \text{pre}_i \rangle_{i \leq j-1}, \langle \text{eff}_i \rangle_{i \leq j-1}, \text{eff}_j}$$
 (for $j = 1$ let $c' = \text{nil}$).
3. Add the following new initial state declaration to I_u :

$$\text{Holds}(\text{last}(v) = \text{nil}), S_0)$$

SUBROUTINE *ExpandPlan*(π, D_u, D_v)
 π is a plan, D_u, D_v planning domains.

1. While π has complex actions from D_v :
 (a) Let c be the last complex action from D_v in π .
 (b) Find $T(v, \langle \text{pre}_i \rangle_{i \leq j}, \langle \text{eff}_i \rangle_{i \leq j}, \text{eff}_k)$ the matching table entry for c in T , for some $k > j > 0$. Let π_c be the plan in this entry.
 (c) Find the latest instances of $c_1, \dots, c_j$ complex actions of D_v in π such that for every $l \leq j$ $c_l = c_{\langle \text{pre}_i \rangle_{i < l}, \langle \text{eff}_i \rangle_{i < l}, \text{eff}_l}$.
 (d) For every $1 < i \leq j$, replace c_i in π with the sequence of actions in π_c that is between b_{i-1} and b_i . Replace c_1 in π with the sequence of actions in π_c that is before b_1 . Replace c in π with the sequence of actions in π_c that is after b_j .
2. Return P .

Figure 4: Sending a message, and expanding a plan.

(minus one) for which there is a subsequence of actions and an edge (D_u, D_v) in G such that a_i, a_{i+1} are in subdomains on different sides of (D_u, D_v) in G , for every $i < k$.

The plan-width k for our partitioned domain is the least k that will allow *PartPlan* to find a plan. In our example in Figure 1 $k = 4$ because for $k < 4$ there is no message that we can send that will allow us to find a plan in the left-hand subdomain. Also, notice that every plan π for D has width at most $|\pi| - 1$.

A *tree decomposition* of planning domain D is a partitioned domain description tree $G(\mathbb{D}, E, l)$ satisfying the running intersection property: if a fluent f appears in P_i and also in P_j , then all the edge-labels on the path between D_i, D_j in G must include f .

Theorem 2.2 Let $G(\mathbb{D}, E, l)$, be a tree decomposition of planning domain D with m subdomains. *PartPlan*(G, I, Q, k, d) returns a plan π of that achieves Q in D starting from I , if and only if one exists with width at most $2k$ and at most $d \cdot m$ actions.

The role of d is to bound the length of the complex actions to at most d interactions; bounding this variable can be used in

an iterative deepening algorithm to find plans with the shortest possible complex actions. The parameter k can be used similarly, and its role is to restrict the length of messages that are sent from children subdomains to parent subdomains.

Corollary 2.3 *Iterative deepening of search parameters k, d in PartPlan is a sound and complete planning algorithm.*

The time bound for procedure *PartPlan* is exponential in the number of fluents in the links between subdomains and the plan-width of the partitioned-planning problem, but is linear in the number of subdomains.

Theorem 2.4 *PartPlan(D, I, Q, k, d) terminates in time $O(m \cdot 2^{2k+l} \cdot \min((a+k)^d, k \cdot 2^w))$, for a, w being the largest number of action symbols and largest number of fluent symbols in any subdomain, respectively, and l being the largest number of symbols in $l(D_u, D_v)$ for all subdomains D_u, D_v .*

2.4 Automatic Factoring

Theorem 2.2 assumes that a tree decomposition $G(\mathbb{D}, E, l)$ is given. For a planning domain D we can build the decompositions by hand, adding a knowledge-engineering and planning perspectives to the domain description. We can also find a decomposition automatically using graph decomposition techniques. In this section we describe such an algorithm.

The notion of tree decomposition in planning domains is a special case of tree decomposition in graphs [Robertson and Seymour, 1986]. The width of a tree-decomposition is the width of the largest subset in the tree minus one. The *treewidth* of a graph $G(V, E)$ is the minimum width over all tree-decompositions of G minus one

Finding optimal tree decompositions is NP-hard. Nonetheless, there are several algorithms that find close-to-optimal tree decompositions (e.g., [Becker and Geiger, 1996; Amir, 2001]), and there are also some heuristics (e.g., [Kjaerulff, 1993]) that have been applied successfully in inference algorithms for Bayesian networks and theorem proving.

The reduction of the planning decomposition problem into a graph decomposition problem is as follows. We create a graph $G(V, E)$ with a vertex $v \in V$ for every propositional fluent in our domain D . E is the set of edges in this graph, and it includes an edge between $v, u \in V$ if and only if there is an action definition that includes both v, u (either as a precondition or as an effect). In the next step we find a tree decomposition (S, T) , $S = \{X_i \mid i \leq m\}$, $T = (I, F)$ for this graph using one of the algorithms mentioned above. Finally, we create a partitioned domain description $G_D(\{D_i\}_{i \leq m}, E_D, l)$ as follows: The vertices (domain descriptions) $D_i = \langle P_i, \mathcal{A}_i \rangle$ are chosen such that P_i includes all the fluents in X_i , and $\mathcal{A}_i$ includes all the action definitions that can be expressed with the fluents in P_i . The edges in E_D are the tree edges, F , in T . The labeling l of an edge $(D_u, D_v) \in E_D$ is the set of fluents that are shared between the vertices D_u, D_v , i.e., $l(u, v) = P_u \cap P_v$.

This algorithm complexity is taken up by the decomposition of the graph $G(V, E)$. Most heuristics are at most $O(n^2)$, and one approximation algorithm [Amir, 2001] takes time $O(n^3 \log^4 n \cdot k^5 \log k)$ (k is the treewidth of G , and is typically much smaller than n). This is an upper bound on the time taken by the planning domain decomposition algorithm.

2.5 Distribution of the Goal

So far we have assumed that the goal is represented in the set of fluents of a single subdomain. In many planning problems the goal consists of a conjunction of conditions on many different fluents. If we create a single subdomain that includes all those fluents, then the width of the tree decomposition will be high, and so will be the planning time. Instead, we can distribute the goal between subdomains, as we show here.

Assume that we are given a tree decomposition $G(\mathbb{D}, E, l)$ of domain D , with $\mathbb{D} = \{D_i\}_{i \leq m}$. Let a goal condition be $\bigwedge_{i \leq m} Q_i$, where Q_i is a part of the goal represented with the fluents in P_i . Arbitrarily, choose a single subdomain, D_t , as the root of the tree. To each subdomain D_i we add a new fluent symbol, q_i , and a new action, a_i^q . For every leaf subdomain, a_i^q 's precondition is Q_i and its effect is q_i . For every internal node in the tree, with children $D_{j_1}, \dots, D_{j_s}$, a_i^q 's precondition is $q_{j_1} \wedge \dots \wedge q_{j_s}$, and its effect is q_i .

When planning with this revised decomposition we use a single goal, q_t , in subdomain D_t . The revised planning problem in this tree decomposition is equivalent to the original one, and we have reduced a (worst case) exponentially large problem down to a linear one.

3 An Extended Example

Consider the following domain, where a robot can move around in a set of rooms connected by doors as a ring [Cimatti and Roveri, 2000]. Each room has a window which can be open, closed, or locked. The robot has four possible actions: move clockwise, move counterclockwise, close the window of the current room, and lock the window of the current room if the window is already closed. The goal state is that the windows of all the rooms are locked. The start state is the robot in one of the r rooms and some configuration of windows open, closed, and locked. In this simple example we assume that the state of windows and the location of the robot are known. In other words, the initial condition is one of $r \cdot 3^r$ possible initial configurations.

Define the following set of fluents:

- Agent location: $in_1, \dots, in_r$
- Windows closed: $closed_1, \dots, closed_r$
- Windows locked: $locked_1, \dots, locked_r$
- Windows locked from room 1 up to room i : $all_1, \dots, all_r$.

The all_i fluents represent conjunctions of the $locked_1, \dots, locked_i$ fluents from parent and ancestor subdomains in the tree, as in the previous section. Define the following set of actions and corresponding effect axioms:

- Actions: $close, lock, move_{cl}, move_{ccl}$
- Effect axioms:

$$\begin{aligned}
 in_i(s) \wedge \neg closed_i(s) &\Rightarrow closed_i(res(close, s)) \\
 in_i(s) \wedge closed_i(s) &\Rightarrow locked_i(res(lock, s)) \\
 in_i(s) &\Rightarrow \\
 in_{i+1}(res(move_{cl}, s)) \wedge \neg in_i(res(move_{cl}, s)) & \\
 in_i(s) &\Rightarrow \\
 in_{i-1}(res(move_{ccl}, s)) \wedge \neg in_i(res(move_{ccl}, s)) & \\
 locked_1(s) &\Rightarrow all_1(s) \\
 all_{i-1}(s) \wedge locked_i(s) &\Rightarrow all_i(s) \quad (1 < i \leq m)
 \end{aligned}$$

In the last set of axioms we use the convention that addition and subtraction are modulo the number of rooms in the ring. In other words, the next room after i is $(i \pm 1 + m) \bmod m$.

For the ring of rooms, an obvious partition turns out to be a good one. Specifically, we associate each subdomain with one room, letting the fluents and actions for subdomain D_i be

$$P_i = \{in_i, closed_i, locked_i, in_{i-1}, in_{i+1}, all_{i-1}, all_i\}$$

$$\mathcal{A}_i = \{close, lock, move_{cl}, move_{ccl}\}$$

Each subdomain contains all actions whose preconditions and effects mention only fluents in that subdomain.

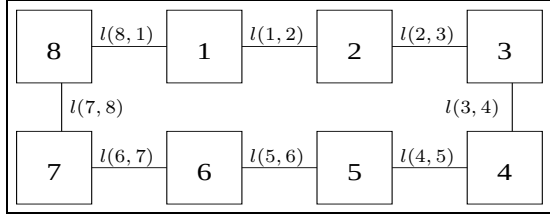


Figure 5: Subdomains for Ring of Rooms.

The graph of subdomains is not a tree (see Figure 5). One way to turn it into a tree decomposition is by removing one of the edges completing a cycle and adding (in the worst case) the label of that edge to the rest of the edges in this graph (this is an example of a more general algorithm that is outside the scope of this paper; see [Amir and McIlraith, 2000]). The result is presented in Figure 6. The graph G connects every two consecutive subdomains but D_1, D_m , for m being the number of rooms (and subdomains). The labeling on the edges is originally $l(i, i+1) = \{in_i, in_{i+1}, all_i\}$ for $i \leq m$. After transforming to graph into a tree we get the labeling

$$l'(i, i+1) = l(i, i+1) \cup l(m, 1) = \{in_i, in_{i+1}, all_i, in_m, in_1\}.$$

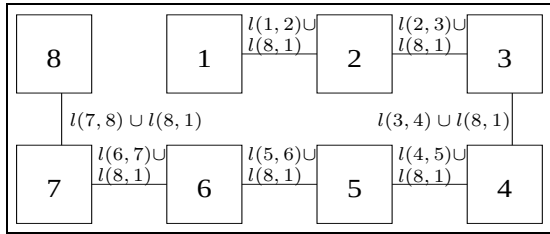


Figure 6: Tree decomposition for Ring of Rooms.

We show how *PartPlan* finds a plan for this domain. We set the goal to be all_m , which is expressed in the fluents of subdomain m . We set $t = m$ to be the root of the tree.

We take node $v = 1$, whose parent in the tree is $u = 2$, and find all the plans possible with $k = 2, d = 2$. One such plan is “if the robot is in room 1 and the window is open, then close, lock and move clockwise.” We find a plan for D_1 :

$$pre_1 = in_1 \wedge \neg in_2 \wedge \neg all_1 \wedge \neg in_m$$

$$eff_2 = \neg in_1 \wedge in_2 \wedge all_1 \wedge \neg in_m.$$

Both pre_1, eff_2 are complete state descriptions in $l'(1, 2)$. There is a plan π_1 that *SinglePlan*($D_1, I_1, \langle pre_1 \rangle, \langle eff_1 \rangle, eff_2$,

d) will return. Thus, one message that is sent to subdomain D_u is “if in some state the robot is in room 1, then D_1 has a sequence of actions that results in all_1 being true and the robot moving to room 2.” In more precise terms, the definition for this capability $c_{2,1}$ is added to subdomain D_u by subroutine *SendMessage*.

Now, we take node $v = 2$, whose parent in the tree is $u = 3$, and find all the plans possible with $k = 2, d = 2$. One such plan is, “if in some state the robot is in room 2, then move counter-clockwise, execute a subdomain-1-action, then close, lock and move clockwise.” We find a plan for D_2 :

$$pre_1 = in_2 \wedge \neg in_3 \wedge \neg all_2 \wedge \neg in_m \wedge \neg in_1$$

$$eff_2 = \neg in_2 \wedge in_3 \wedge all_2 \wedge \neg in_m \wedge \neg in_1,$$

These are all complete state descriptions in $l'(2, 3)$. There is a plan $\pi(c_2)$ that *SinglePlan*($D_2, I_2, \{pre_1\}, \{eff_1\}, eff_2, d$) will return. The message that is sent to subdomain D_u is roughly speaking “if the robot is in room 2, then D_2 has a sequence of actions that results in all_2 being true”. In more precise terms, the definition for action $c_{2,1}$ is added to subdomain D_u by subroutine *SendMessage*.

This method continues for all the subdomains up to D_m that contains the goal. Subdomain m can conclude that there exists a plan for closing and locking all the windows, satisfying all_m . If the initial condition of our problem specifies that the robot is in room r , then starting in subdomain D_r we will already have a plan independent of the position of the robot. This will lead to a complete plan in subdomain m .

When the plan is found in subdomain m , the algorithm then goes on to expanding the complex actions into their capability plans. For example, when the complex action $c_{2,1}$ of subdomain 1 is encountered in the plan π , it is expanded into a segment $\langle close, lock, move_{cl} \rangle$ that is placed instead of $c_{2,1}$ in plan π . It results in a sequence of actions that is between the state satisfying condition eff_1 (i.e., when the robot appears in room 1) and the state satisfying eff_2 .

Using this domain and associated tree decomposition above we implemented our algorithm to see that the run time scaled linearly as the number of rooms increased. The results are compared in figure (7), which shows four different planning algorithm on this domain as the size of the domain grows. For all runs, the optimal-sized plans were found. In very small domains, there is some constant time overhead involved in the *PartPlan* algorithm, but asymptotically the *PartPlan* algorithms begin to dominate because of the linear scaling with respect to the number of rooms. Although this example is not rigorous enough to show the empirical dominance of the *PartPlan* algorithm for every domain, it does motivate further development of the ideas in real-world domains.

4 Related Work

Some of the techniques that we apply have been used before in automated reasoning. Best known are methods for reasoning with probabilistic graphical models [Pearl, 1988], logical theories [Dechter and Rish, 1994; Amir and McIlraith, 2000] and constraint satisfaction problems [Dechter and Pearl, 1989]. The common principle for those applications is the advantage taken of low *treewidth* available in many domains. With a tree decomposition of close-to-optimal width,

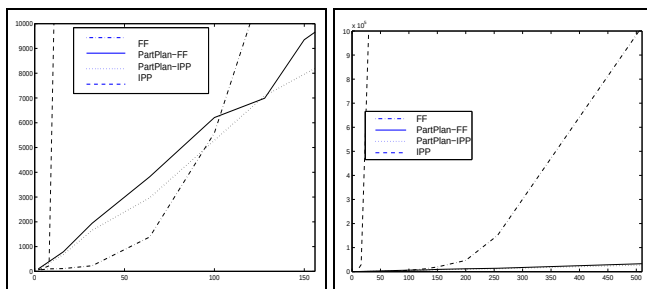


Figure 7: Comparison of run time for IPP, FF, PartPlan-IPP, and PartPlan-FF. The figure on the left is for 0-150 rooms, the figure on the right is for 0-500 rooms. The scale on the y-axis is in milliseconds.

these reasoning algorithms use two sweeps of the tree, an evidence collection and evidence distribution sweep, and the entire process is linear in the number of nodes.

Similar techniques to these have also been used in probabilistic planning to find *approximate* policies for factored MDPs (e.g., [Guestrin *et al.*, 2002]), and also in the context of multi-agent stochastic planning with MDPs. The main difference with this approach is that we find an *exact* solution, and we act in domains that do not have universal reachability (all states are reachable from all states), as is assumed in MDPs.

Section 1 discusses approaches for multi-agent planning [Lansky and Getoor, 1995; Ephrati and Rosenschein, 1994; Wolverson and desJardins, 1998].

5 Conclusions

Two contributions of our work are automated planning domain decomposition by minimizing subdomain interactions, and an algorithm to plan in this decomposed domain with no backtracking. The planning algorithm is sound and complete, and exploits a low treewidth partition for a particular planning domain, enabling planning to scale linearly with the domain size. Our automatic factoring algorithm for a planning domain attempts to optimize the run-time complexity of the planner. Theoretical results and empirical examples show that the planning algorithm scales well in domains with decomposable structure.

Directions for future work in this area include the development of algorithms for richer domains and description languages, such as nondeterministic planning actions, stochastic actions, and actions involving duration and numerical parameters. Our approach is built on the language of situation calculus (in which we developed our first version of the algorithms), leading us to believe that it will extend to these directions and more. Our work connects automated reasoning and planning, and we expect to extend this connection to allow first-order decomposition of planning and other related problems, such as diagnosis and reactive control.

Acknowledgments

The first author is supported under an ONR MURI Funds N00014-01-1-0890, and N00014-00-1-0637, and NSF grant

ECS-9873474. The second author is supported under an NSF Graduate Research Fellowship.

References

- [Amir and McIlraith, 2000] E. Amir and S. McIlraith. Partition-based logical reasoning. In *Proc. KR '00*, pages 389–400. MK, 2000.
- [Amir, 2001] E. Amir. Efficient approximation for triangulation of minimum treewidth. In *Proc. UAI '01*, pages 7–15. MK, 2001.
- [Amir, 2002] E. Amir. Projection in decomposed situation calculus. In *Proc. KR '02*, pages 315–326. MK, 2002.
- [Becker and Geiger, 1996] A. Becker and D. Geiger. A sufficiently fast algorithm for finding close to optimal junction trees. In *Proc. UAI '96*, pages 81–89. MK, 1996.
- [Cimatti and Roveri, 2000] A. Cimatti and M. Roveri. Conformant planning via symbolic model checking. *JAIR*, 13:305–338, 2000.
- [Dechter and Pearl, 1989] R. Dechter and J. Pearl. Tree clustering for constraint networks. *AIJ*, 38:353–366, 1989.
- [Dechter and Rish, 1994] R. Dechter and I. Rish. Directional resolution: The davis-putnam procedure, revisited. In *Proc. KR '94*, pages 134–145. Morgan Kaufmann, 1994.
- [Ephrati and Rosenschein, 1994] E. Ephrati and J. S. Rosenschein. Divide and conquer in multi-agent planning. In *Proc. AAAI '94*, pages 375–380. AAAI Press, 1994.
- [Erol *et al.*, 1994] K. Erol, J. Hendler, and D. S. Nau. HTN planning: Complexity and expressivity. In *Proc. AAAI '94*, pages 1123–1128. AAAI Press, 1994.
- [Fox and Long, 2002] M. Fox and D. Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains. <http://www.dur.ac.uk/d.p.long/IPC/pddl.html>, 2002.
- [Guestrin *et al.*, 2002] C. Guestrin, D. Koller, and R. Parr. Multiagent planning with factored mdps. In *Proc. NIPS'01*. MIT, 2002.
- [Hoffmann and Nebel, 2001] J. Hoffmann and B. Nebel. The FF planning system: Fast plan generation through heuristic search. *JAIR*, 14:253–302, 2001.
- [Kjaerulff, 1993] U. Kjaerulff. *Aspects of efficiency improvement in bayesian networks*. PhD thesis, Aalborg University, 1993.
- [Knoblock, 1990] Craig A. Knoblock. Learning abstraction hierarchies for problem solving. In *Proc. AAAI '90*, 1990.
- [Koehler and Hoffmann, 2000] J. Koehler and J. Hoffmann. On reasonable and forced goal orderings and their use in an agenda-driven planning algorithm. *JAIR*, 12:339–386, 2000.
- [Lansky and Getoor, 1995] A. L. Lansky and L. C. Getoor. Scope and abstraction: Two criteria for localized planning. In *Proc. IJCAI '95*, pages 1612–1618, 1995.
- [McIlraith and Amir, 2001] Sheila McIlraith and Eyal Amir. Theorem proving with structured theories. In *Proc. IJCAI '01*, pages 624–631. MK, 2001.
- [Pearl, 1988] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. MK, 1988.
- [Reiter, 2001] R. Reiter. *Knowledge In Action*. MIT, 2001.
- [Robertson and Seymour, 1986] N. Robertson and P. D. Seymour. Graph minors. II: algorithmic aspects of treewidth. *J. of Algorithms*, 7:309–322, 1986.
- [Wolverson and desJardins, 1998] M. Wolverson and M. desJardins. Controlling communication in distributed planning using irrelevance reasoning. In *Proc. AAAI '98*, pages 868–874, 1998.

A Parametric Hierarchical Planner for Experimenting Abstraction Techniques

Giuliano Armano, Giancarlo Cherchi, Eloisa Vargiu

Department of Electrical and Electronical Engineering

University of Cagliari

I-09123, Cagliari, Italy

{armano, cherchi, vargiu}@diee.unica.it

Abstract

This paper presents a parametric system, devised and implemented to perform hierarchical planning by delegating the actual search to an external planner (the “parameter”) at any level of abstraction, including the ground one. Aimed at giving a better insight of whether or not the exploitation of abstract spaces can be used for solving complex planning problems, comparisons have been made between instances of the hierarchical planner and their non hierarchical counterparts. To improve the significance of the results, three different planners have been selected and used while performing experiments. To facilitate the setting of experimental environments, a novel semi-automatic technique, used to generate abstraction hierarchies starting from ground-level domain descriptions, is also described.

1 Introduction

There is experimental evidence that humans repeatedly use abstractions while solving different kinds of problems [Stillings *et al.*, 1987], thus justifying the research in the field of automated hierarchical planning.

It is apparent that abstraction is usually not effective on simple problems, due to the overhead introduced by the need of going back and forth across abstract spaces while performing the search. In other words, enforcing abstraction on simple problems may end up by wasting computational resources. Yet, under certain assumptions abstraction can significantly reduce the search time when applied to complex problems [Knoblock, 1991].

In order to investigate the impact of abstraction mechanisms on the search complexity, we devised and implemented a hierarchical wrapper able to embody any domain-independent planner provided that a compliance with the STRIPS subset of PDDL 1.2 standard [McDermott *et al.*, 1998] is ensured. The embodied planner is exploited at any level of the hierarchy, each level being characterized by its own definitions. A suitable decoupling between levels is guaranteed by using domain-specific rules that establish the correspondence between

entities belonging to a level and its superior. Translation rules are given in a PDDL-like format, explicitly defined to support abstractions.

Experiments have been performed on some classical planning domains, widely acknowledged by the planning community –although not specifically tailored for abstraction. To better assess the significance of the results, three different planners have been used while performing experiments.

The remainder of this paper is organized as follows: first, some relevant work on planning by abstraction is briefly recalled, to give the reader a better insight of the issues deemed relevant. Then, the overall architecture of the system is illustrated, including the semi-automatic technique we developed for generating abstract spaces. Subsequently, experiments are described and results are discussed. Finally, conclusions are drawn and future work is outlined.

2 Related Work

Building an ordered set of abstractions for controlling the search has proven to be an effective approach for dealing with the complexity of planning tasks. This technique requires the original search space to be mapped into corresponding abstract spaces, in which irrelevant details are disregarded at different levels of granularity.

Two main abstraction mechanisms have been studied in the literature: action- and state-based. The former combines a group of actions to form macro-operators [Korf, 1987]. The latter exploits representations of the domain given at a lower level of detail; its most significant forms rely on (i) relaxed models, obtained by dropping operators’ applicability conditions [Sacerdoti, 1974], and on (ii) reduced models, obtained by completely removing certain conditions from the problem space [Knoblock, 1994]. Both models, while preserving the provability of plans that hold at the ground level, perform a “weakening” of the original problem space, thus suffering from the drawback of introducing “false” solutions at the abstract levels [Giunchiglia and Walsh, 1990].

As for Knoblock’s abstraction hierarchies, each predicate is associated with a unique level of abstraction –

according to the constraints imposed by the ordered monotonicity property [Knoblock, 1994]. Any such hierarchy can be obtained by progressively removing certain predicates from the domain (or problem) space.

From a general perspective, let us assume that abstractions might occur on types, predicates, and operators. Relaxed models are a typical example of predicate-based abstraction, whereas macro-operators are an example of operator-based abstraction.

3 System Architecture

The system has been called *HW[]*, standing for (parametric) *Hierarchical Wrapper*. Note that square brackets are part of the name, indicating the ability to embed an external planner; being *P* any such planner, the notation *HW[P]* shall be used to denote an instance of *HW[]* able to exploit the planning capabilities of *P*.

Figure 1 illustrates the architecture of the system, focusing on its main components, i.e., an engine and the embedded planner. The former is devoted to controlling the communication between adjacent levels, whereas the latter is exploited to perform planning at any given level of abstraction.

Any domain-independent planner can be embodied within the system, provided that compliance with the STRIPS subset of the PDDL 1.2 standard is ensured.

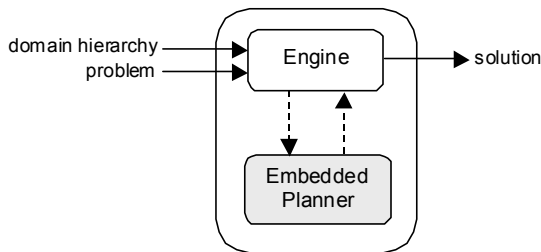


Figure 1. The architecture of the system.

Although the system supports a multiple-level hierarchy, for the sake of simplicity, in the following we assume that only one abstract level exists, giving rise to a two-level (i.e., ground and abstract) hierarchical description.

3.1 The Planning Algorithm

Once instantiated with an external planner *P*, *HW[P]* takes as inputs a ground-level problem and a structured description of the corresponding domain, including a set of rules to be used while mapping ground into abstract states and vice-versa. In fact, to perform planning at different levels of abstraction, the engine of *HW[]* must operate bi-directional translations (upwards and downwards) to permit communication between adjacent levels.

To find a solution of a given problem, first the engine of *HW[P]* translates the *init* and *goal* sections from the ground to the abstract level. *P* is then invoked to search for an abstract solution. Subsequently, each abstract operator is refined by repeatedly invoking *P*. The refinement of an abstract operator is performed by activating *P*,

at the ground level, on the goal obtained by translating downward its effects. Note that the initial state of each refinement depends on the previous refinement; hence, refinements must be performed according to the order specified by the abstract plan. To avoid incidental deletion of subgoals already attained during previous refinements, they are added to the list of subgoals that results from translating downward the effects of the current abstract operator to be refined.

When the attempt to refine the current abstract solution fails *P* is invoked to find the next abstract solution,¹ unless the number of abstract solutions found so far exceeds a given threshold (*m*). If no abstract solution could be successfully refined, to ensure the completeness of the algorithm, an overall search is performed at the ground level. The whole process ends when a ground solution is found or the overall search fails.

3.2 Extending PDDL for Dealing with Abstraction

A problem and its corresponding domain are described in accordance with the standard PDDL 1.2 syntax, using the “*define problem*” and “*define domain*” statements, respectively. To describe how bi-directional communication occurs between adjacent levels an extension to the standard PDDL has been devised and adopted.

More precisely, the syntactic construct “*define hierarchy*” has been introduced. It encapsulates an ordered set of domains, together with a corresponding set of mappings between adjacent levels of abstraction. Since the mappings are given in term of types, predicates and operators, three subfields have been introduced (i.e., *:types*, *:predicates*, and *:actions*), to represent the abstraction over such dimensions. The general form of the construct is:

```
(define (hierarchy <name>)
  (:domains <domain-name>*)
  (:mapping (<src-domain> <dst-domain>)
    [:types <types-def>]
    [:predicates <predicates-def>]
    [:actions <actions-def>]))
```

The following notation is adopted in the *:types* field to represent a clause for mapping types:

```
(abstract-type ground-type)
```

It specifies that *ground-type* becomes *abstract-type* while performing upward translations. To disregard a type, the following notation must be used:

¹ Due to the limitations of most of the existing planners, the process of incrementally querying for another solution may be simulated by preliminarily querying for *m* abstract solutions, to be released incrementally on demand.

Type	Node relationships	Supporting Evidence	Action
1		(i) if $a = c$ and $b = d$ there is no supporting evidence for assuming that A usually precedes B in a plan, and vice-versa;	remove both edges.
		(ii) if $a > c$ there is a high likelihood that A precedes B;	remove top edge.
		(iii) if $c > a$, there is a high likelihood that B precedes A.	remove bottom edge.
2		(i) if $a > c$ there is a high likelihood that A precedes B;	remove top edge.
		(ii) if $c > a$, there is a high likelihood that B precedes A.	remove bottom edge.
3		A (B) negates one or more preconditions required by B (A).	remove both edges.
4		B negates one or more preconditions required by A.	remove bottom edge.
5		B negates one or more preconditions required by A.	remove both edges.
6		A and B are usually complementary or loosely-coupled actions.	remove both edges.
7		A precedes B with high likelihood.	remove top edge.
8		A negates one or more preconditions required by B.	remove the top edge.

Table 1. Heuristics for pruning the operators' graph.

(nil ground-type)

Moreover, the following notation is adopted in the `:predicates` field to represent a clause for mapping predicates:

```
((abstract-predicate ?p11 ?p21 ...)
 (ground-predicate ?p12 - t12 ?p22 - t22 ...))
```

It specifies that `ground-predicate` must be preserved while going upward and vice-versa. Note that, if no differences exist in mapping a predicate between adjacent levels, the corresponding clause can be omitted.

To disregard a predicate while performing upward translations, the following notation is used:

```
(nil
 (ground-predicate ?p12 - t12 ?p22 - t22 ...))
```

It specifies that `ground-predicate` is not translated into any abstract-level predicate.

In addition, `abstract-predicate` can be expressed as a logical combination of some ground-level predicates.

To describe how to build the set of operators for the abstract domain, in the `:actions` field, four kinds of mapping can be expressed:

- an action remains unchanged or some of its parameters are disregarded;
- an action is removed;
- an action is expressed as a combination of actions belonging to the ground domain;
- a new operator is defined from scratch.

3.3 Generating Abstractions

To facilitate the setting of abstract spaces, as an alternative to the hand-coded approach used in [Armano *et al.*, 2003], a novel semi-automatic technique for generating abstraction hierarchies starting from ground-level domain descriptions has been devised and adopted.

From our particular perspective, performing abstraction basically involves executing two steps: (i) searching for macro-operator schemata through a priori or a posteriori analysis, (ii) selecting some of the schemata evidenced so far and translating them into abstract operators.

In this subsection, we concentrate on the task of finding macro-operator schemata throughout an a-priori analysis performed on the given domain and problem, rather than adopting the a-posteriori technique illustrated in [Armano and Vargiu, 2001], aimed at finding macro-operator schemata according to a post-mortem analysis performed on plan “chunks”.

Step (i) is performed by an algorithm for building and then pruning a directed graph, whose nodes represent operators and whose edges represent relations between effects of the source node and preconditions of the destination node. In particular, for each source node A and for each destination node B, representing operators defined in the given domain, the corresponding edge is labeled with a pair of non-negative numbers, denoted by $\langle a \ b \rangle$. The pair accounts for how many predicates A can establish (a) and negate (b) that are also preconditions of B. It is worth noting that source and destination node may coincide, thus giving rise to a self-reference.

Pruning is performed according to the domain-independent heuristics reported in Table 1. Note that the pruned graph does not contain edges labeled $\langle 0 \ 0 \rangle$, the corresponding operators being completely independent.

At this point, the most promising macro-operator schemata can be easily extracted from the pruned graph, each path being related with a candidate macro-operator.

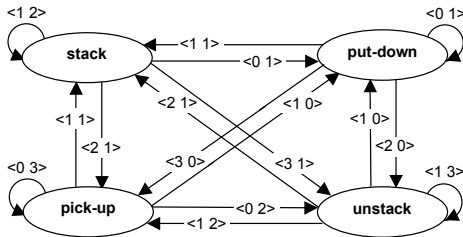


Figure 2. The directed graph (before pruning), representing static relations between operators of the blocks-world domain.

As an example, let us consider the well-known *blocks-world* domain, encompassing four operators: *stack*, *pick-up*, *unstack*, *put-down*. The corresponding graph is shown in Figure 2. Bearing in mind that the same mechanism has been applied to all operators' pairs, let us concentrate –for instance– on the relation that holds between *stack* (source node) and *pick-up* (destination node).

Considering that the effects of the *stack* operator are:

```
(not (holding ?x))
(not (clear ?y))
(clear ?x)
(handempty)
(on ?x ?y)
```

and that the preconditions of the *pick-up* operator are:

```
(clear ?x)
(ontable ?x)
(handempty)
```

we label the corresponding edge with the pair $\langle 2 \ 1 \rangle$. It is apparent that *stack* establishes two preconditions for *pick-up*, while negating another.

As for the pruning activity, Figure 3 shows the result-

ing graph for the blocks-world domain.² The resulting macro-operator schemata are (“;” being used for concatenation): *pick-up;stack*, *unstack;put-down*, *pick-up;put-down*, and *stack;unstack*.

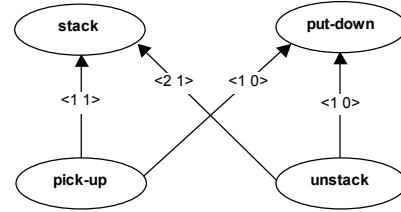


Figure 3. The directed graph (after pruning), representing static relations between operators of the blocks-world domain.

Step (ii) is performed by selecting a subset of the resulting macro-operator schemata, and by translating them into abstract operators. In principle, generating an abstract operator is not a deterministic task; for this reason in the current implementation of the system this mechanism has not yet been completely automated. Nevertheless, the simplest way of generating an abstract operator consists of deleting from the abstract level all predicates that do not occur among preconditions or effects of any selected macro-operator. This process influences (and is influenced by) the translation rules that apply to both types and predicates. For instance, the absence of a predicate as a precondition or effect of any induced abstract operator entails its deletion from the abstract level.

As for the blocks-world example, two macro-operator schemata have been disregarded (i.e., *stack;unstack* and *pick-up;put-down*), as they do not alter the state of the domain (the resulting set of effects being empty). In fact, it is apparent that they are composed of complementary actions.

It is worth pointing out that the approach described above can be used also for generating abstractions tailored to a given problem, by simply adding a dummy operator representing the goal(s) of the problem itself. This “goal” operator has only preconditions (its set of effects being empty), representing a logic conjunct of predicates that characterize the goal of the input problem. In this way, all sequences deemed relevant to solve the problem are easily put into evidence (as they end with the “goal” operator).

4 Experimental Results

The current prototype of the system has been implemented in C++. Experiments have been performed with three planners: GRAPHPLAN [Blum and Furst, 1997], BLACKBOX [Kautz and Selman, 1998], and LPG [Gervini and Serina, 2002]. In the following, GP, BB, and LPG shall be used to denote the GRAPHPLAN, BLACKBOX, and LPG algorithms, whereas *HW[GP]*,

² Since we are interested in finding macro-operators, we do not take into account self-references.

$HW[BB]$, and $HW[LPG]$ shall be used to denote their hierarchical counterparts.

To assess the capability of abstraction to improve the search, we performed some tests on five domains taken from the 1998, 2000, and 2002 AIPS planning competitions [Long, 1998; Bacchus, 2000; Long, 2002]: elevator, logistics, blocks-world, gripper, and zeno-travel. Experiments were conducted on a machine powered by an Intel Celeron CPU, working at 1200 Mhz and equipped with 256Mb of RAM. A time bound of 1000 CPU seconds has also been adopted.

#	GP	HW [GP]	BB	HW [BB]	LPG	HW [LPG]
elevator						
1-4	0.01	0.06	0.1	0.33	0.01	0.11
3-1	0.23	0.36	1.34	1.20	0.02	0.15
4-1	1.96	0.83	1.03	1.74	0.02	0.16
4-4	10.11	0.84	311.5	1.79	0.02	0.16
5-1	364.7	2.03	180.8	2.54	0.02	0.18
7-2	--	12.04	--	3.89	0.03	0.29
logistics						
4-2	0.68	1.22	0.27	0.46	17.93	--
5-2	0.08	0.16	0.15	0.46	0.02	--
7-0	--	10.93	4.49	2.17	2.12	--
8-1	--	16.26	2.90	3.02	1.55	--
10-0	--	43.43	8.27	3.76	2.17	--
15-0	--	203.4	10.91	6.33	0.15	--
blocks-world						
4-0	0.34	0.32	0.16	0.67	0.02	0.08
6-0	3.04	1.82	0.26	1.68	0.05	0.23
8-0	31.61	11.13	0.92	2.46	0.36	0.31
10-0	--	--	6.82	5.00	0.62	0.67
11-0	--	--	16.23	4.25	4.23	0.83
14-0	--	--	--	9.84	5.00	1.91
15-0	--	--	--	--	7.49	2.07
17-0	--	--	--	--	33.93	3.49
20-0	--	--	--	--	66.78	7.88
22-0	--	--	--	--	183.16	12.21
25-0	--	--	--	--	668.98	24.94
zeno-travel						
1	0.02	0.52	0.22	0.36	0.02	0.03
8	--	42.55	0.94	2.36	0.14	0.49
9	--	--	0.34	3.37	0.13	1.08
11	--	--	11.20	2.78	0.16	1.06
13	--	--	62.99	20.52	0.42	2.47
14	--	--	--	20.04	3.90	21.93
gripper						
2	4.72	0.56	0.42	0.63	0.02	0.07
3	7.91	1.73	5.22	1.20	0.02	0.12
4	18.32	2.63	268.7	1.55	0.02	0.14
5	57.21	4.38	421.1	1.54	0.03	0.15
6	--	7.97	586.4	2.26	0.03	0.17
9	--	24.29	--	3.63	0.05	0.36

Table 2. Performance comparison of BB , GP , and LPG with their hierarchical counterparts.

All domains have been structured according to a ground and an abstract level, the latter having been generated following the approach described in the previous subsection. For each domain, several tests have been performed –characterized by increasing complexity. Table 2

compares the CPU time of each planner over the set of problems taken from the AIPS planning competitions. Dashes show problem instances that could not be solved by the corresponding system within the adopted time-bound.

Elevator. Experiments show that –for GP and BB – the CPU time increases very rapidly while trying to solve problems of increasing length, whereas $HW[GP]$ and $HW[BB]$ keep solving problems with greater regularity (although the relation between number of steps and CPU time remains exponential). LPG is able to solve long plans in a very short time, thus doing away with the need to resort to $HW[LPG]$.

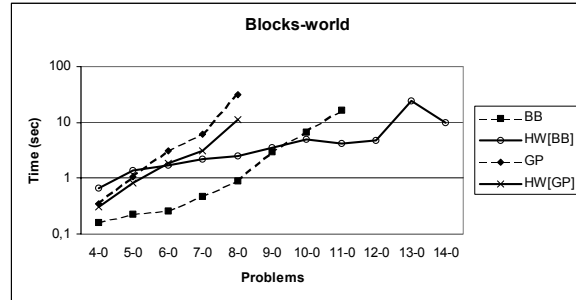


Figure 4. CPU time comparisons in the blocks-world domain.

Logistics. In this domain GP easily solves problems up to a certain length, but it is unable to solve problems within the imposed time limits if a given threshold is exceeded. On the other hand, $HW[GP]$ succeeds in solving problems of increasing length without encountering the above difficulties. BB performs better than $HW[BB]$ for small problems, whereas $HW[BB]$ outperforms BB on more complex problems. LPG is able to solve long plans in a few seconds at the most. For unknown reasons LPG was not able to refine any abstract operator when invoked by the engine of $HW[LPG]$.

Blocks-world. Tests performed on this domain reveal a similar trend for GP and $HW[GP]$, although the latter performs slightly better than the former. BB performs better than $HW[BB]$ for simple problems, whereas $HW[BB]$ outperforms BB on problems of medium complexity. LPG is able to solve problems whose solution length is limited to 100 steps. In this domain, $HW[LPG]$ clearly outperforms LPG on more complex problems.

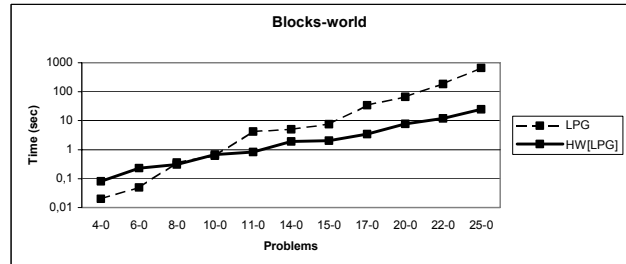


Figure 5. CPU time comparisons in the blocks-world domain.

Zeno-travel. Unfortunately, in this domain, neither GP nor $HW[GP]$ are able to successfully tackle most of the problems of this domain. An improvement of $HW[BB]$

over BB can be observed, similar to the one shown for the blocks-world domain. LPG is able to solve long plans in a few seconds at the most, thus avoiding the need to resort to *HW[LPG]*.

Gripper. For the gripper domain, both *HW[GP]* and *HW[BB]* clearly outperform their non-hierarchical counterparts. LPG is able to solve long plans in a very short time.

For the sake of brevity, only two plots (i.e., Figures 4-5 concerning the blocks-world domain) of relative performances –i.e. non hierarchical vs. hierarchical– are reported.

5 Conclusions and Future Work

In this paper a parametric system has been presented, devised to perform hierarchical planning by delegating the actual search to an external planner (the parameter). Aimed at giving a better insight of whether or not the exploitation of abstract spaces can be useful for solving complex planning problems, comparisons have been made between any instances of the hierarchical planner and its non-hierarchical counterpart.

To better investigate the significance of the results, three different planners have been used to make experiments. To facilitate the setting of experiments, a novel semi-automatic technique for generating abstract spaces has been devised and adopted. Experimental results highlight that abstraction is useful for classical planners, such as GP and BB. On the contrary, the usefulness of resorting to hierarchical planning for the latest-generation planner used for experiments (i.e., LPG) clearly emerges only in the blocks-world domain.

As for future work, we are currently addressing the problem of automatically generating abstract operators.

References

- [Armano and Vargiu, 2001] Giuliano Armano and Eloisa Vargiu. An Adaptive Approach for Planning in Dynamic Environments. *Proceedings of the International Conference on Artificial Intelligence (IC-AI 2001), Special Session on Learning and Adapting in AI Planning*, pages 987--993, Las Vegas, Nevada, June 2001.
- [Armano *et al.*, 2003] Giuliano Armano, Giancarlo Cherchi, and Eloisa Vargiu. Experimenting the Performance of Abstraction Mechanisms through a Parametric Hierarchical Planner. *Proceedings of IASTED International Conference on Artificial Intelligence and Applications (AIA'2003)*, Innsbruck, Austria, Febbraio 2003.
- [Bacchus, 2000] Fahiem Bacchus. Results of the AIPS 2000 Planning Competition, 2000.
URL: <http://www.cs.toronto.edu/aips2000>.
- [Blum and Furst, 1997] Avrim L. Blum and Marrick L. Furst. Fast Planning through Planning Graph Analysis. *Artificial Intelligence*, 90(1-2):279--298, 1997.
- [Gerevini and Serina, 2002] Alfonso Gerevini and Ivan Serina. LPG: A Planner Based on Local Search for Planning Graphs. *Proceedings of the 6th International Conference on AI Planning and Scheduling*, AAAI Press, Menlo Park, 2000.
- [Giunchiglia and Walsh, 1990] Fausto Giunchiglia and Toby Walsh. A theory of Abstraction, Technical Report 9001-14, IRST, Trento, Italy, 1990.
- [Kautz and Selman, 1998] Henry Kautz and Bart Selman. BLACKBOX: A New Approach to the Application of Theorem Proving to Problem Solving. In *Working notes of the Workshop on Planning as Combinatorial Search, AIPS-98*, pages 58--60, Pittsburg, PA, 1998.
- [Korf, 1987] Rich E. Korf. Planning as Search: A Quantitative Approach. *Artificial Intelligence*, 33(1):65--88, 1987.
- [Knoblock, 1991] Craig A. Knoblock. Search Reduction in Hierarchical Problem Solving. *Proceedings of the 9th National Conference on Artificial Intelligence*, pages 686--691, Anaheim, CA, 1991.
- [Knoblock, 1994] Craig A. Knoblock. Automatically Generating Abstractions for Planning. *Artificial Intelligence*, 68(2):243--302, 1994.
- [Long, 2002] Derek Long. Results of the AIPS 2002 Planning Competition, 2000.
URL: <http://www.dur.ac.uk/d.p.long/competition.html>.
- [Long, 1998] Derek Long. The AIPS-98 Planning Competition. *AI Magazine*, 21(2):13--33, 1998.
- [McDermott *et al.*, 1998] Drew McDermott, Malik Ghalilab, Adele Howe, Craig Knoblock, Anshwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. PDDL – The Planning Domain Definition Language, Technical Report CVC TR-98-003 / DCS TR-1165, Yale Center for Communicational Vision and Control, October 1998.
- [Sacerdoti, 1974] Earl D. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5:115--135, 1974.
- [Stillings *et al.*, 1987] Neil A. Stillings, Christopher H. Chase, Mark H. Feinstein, Jay L. Garfield, Edwina L. Rissland, and Steven E. Weisler. *Cognitive Science: An Introduction*. MIT Press, Cambridge, Massachusetts, 1987.

On the application of least-commitment and heuristic search in temporal planning

Antonio Garrido and Eva Onaindia

Dpto. Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
Camino de Vera s/n, 46071 Valencia, Spain
{agarridot,onaindia}@dsic.upv.es

Abstract

Graphplan planning graphs are structures widely used in modern planners. The exclusion relations calculated in the planning graph extension provide very useful information, especially in temporal planning where actions have different duration. However, Graphplan backward search has some inefficiencies that impose limitations when dealing with large temporal problems. This paper presents a new search process for temporal planning to avoid these inefficiencies. This search uses the information of a planning graph and shows beneficial in the scalability of the planner. Moreover, our experiments show that a planner with this new search is competitive with other state-of-the-art planners *w.r.t.* the plan quality.

1 Introduction

Many of the current challenges in AI planning focus on increasing the functionalities of planners to deal with more real features, such as temporal capabilities, explicit management of resources, more expressive domain definition languages, heuristic techniques and optimisation criteria, etc. [Fox and Long, 2001; Gerevini and Serina, 2002; Smith and Weld, 1999]. This paper deals with three of the previous functionalities: i) planning with temporal features (actions with duration), ii) more expressive domain definition languages (PDDL2.1 [Fox and Long, 2001]), and iii) plan optimisation (makespan). Traditional temporal planners have adopted a conservative model of actions, where two actions cannot overlap in *any* way if they have conflicting preconditions or effects. This model is adequate in some planning domains, but there exist others that require a richer model of actions. Level 3 of PDDL2.1 used in IPC-2002 provides a model of durative actions which allows a more accurate exploitation of action concurrency to obtain shorter makespan plans.

This paper describes our experiences with a Temporal Planning SYStem (from now on TPSYS), based on Graphplan [Blum and Furst, 1997] and TGP [Smith and Weld, 1999], to manage the model of durative actions proposed in level 3 of PDDL2.1. TPSYS performs a Graphplan backward search, which guarantees the properties of completeness

and optimality. However, backward search has some inefficiencies that impose important limitations in large temporal problems. In these problems, the search space is vastly increased and the performance of the algorithm degrades. We suggest a new two-stage search process to overcome these inefficiencies. First, a backward search generates an initial relaxed plan. Next, this relaxed plan is used as an outline for generating a solution plan by means of a non-complete heuristic process, where actions are only definitively allocated in the plan when they are applicable and no mutex (least-commitment). This allows to increase the scalability of search, producing non-optimal, but good quality, plans.

2 A Review of TPSYS

TPSYS is based on a three-stage process [Garrido *et al.*, 2002], which combines the ideas of Graphplan and TGP. This means that TPSYS incrementally extends a temporal planning graph, performs a backward search through that graph and extracts a plan.

Unlike *conservative* actions, durative actions present more conditions to be guaranteed for the success of the action. These conditions are $SCond_a$, $ECond_a$ and Inv_a with the conditions of a to be guaranteed at the *start*, *end* and *over all* the execution of a , respectively. Durative actions have two types of effects: $SEff_a$ and $EEff_a$ with the effects to be asserted at the *start* and *end* of a , respectively.

The first stage of TPSYS calculates the action-action and proposition-action static mutex relationships. These mutex relationships are static because they only depend on the definition of the actions and they always hold. The second stage extends a temporal planning graph which alternates temporal levels of propositions ($P_{[t]}$) and actions ($A_{[t]}$). Unlike Graphplan, levels in TPSYS represent instants of time $t \in \mathbb{R}^+$ in which propositions are present and actions can start/end. Action-action, proposition-action and proposition-proposition mutex relationships are calculated during the extension of the temporal planning graph. The third stage performs the search in a Graphplan backward way, extracting an optimal plan through the planning graph. Since the third stage starts as soon as all the propositions in the final situation are present, non pairwise mutex, and the plan extraction is complete, TPSYS obtains the plan of optimal makespan.

Backward search in TPSYS preserves the same properties of completeness and optimality of Graphplan, but it en-

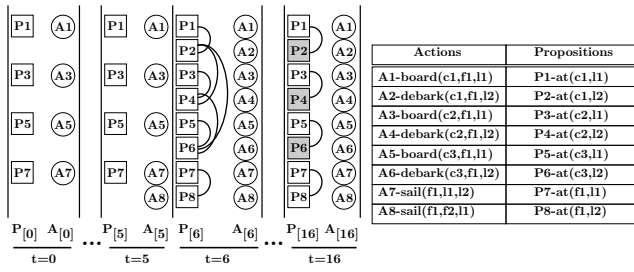


Figure 1: Outline of the temporal planning graph for the ferry problem. Shaded propositions represent the non-pairwise mutex problem goals at time 16. Mutex relations between propositions are represented by thick lines.

tails the most time consuming stage. Furthermore, this search presents some inefficiencies (inherited from Graphplan) which impose important limitations in temporal problems. Let us consider a simple problem from the ferry domain. The domain consists of transporting a number of cars from one location to another using a ferry which can carry only one car at a time. To keep the problem simple enough we assume three cars $c1$, $c2$ and $c3$ to be transported from location $l1$ to $l2$ by ferry $f1$. The actions are $board(?car, ?ferry, ?locat)$, $sail(?ferry, ?locat1, ?locat2)$ and $debark(?car, ?ferry, ?locat)$, with durations 1, 5 and 2, respectively. The optimal plan contains 11 actions (3xboard, 5xsail and 3xdebark) and the makespan is 34. The mutex relations are binary, so the second stage ends at time 16 and the search starts from there (see Figure 1). Since every pair of actions is mutex, only one action is planned in each level. If we assume that applicable actions are selected in each level from top to bottom in the planning graph of Figure 1, action A2 is firstly planned at time 14. Next, actions A7 and A1 are planned at times 9 and 8, and so on. Because no feasible plan is found, the search backs to time 14 where all permutations of actions A2, A4, A6 are planned under the same schema with no success. This is the first indication of inefficiency: a lot of effort is wasted trying, unsuccessfully, to plan nearly identical actions in problems with symmetry. As no plan is found from time 16, the planning graph is extended to time 17 and the search is re-started from scratch. This entails the second indication of inefficiency: no actions planned in previous stages of search are reused as a part of the current plan, committing similar failures in the new *one-deeper-level* search space. Although memoization helps reduce the amount of failures committed during search, the way in which it prunes one branch of search does not allow to solve the real conflict until exhausting the whole level. This is the third indication of inefficiency: if one proposition becomes unsupported, no new actions supporting it are studied but that branch is discarded and the algorithm performs a backtracking stage.

These inefficiencies have a negative influence in Graphplan-based temporal approaches. While a classical Graphplan's planning graph for this problem has 11 levels, in temporal planning no plan is found until level 34 is extended and explored. Particularly, in TPSYS the number of levels generated depends on the *dispersion* of the durations of the actions.

The worst performance happens when the gcd of the durations is 1, forcing the algorithm to generate the maximum number of levels. In consequence, the previous inefficiencies and wasted search are repeated more frequently.

3 Combining Least-Commitment and Heuristics to Improve the Search Process

In this section we substitute the backward search by a two-stage search to avoid the previous inefficiencies in a temporal planning approach. First, a backward chaining stage generates an initial relaxed plan from the information of the temporal planning graph. Second, a forward chaining stage allocates the execution time of the actions in the relaxed plan.

3.1 Generation of an Initial Relaxed Plan

This stage generates an initial relaxed plan from the information of the temporal planning graph (from now on *TG*) to be used as a *skeleton* of the final plan. We define a relaxed plan Π as a partially ordered set of actions in which both the problem goals and action conditions hold. It is called *relaxed* because neither mutex relationships between actions nor commitment on their start time are considered.

A relaxed plan always contains two fictitious actions with no duration called IS and FS. IS achieves the propositions of the initial situation, whereas FS requires the problem goals. Π is generated similarly to the relaxed solution plan in the FF planner [Hoffmann, 2000] with the exception that we handle durative actions (see Algorithm 1).

- 1: $goals \leftarrow$ problem goals {*must hold in any plan for the problem*}
- 2: $\Pi \leftarrow \{IS \cup FS\}$ {*obligatory actions*}
- 3: **while** $goals \neq \emptyset$ **do**
- 4: extract g_i from goals
- 5: **if** g_i is not supported in Π **then**
- 6: $a \leftarrow \arg \min(\text{number of mutex which } a_i \text{ imposes in } \Pi)$
 $\forall a_i \text{ which supports } g_i$
- 7: **if** a is the only action which supports $g_i \wedge g_i$ is a condition of an obligatory action **then**
- 8: mark a as obligatory in Π
- 9: $\Pi \leftarrow \Pi \cup \{a\}$ {*no commitment on start time of a yet*}
- 10: $goals \leftarrow goals \cup \{SCond_a \cup Inv_a \cup ECond_a\}$

Algorithm 1: Generation of an initial relaxed plan Π .

Step 6 studies the actions which support g_i and selects the action which minimises the number of mutex in Π . In problems with multiple resources, this selection tends to use as many resources as available. For instance, in a ferry problem with two or more ferries $f1, f2, \dots$, step 6 distributes the use of the ferries in a homogeneous way. If action $debark(c1, f1, l2)$ is used for debarking the car $c1$, action $debark(c2, f2, l2)$ is first selected when debarking the car $c2$ because $debark(c2, f1, l2)$ is mutex with $debark(c1, f1, l2)$ and imposes more mutex in Π . This selection provides the relaxed plan more information about the structure of the problem, thus increasing the quality (number of actions in parallel) of the final plan. Note that in step 6 we do not need to perform real search (or backtracking) because none of the mutexes are considered and, therefore, this strategy always leads to a solution. Moreover, we introduce the term of obligatory action

a as the only action which supports a goal which must hold in any plan for the problem. This is helpful because it means that a must be always present in Π (steps 7–8). Obviously, both IS and FS are obligatory.

One important property of Π is that if there is no mutex between overlapping actions, all actions in Π form a feasible, optimal plan. The proof of this is straightforward and relies on the complete extension of the temporal graph. The level in which the temporal graph extension ends indicates the minimal time in which the goals can be achieved by non-pairwise mutex actions. Thus, this level provides a minimal bound of the makespan for a feasible plan and, if no mutex between actions holds, the plan is not only feasible but also optimal. Unfortunately, this is not a very common situation and mutex relations break the plan *relaxation*. This entails to postpone the allocation of actions, and/or to plan new actions to solve the unsupported (sub)goals.

3.2 Planning and Allocating of Actions

This stage performs the allocation in time of actions in the relaxed plan. We use a structure called `set_of_plans`, with the search space formed by all the generated plans $\{\Pi_i\}$. Actions in each Π_i are divided into two disjunctive sets: $Alloc_i$ and $Relax_i$. $Alloc_i$ contains the actions which have been allocated in time and will never be removed from Π_i . $Relax_i$ contains the actions which have not been allocated yet, and so they can be removed from Π_i . Initially, $Alloc_i$ is empty and $Relax_i$ contains all the actions in Π_i (the initial Π_i is the relaxed plan computed in the previous section). This stage finishes once $Relax_i$ gets empty, obtaining in $Alloc_i$ the actions of the plan which support all the problem goals.

```

1: set_of_plans  $\leftarrow \Pi$ , generated in Algorithm 1
2: while set_of_plans  $\neq \emptyset$  do
3:   extract the lowest cost  $\Pi_i$  from set_of_plans
4:   if  $Alloc_i$  supports all the problem goals then
5:     exit with success
6:   else
7:      $a \leftarrow \arg \max(\text{allocation priority})$ 
        $\forall a_j \in Relax_i$  which can start at  $\text{time\_of\_execution}_i$ 
8:     if  $a$  is mutex in  $Alloc_i$  then
9:       if  $a$  is not obligatory then
10:        remove  $a$  from  $Relax_i$ 
11:      else
12:        postpone start time of  $a$  in  $Relax_i$ 
13:    else
14:      if  $a$  is applicable then
15:        allocate  $a$  in  $Alloc_i$  at  $\text{time\_of\_execution}_i$ 
16:      else
17:        insert new plans into set_of_plans to make  $a$  applicable
18:    update  $\text{time\_of\_execution}_i$ 

```

Algorithm 2: Planning and allocating of actions.

The idea is to move forward in time, simulating the real execution of Π_i , progressively taking care of the actions which can start their execution (see Algorithm 2). The current time of execution in Π_i , $\text{time_of_execution}_i$, is initialised to 0. The algorithm always selects the plan Π_i of lowest cost from `set_of_plans` (step 3). If all the problem goals are

supported the algorithm exits with success (step 5). Otherwise, actions from $Relax_i$ which can start at the current $\text{time_of_execution}_i$ are tried to be allocated (step 7). If one action a is mutex with actions in $Alloc_i$ and non-obligatory, a is removed from Π_i (step 10), delaying the fulfillment of its goals to a future time of execution. The reason to remove a non-obligatory action is that it could be a bad choice for the plan. If a is obligatory it is not removed but its start time is postponed (step 12). If a is not mutex and applicable, a is allocated in time (step 15). This step entails the first indication of loss of completeness. Although there exist alternative actions to a , if a can be allocated those actions are not considered in Π_i . Step 17 is a branching point in which new actions are inserted (generating new plans) to achieve unsupported conditions of a . For each action a_j supporting each condition a new plan Π_j is generated and inserted into `set_of_plans` with a_j marked as obligatory in Π_j . It is important to note that a_j is not allocated in time, but it is inserted with its earliest start time of execution extracted from the TG . This is part of the least-commitment technique performed in the allocation of actions when they are inserted into plans. Step 18 moves to the next relevant $\text{time_of_execution}_i$ in which actions can start, extending the TG if necessary.

The algorithm has two important points of selection: steps 3 and 7. Step 3 selects the plan Π_i with the lowest cost from `set_of_plans`, where the cost is estimated as follows:

$$\begin{aligned}
\text{cost}(\Pi_i) &= \text{cost}(Alloc_i) + \text{cost}(Relax_i), \text{ where} \\
\text{cost}(Alloc_i) &= \alpha \cdot \text{unsup}(\text{FS}, Alloc_i) + \beta \cdot \text{duration}(\Pi_i) \\
\text{cost}(Relax_i) &= \sum_{\forall a \in Relax_i} \gamma \cdot \text{unsup}(a, Alloc_i) + \\
&\quad \delta \cdot \text{add}(a, \text{FS}) + \theta \cdot \text{del}(a, \text{FS}) + \\
&\quad \lambda \cdot \text{PA_mutex}(a, \text{FS}) + \mu \cdot \text{duration}(a)
\end{aligned}$$

The cost of a plan Π_i consists of the sum of the cost due to $Alloc_i$ and $Relax_i$. $\text{unsup}(a, Alloc_i)$ is an estimation of the number of actions necessary to solve the unsupported conditions of a from the current $\text{time_of_execution}_i$. It is estimated through the TG , ignoring the delete effects of actions (as in the heuristic used in FF). $\text{add}(a, \text{FS})$ is the number of conditions of FS which a supports, whereas $\text{del}(a, \text{FS})$ is the number of conditions of FS which a deletes. $\text{PA_mutex}(a, \text{FS})$ is the number of mutex between conditions of FS and action a . This mutex information is extracted from the TG and indicates the impossibility to have simultaneously the conditions of FS and a . duration represents the duration of the plan/action to take into account the makespan of Π_i . Note that $\alpha, \beta, \gamma, \theta, \lambda, \mu \geq 0$ because they have a positive impact in the cost of the plan. In opposition, $\delta \leq 0$ because it has a negative impact in the cost.

In step 7, the action a with the maximal allocation priority is selected to be studied at $\text{time_of_execution}_i$. This priority is estimated as follows:

$$\begin{aligned}
\text{prio}(a) &= \rho \cdot \text{unsup}(a, Alloc_i) + \sigma \cdot \text{succ}(a, Relax_i) + \\
&\quad \tau \cdot \text{meetsucc}(a, Relax_i) + \psi \cdot \text{duration}(a)
\end{aligned}$$

The allocation priority of action a depends on several local factors. $\text{succ}(a, Relax_i)$ is the number of direct successor

(dependent) actions of a in $Relax_i$. $meetsucc(a, Relax_i)$ is the number of direct successor actions of a which can start as soon as a ends, i.e. which meet a . These two values indicate the importance of an action for the successor actions in $Relax_i$. Intuitively, the more successor actions are directly supported by a , the more important a is in the plan. Similarly, the *meeting* successor actions indicate the number of successor actions which can be immediately executed without mutex. *unsup* and *duration* are defined as above. Coefficients $\sigma, \tau \geq 0$ because they imply to select an appropriate action to be allocated. However, $\rho, \psi \leq 0$ because they indicate that action a is not promising enough to be allocated yet.

The previous evaluation functions can have many more heuristic factors, but according to our experimental analysis they are general enough for most temporal planning problems. We can also implement different heuristic methods by setting the values of the coefficients, which in our case are the same for all the domains. Although the values of the coefficients can influence the search, their precise value is not as relevant as in other heuristic approaches based on local search as LPG [Gerevini and Serina, 2002]. For instance, the static cost of inserting an action in LPG is based on the number of unsupported conditions, what cannot represent the real complexity of solving each condition. On the contrary, the coefficient *unsup* deals with that complexity because it estimates the real cost to support it in the *TG*.

It is important to note that plans are incrementally generated without discarding allocated actions and with no redundancy due to symmetry. Although Algorithm 2 explores the complete space of actions to make actions applicable, the allocation priority imposes an order of execution and discards the rest of feasible orderings (second indication of loss of completeness). For instance, in the *ferry* problem of section 2, when studying the actions to *debark* $\{A2, A4, A6\}$ the algorithm allocates one action and postpones the others. This avoids the complete exploration of all the permutations of $A2, A4$ and $A6$, preventing the planner from the generation of symmetric plans.

4 Experimental Results

We have implemented the previous search on top of TPSYS, conducting several experiments¹. The two first experiments compare the new search vs. the Graphplan backward search. In the **first experiment**, we studied the impact of the new search in problems with a high degree of symmetry, such as the *ferry* and *gripper*, focussing on the planner scalability. Figures 2-a,b show the results for some problems of these domains. Although it is not surprising that the least-commitment search (tpsys-LC) is faster than Graphplan backward search (tpsys-GP), it is worthy to mention that tpsys-LC scales up much better than tpsys-GP, especially in the *gripper* domain. Moreover, tpsys-LC found optimal plans without backtracking for all the problems.

The **second experiment** deals with some problems of the simple-time track of the last IPC-2002². Although both ap-

proaches have difficulties to solve all the problems (see Figures 2-c,d,e,f), the least-commitment search can solve more problems. On one hand, backward search difficulties rise as a result of the redundancy in the complete, blind search process. On the other hand, least-commitment search difficulties rise as a result of: i) the heuristic, *greedy* approach, which can lead to the wrong path in the search, and ii) the non-complete preserving search. The *zenotravel* and *satellite* domains show the highest speedups. It is important to note that the best improvement is produced in the *satellite* domain, which is the only domain in the competition that exploits the end conditions of durative actions.

The third and fourth experiments are aimed at evaluating the quality of the plans generated by the new approach. We are mainly interested in the makespan of the plans, but we also consider the number of actions as an additional indication of the plan quality. The **third experiment** uses the plans generated in the experiment 1 (Figures 2-a,b) and compares them with the plans generated by LPG³ [Gerevini and Serina, 2002] and Mips⁴ [Edelkamp, 2002]. We have chosen these two planners as they handle temporal features and showed distinguished performance in the last IPC. To date our interest has not focussed on code optimisation, so in this experiment we will not compare the execution time of the planners⁵ but only their plan quality. Table 1 shows the comparison between the three planners taking into account the makespan and the number of actions of each plan. We have not included the results for the *ferry* domain because the three planners generate the same fully sequential plans. However, plans for the *gripper* domain can have actions in parallel (the *gripper* is a resource with capacity for two balls) so that the planner can exploit a better concurrency. In order to simplify the resulting plans the actions have been assigned duration 1. Hence, the optimal makespan is $2n - 1$ and the number of actions in an optimal plan is $3n - 1$, where n is the number of balls in the problem. Surprisingly, both LPG or Mips generate sequential plans as can be easily noticed from the fact that the makespan comes directly from the number of actions in the plan. tpsys-LC is the only of the three planners which generates optimal plans for all the problems, highly exploiting the action parallelism.

In the **fourth experiment**, we have used the plans generated in the last IPC to perform a comparison between tpsys-LC and some of the state-of-the-art planners which participated in the simple-time track. We have analysed

<http://www.dur.ac.uk/d.p.long/IPC>

³LPG is based on a non-deterministic local search and, consequently, it would not be fair to work with the plan from only one execution. In our experiments with LPG, we have run each problem ten times and extracted the average values. We have used the LPG1.0 version as provided in: <http://prometeo.ing.unibs.it/lpg>

⁴We have used the Mips version as provided in: <http://www.informatik.uni-freiburg.de/~mmps>

⁵LPG and Mips are clearly faster than tpsys-LC. For instance, they get to solve some simple problems before tpsys-LC finishes its 1st and 2nd stages. Currently, these stages represent an important bottleneck in our implementation and we think they could be drastically reduced.

¹The experiments were run on a Pentium IV 2 GHz with 512 Mb.

²More information on the domains, problems and results of the international planning competition (IPC-2002) in:

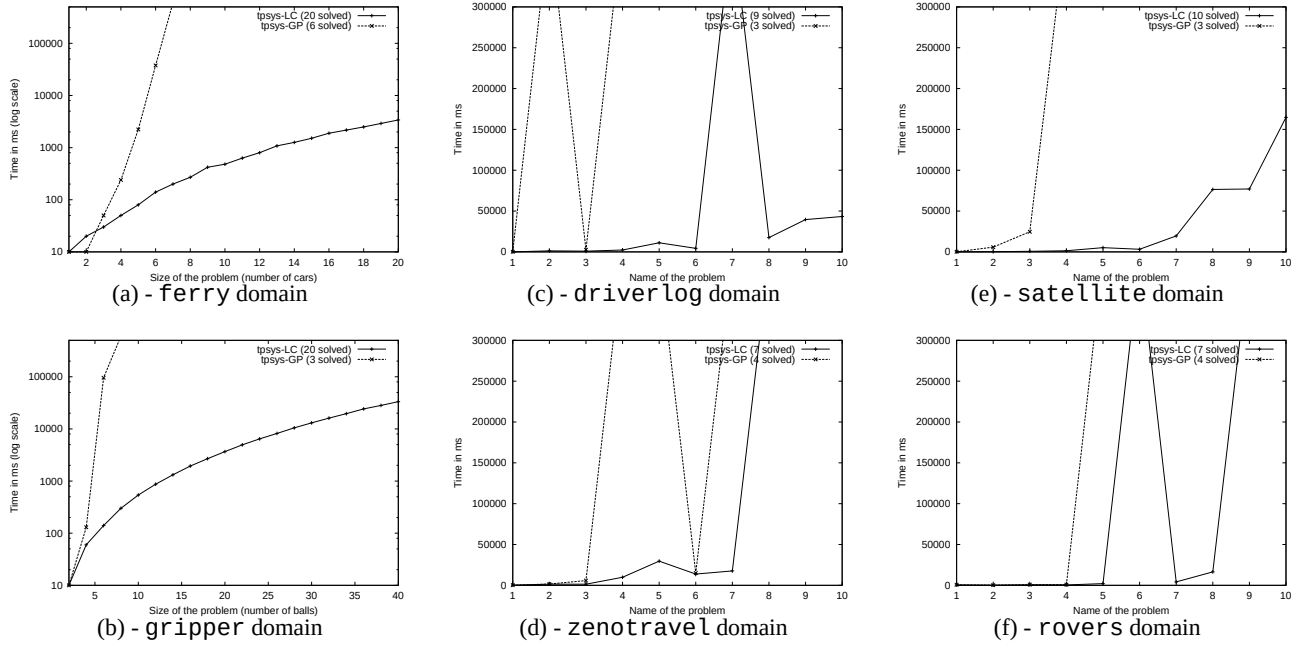


Figure 2: Comparison of the least-commitment search vs. the Graphplan backward search. Tests were censored after 300 s.

Problem	tpsys-LC	LPG	Mips
gripper-4	7 (11)	13 (13)	15 (15)
gripper-8	15 (23)	25 (25)	31 (31)
gripper-12	23 (35)	45 (45)	47 (47)
gripper-16	31 (47)	53 (53)	63 (63)
gripper-20	39 (59)	63 (63)	79 (79)
gripper-24	47 (71)	77 (77)	95 (95)
gripper-28	55 (83)	89 (89)	111 (111)
gripper-32	63 (95)	105 (105)	127 (127)
gripper-36	71 (107)	117 (117)	143 (143)
gripper-40	79 (119)	127 (127)	159 (159)

Table 1: Comparison of the makespan (number of actions are in brackets) of the plans for the gripper domain obtained by tpsys-LC, LPG and Mips planners.

the plans generated by the domain-independent (*di*) planners LPG, Mips and VHPOP, and the domain-dependent (*dd*) planners SHOP2, TALPlanner and TLPlan. For LPG and Mips we have run the problems again and for the rest of the planners we have used the results of the competition. Although Sapa, TP4 and IxTeT also participated in the competition we have not found enough results for the simple-time track to be considered in our comparison. Figure 3 shows the results of this comparison (for lack of space we only include the domains with more problems solved). In the *driver-log* domain, the plans of tpsys-LC are generally longer than the rest of planners with the exception of LPG and SHOP2. In the *satellite* domain, tpsys-LC behaves in average better than the rest of the planners, with the only exception of TLPlan. In the *rovers* domain, tpsys-LC generates again plans of very good quality, which are only

improved by SHOP2 and TLPlan. Further, we have noticed that tpsys-LC generates plans with nearly the same (or even fewer) actions than the rest of the planners. These experimental results show that the new approach significantly improves the scalability of the Graphplan backward search.

5 Conclusions through Related Work

Least-commitment techniques have been widely used in planning, relying on the consistency of temporal constraints (IxTeT and HSTS), postponing the assignment of values to variables or the order of execution of actions, etc. [Weld, 1994]. We use a least-commitment approach to overcome the limitations of the Graphplan backward search detected in previous works [Fox and Long, 1999; Zimmerman and Kambhampati, 1999]. Our approach basically postpones the allocation in time of actions until they become not mutex and applicable. Thus, the algorithm generates a relaxed plan similarly to FF, to be used as a *skeleton* of the plan. Next, it allocates actions in time according to their mutex relations and to several local heuristic criteria in the line of LPG. Our critical difference with LPG relies on several points. First, the planning graph is temporal and moves chronologically in time instead of planning steps. Second, the *TG* is not only used to extract heuristic information but also to generate a relaxed plan. Third, this plan is repaired in a forward chaining direction, and unlike LPG the allocated actions are never removed. Fourth, the heuristics exploit better the structure of the plan and the real importance of each action in the plan. Finally, tpsys-LC uses a more precise model of action mutex which implies fewer constraints on the execution of actions and a larger search space.

This paper contributes in the way in which least-

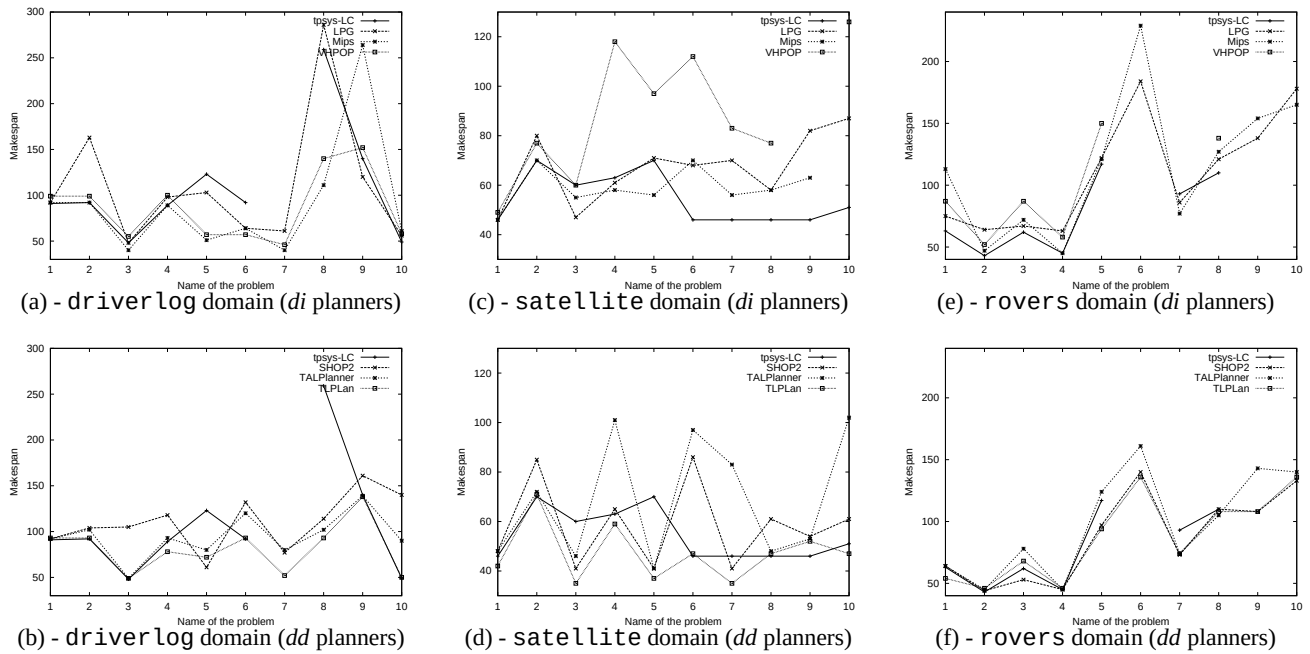


Figure 3: Comparison of the quality of the plans obtained by tpsys-LC and some state-of-the-art planners.

commitment can be applied in a Graphplan-based temporal (or classical) approach, substituting the backward search by the new two-stage search. The new search combines the information calculated in the planning graph with a heuristic, greedy search process, and increases the planner scalability. The advantage is that the planner can exploit a high level of action concurrency, what leads to plans highly competitive with other state-of-the-art planners under a deterministic approach. Although in temporal planning the most important criterion of quality is the makespan, the experimental results show that the new search generates plans in which the number of actions is quite good (even in problems of IPC-1998 and IPC-2000). The main disadvantage of this approach is that it is not complete preserving. However, although completeness and optimality are desired properties, guaranteeing them entails a huge complexity, preventing planners from producing plans with more than a few actions. The algorithm has still some limitations and our future work is to refine the heuristic functions to improve the quality of the plans, the performance and to avoid some of the inconveniences of the greedy search.

Acknowledgments

This work has been partially supported by the Spanish MCyT under projects DPI2001-2094-C03-03 and TIC2001-4936-E, and by the Universidad Polit cnica de Valencia under projects 20010017 and 20010980.

References

[Blum and Furst, 1997] A.L. Blum and M.L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90:281–300, 1997.

[Edelkamp, 2002] S. Edelkamp. Mixed propositional and numeric planning in the model checking integrated planning system. In *Proc. Workshop on Planning for Temporal Domains (AIPS-2002)*, pages 47–55, 2002.

[Fox and Long, 1999] M. Fox and D. Long. The detection and exploitation of symmetry in planning domains. In *Proc. IJCAI-99*, pages 956–961, 1999.

[Fox and Long, 2001] M. Fox and D. Long. PDDL2.1: an extension to PDDL for expressing temporal planning domains. Technical report, University of Durham, UK, 2001.

[Garrido *et al.*, 2002] A. Garrido, M. Fox, and D. Long. A temporal planning system for durative actions of PDDL2.1. In *Proc. ECAI-2002*, pages 586–590, 2002.

[Gerevini and Serina, 2002] A. Gerevini and I. Serina. LPG: a planner based on local search for planning graphs with action costs. In *Proc. AIPS-2002*, pages 281–290, 2002.

[Hoffmann, 2000] J. Hoffmann. A heuristic for domain independent planning and its use in an enforced hill-climbing algorithm. In *Proc. 12th Int. Symp. on Methodologies for Intelligent Systems*, Charlotte, North Carolina, USA, 2000.

[Smith and Weld, 1999] D.E Smith and D.S. Weld. Temporal planning with mutual exclusion reasoning. In *Proc. IJCAI-99*, pages 326–337, Stockholm, Sweden, 1999.

[Weld, 1994] D. Weld. An introduction to least commitment planning. *AI Magazine*, 15(4):93–123, 1994.

[Zimmerman and Kambhampati, 1999] T. Zimmerman and S. Kambhampati. Exploiting symmetry in the planning-graph via explanation-guided search. In *Proc. AAAI-99*, 1999.

Resource Temporal Networks: Definition and Complexity

Philippe Laborie

ILOG

9, rue de Verdun, 94253 Gentilly Cedex, France
plaborie@ilog.fr

Abstract

This paper introduces the concept of Resource Temporal Network (RTN), a constraint network that subsumes both classical attributes used in A.I. Planning and capacity resources traditionally handled in Scheduling. After giving a formal definition of RTNs, we analyze their expressive power and study complexities of several fragments of the RTN framework. We show that solving an RTN is in general NP-Complete - which is not surprising given the expressivity of the framework - whereas computing a Necessary Truth Criterion is polynomial. This last result opens the door for promising algorithms to solve RTNs.

1 Introduction

Historically, *Artificial Intelligence Planning* focuses on absolute changes (for instance change of the truth value of a predicate), conditions on the state of the world and symbolical precedences between operations whereas *Scheduling* focuses on relative changes of the world (resource consumption or production) and numerical time. Recent advances in both fields have enlarged their relative ambition: time and resources are increasingly being studied by the AI Planning community whereas conditions and absolute changes are necessary to express complex scheduling problems involving for example alternative recipes or complex conditions and effects on the level of resources. This paper introduces and analyzes the computational complexity of a formalism that mixes on the same fluent the ingredients of AI Planning (absolute changes, conditions) and the ones of Scheduling (relative changes, numerical time). There has been some recent work on the decidability of planning with numerical state variables [Helmert, 2002] but to the best of our knowledge, no computational complexity study has been published that analyzes a complex but realistic and useful fragment of the overall framework of planning with time and numerical state variables. This paper is a step in this direction.

2 Resource Temporal Networks

2.1 Definition

Definition 1 (Resource) *A resource is a numerical fluent whose value can vary over time. The value of this fluent at date t is called the resource level at date t and is denoted $l(t)$.*

In this paper, we focus on time and resource levels belonging to a dense set (rational numbers): $t \in \mathbb{Q}$, $l(t) \in \mathbb{Q}$. Although the framework can be extended to handle continuous changes, we only consider discrete changes.

Definition 2 (Resource Statements) *We define the following statements on a resource where t , t_s and t_e denote variable time-points and q some constant in $\mathbb{Q}$:*

- An absolute change is a statement denoted $A(q, t)$ stating that because of this change at date t , the level of the resource changes from a current level l to q .
- A relative change is a statement denoted $R(q, t)$ stating that because of this change at date t , the level of the resource changes from current level l to $l + q$. This corresponds to a production of $|q|$ units if $q > 0$ or a consumption of $|q|$ units if $q < 0$.
- A lower-than condition is a statement denoted $L(q, t_s, t_e)$ stating that the level of the resource must remain lower than or equal to q over time interval $[t_s, t_e]$.
- A greater-than condition is a statement denoted $G(q, t_s, t_e)$ stating that the level of the resource must remain greater than or equal to q over time interval $[t_s, t_e]$.

$\mathcal{A}, \mathcal{R}, \mathcal{L}, \mathcal{G}$ respectively denote the sets of absolute, relative changes, lower-than and greater-than conditions. If $x \in \mathcal{A} \cup \mathcal{R}$ is a resource change, we denote $t(x)$ the time-point of x and $q(x)$ the resource quantity involved in x . If $x \in \mathcal{L} \cup \mathcal{G}$ is a condition, we denote $t_s(x)$ the start time-point of x , $t_e(x)$ its end time-point and $q(x)$ the resource quantity involved in x . Note that the above resource statements also allow for expressing equal conditions of the form $E(q, t_s, t_e)$ stating that the level of the resource remains equal to q over the time interval $[t_s, t_e]$ as the conjunction of a lower-than condition $L(q, t_s, t_e)$ and a greater-than condition $G(q, t_s, t_e)$. The set of equal conditions is denoted $\mathcal{E}$. Resource statements are

the basic ingredients for defining Resource Temporal Networks (RTNs). Informally, an RTN represents a set of possible evolutions of a given numerical variable (the level of the resource) over time.

Definition 3 (Resource Temporal Network) A resource temporal network (RTN) is a tuple $(\mathcal{T}, \mathcal{A}, \mathcal{R}, \mathcal{L}, \mathcal{G}, \mathcal{N})$ where: $\mathcal{T}$ is a set of time-point variables, $\mathcal{A}, \mathcal{R}, \mathcal{L}$ and $\mathcal{G}$ respectively denote a set of absolute, relative changes, lower-than and greater-than conditions referring to time-points in $\mathcal{T}$. $\mathcal{N}$ is a Temporal Network using the formalism $STN^\neq$ defined in [Gerevini and Cristani, 1997]¹ on the time-points of $\mathcal{T}$.

In this paper, for simplicity reasons, we focus on decision problems and reasoning involving a unique resource. Our framework can naturally be extended toward (1) parametrized resources of the form $Res(y)$ where y is a variable parameter describing which resource a given resource statement applies and (2) handling variable quantities q in resource statements. Those extensions are out of the scope of the paper.

Definition 4 (Instantiation of an RTN) An instantiation of an RTN is a function $\sigma : \mathcal{T} \rightarrow \mathbb{Q}$.

It is to be noted that in this paper, we assume that all the resource statements are known and the only decision variables of an RTN are the time-points in $\mathcal{T}$. We do not handle, at this point, the complete AI Planning problem of generating an RTN given a partial RTN and a set of operators described as RTNs.

Definition 5 (Time-consistent instantiation) An instantiation σ is said to be time-consistent iff σ is consistent with the $STN^\neq \mathcal{N}$.

Computing a time-consistent instantiation of an RTN can be done in $O(n^3)$ [Gerevini and Cristani, 1997]. We now define the notion of resource-consistent instantiation. In the definition below, conditions [1] and [2] ensure that in a resource-consistent instantiation, two absolute changes assigning different levels cannot be simultaneous and an absolute change cannot be simultaneous with a relative change. The rationale for this semantics is that, as in [Fox and Long, 2002], we allow simultaneity only for pair of changes (x, y) that are commutative, that is such that applying x just after y leads to the same level as applying y just after x . Given a date τ , point [3] defines $\tilde{\sigma}(\tau)$, the last date before τ at which an absolute change occurred and $\tilde{q}(\tau)$ the resource level assigned by such an absolute change. Note that $\tilde{q}(\tau)$ can be defined without ambiguity because two absolute changes assigning different levels cannot be simultaneous. Equation [4] defines the profile of the resource level over time. Conditions [5] and [6] state that this profile must satisfy the lower-than and greater-than conditions.

Definition 6 (Resource-consistent instantiation) An instantiation σ is said to be resource-consistent iff the following conditions are satisfied:

- $\forall (x, y) \in \mathcal{A}^2, q(x) \neq q(y) \Rightarrow \sigma \circ t(x) \neq \sigma \circ t(y)$ [1]
- $\forall (x, y) \in \mathcal{A} \times \mathcal{R}, \sigma \circ t(x) \neq \sigma \circ t(y)$ [2]

¹This formalism allows for both the representation of metric constraints $t - t' \leq r \in \mathbb{Q}$ and unequations $t \neq t'$.

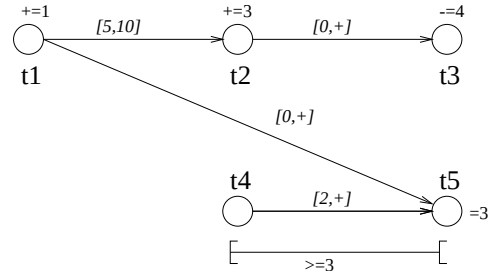


Figure 1: Example of RTN

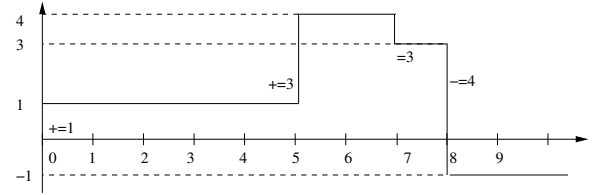


Figure 2: Solution of an RTN

- $\forall \tau \in \mathbb{Q}$, let's define $\tilde{\sigma}(\tau)$ and $\tilde{q}(\tau)$ as follows [3]
 - if $\exists x \in \mathcal{A}, \sigma \circ t(x) \leq \tau$, then:
 - * $\tilde{\sigma}(\tau) = \max_{x \in \mathcal{A}, \sigma \circ t(x) \leq \tau} \sigma \circ t(x)$
 - * $\tilde{q}(\tau) = q(x)$ with $x \in \mathcal{A}, \sigma \circ t(x) = \tilde{\sigma}(\tau)$
 - otherwise, $\tilde{\sigma}(\tau) = -\infty, \tilde{q}(\tau) = 0$.

There exists a function $l_\sigma(\tau) : \mathbb{Q} \mapsto \mathbb{Q}$ such that :

- $l_\sigma(\tau) = \tilde{q}(\tau) + \sum_{x \in \mathcal{R}, \tilde{\sigma}(\tau) < \sigma \circ t(x) \leq \tau} q(x)$ [4]
- $\forall x \in \mathcal{L}, \forall \tau \in [\sigma \circ t_s(x), \sigma \circ t_e(x)), l_\sigma(\tau) \leq q(x)$ [5]
- $\forall x \in \mathcal{G}, \forall \tau \in [\sigma \circ t_s(x), \sigma \circ t_e(x)), l_\sigma(\tau) \geq q(x)$ [6]

Definition 7 (Solution of a RTN) A solution to an RTN is an instantiation σ that is both time- and resource-consistent.

2.2 Example

Suppose the following RTN: $\mathcal{T} = \{t_1, t_2, t_3, t_4, t_5\}$, $\mathcal{A} = \{A(3, t_5)\}$, $\mathcal{R} = \{R(1, t_1), R(3, t_2), R(-4, t_3)\}$, $\mathcal{L} = \emptyset$, $\mathcal{G} = \{G(3, t_4, t_5)\}$, $\mathcal{N} = \{t_2 - t_1 \in [5, 10], t_3 - t_2 \in [0, +\infty), t_5 - t_1 \in [0, +\infty), t_5 - t_4 \in [2, +\infty)\}$. This RTN is depicted in Figure 1. The following instantiation σ is clearly a solution to this RTN: $\sigma(t_1) = 0, \sigma(t_2) = \sigma(t_4) = 5, \sigma(t_3) = 8, \sigma(t_5) = 7$. It corresponds to the function $l_\sigma(t)$ depicted in Figure 2.

3 Expressive Power

In this section, we show that most of the classical attributes used in AI Planning as well as most of the resources used in Scheduling can be represented in the RTN framework.

3.1 Planning Attributes

STRIPS operators

Let p be a STRIPS predicate. It can be represented by a resource for which level 0 means that p is false and level 1 means that p is true. Let o be an operator at time-point t . If p

is in the precondition of operator o , this can be captured by a greater-than condition $G(1, t', t)$ with the constraint $t' \prec t^2$. If $\neg p$ is in the precondition of operator o , this can be captured by a lower-than condition $L(0, t', t)$. If p is in the delete list of operator o , this can be captured by an absolute change $A(0, t)$ stating that after t , p will be false. If p is in the add list of operator o , this can be captured by an absolute change $A(1, t)$ stating that after t , p will be true.

IXTeT attributes

Let att be an IXTeT attribute [Ghallab and Laruelle, 1994]. We can build a mapping μ_{att} that maps the possible values of attribute att to $\mathbb{Q}$. Then, a hold predicate $hold(att, v, t, t')$ can be modeled by an equal condition $E(\mu_{att}(v), t, t')$. An event $event(att, v, v', t)$ by the conjunction of an equal condition $E(\mu_{att}(v), t', t)$, an absolute change $A(\mu_{att}(v'), t)$ and a temporal constraint $t' \prec t$.

PDDL 2.1

Let's consider a durative action of PDDL 2.1 [Fox and Long, 2002]. This action can be represented by two time points t_s (start) and t_e (end) in our formalism. A condition at start on a non-numerical proposition p can be captured by a greater-than condition $G(1, t, t_s)$ with the precedence $t \prec t_s$, an effect at start by an absolute change $A(1, t_s)$ (similar modeling for conditions and effects at end) and an invariant condition by a greater-than condition $G(1, t_s, t_e)$. Conditions and invariants of the form $\leq$, $\geq$ and $=$ on numeric variables can be captured by RTN conditions L , G and E . Numeric effects *assign* correspond to absolute changes, whereas *increase* and *decrease* correspond to relative changes.

3.2 Scheduling Resources

Discrete resources

A discrete resource of maximal capacity Q [Laborie, 2003] can be captured by an RTN with a greater-than condition $G(0, -\infty, +\infty)$ ³ a lower-than condition $L(Q, -\infty, +\infty)$ and an initial production $R(Q, -\infty)$. An activity requiring q units of resource over the time interval $[t_s, t_e]$ is represented by a pair of relative changes $R(-q, t_s)$, $R(+q, t_e)$. If the discrete resource is given a varying maximal capacity profile, this can be modeled as a set of lower-than conditions $L(Q_i, t_i, t_{i+1})$.

Reservoir

A reservoir of maximal capacity Q and initial level L [Laborie, 2003] can be captured by an RTN with a greater-than condition $G(0, -\infty, +\infty)$, a lower-than condition $L(Q, -\infty, +\infty)$ and an absolute change $A(L, -\infty)$. A production activity corresponds to a relative change $R(q, t)$ where $q > 0$ whereas a consumption activity corresponds to a relative change $R(q, t)$ where $q < 0$.

State Resources

In ILOG Scheduler [ILOG, 2002], state resources are defined as objects that at each timepoint can take only one possible

state among a known set of possible states $S = \{s_i\}$. Activities requiring different states of the state resource cannot overlap. We can build a mapping μ_{res} that maps the possible states of the state resource to $\mathbb{Q}$. Then, the requirement of a given state s of a state resource by an activity executing on the time interval $[t_s, t_e]$ can be modeled as an absolute change $A(\mu_{res}(s), t_s)$ and an equal condition $E(\mu_{res}(s), t_s, t_e)$.

Additional Expressivity

The RTN framework allows for modeling complex resources and activities in scheduling. In manufacturing for instance, maintenance activities need to be executed as soon as the level of some numerical variable (measuring the “need for maintenance”) has reached a certain level. The level of this variable is increased (relative change) by production activities and is reset to 0 (absolute change) by maintenance activities. Another example is scheduling while ensuring some condition on a numerical variable during the execution of an activity (e.g. maintaining the temperature of a furnace within a suitable interval). Such kind of conditions are very important in process industry and chemistry. The conjunction of absolute changes, relative changes and conditions holding over variable time intervals offers a powerful formalism for representing complex scheduling problems. Additional features such as dependence between variable resource quantities q and time-points t that do not directly fit into the RTN model can be handled by additional constraints in a constraint propagation framework.

4 Complexity

In this section, we analyze the algorithmic complexity of solving and providing necessary truth criteria for general RTNs and particular fragments of the RTN formalism. By NP-Complete we mean NP-Complete in the strong sense.

4.1 Notations

Let us consider the following notations about temporal constraints: PA denotes the point algebra of [Vilain and Kautz, 1986] which is a restriction of $STN^\neq$ that only consists of the set of qualitative relations $\{\emptyset, \prec, \preceq, =, \succ, \succeq, \neq, ?\}$ between timepoints. $STN^\neq$ denotes a general $STN^\neq$. We write $t \prec t'$ or $t \preceq t'$ to express the fact that the corresponding relation is subsumed by the temporal network. We use the following notations about resource statements: $\mathcal{R}^+$ denote any set of relative changes x such that $q(x) > 0$ (producers). $\mathcal{R}_1^{+-}$ denote any set of pairs of resource statements $(R(+1, t_s), R(-1, t_e))$ where $t_s \prec t_e$. $\mathcal{L}_Q$ denotes the lower-than condition $L(Q, -\infty, +\infty)$. $\mathcal{AE}$ denotes any set of pairs (x, y) where $x = A(q, t_s)$ and $y = E(q, t_s, t_e)$ with $t_s \prec t_e$. A fragment of the complete RTN framework is denoted $(X|Y)$ where X is the set of changes and conditions allowed in this fragment and Y the type of temporal constraints. n denotes the number of resource statements in the RTN, m the number of temporal constraints between time-points and $\maxflow(n, m)$ the complexity of computing a maximum flow on a graph with n nodes and m arcs⁴.

² $t' \prec t$ denotes the constraint $t - t' \in (0, +\infty)$.

³We assume that $-\infty$ denotes a time-point before any other time-point and $+\infty$ denotes a time-point after any other time-point.

⁴State-of-the-art maximum flow algorithms do it in $O(nm \log n)$ in worst case [Hochbaum, 1998].

4.2 Finding a solution

Complexity 1 *The problem of finding a solution to an RTN is in NP.*

Proof: The time-consistency of an instantiation σ can be checked in polynomial time and, given its definition, the function l_σ can be build and the resource-consistency checked in polynomial time. A simple algorithm to check that an instantiation is a solution runs in $O(m + n \log n)$. ■

The proof of the three polynomiality results below is omitted because trivial.

Complexity 2 $(\mathcal{R}^+, \mathcal{L}_Q | STN^\neq)$ can be solved in $O(n^3)$.

Complexity 3 $(\mathcal{R}_1^{+-}, \mathcal{L}_1 | PA)$ can be solved in $O(n^3)$.

Complexity 4 $(\mathcal{A}, \mathcal{R} | STN^\neq)$ can be solved in $O(n^3)$.

Complexity 5 $(\mathcal{R}_1^{+-}, \mathcal{L}_1 | STN^\neq)$ is NP-Complete.

Proof: The scheduling decision problem $1|prec(l_{ij}), p_j = 1|C_{max} \leq T$ that consists in scheduling on a machine a set of activities of unit duration subject to min/max delays between activities and maximal makespan is a special case of $(\mathcal{R}_1^{+-}, \mathcal{L}_1 | STN^\neq)$. See for instance [Finta and Liu, 1996] for a proof of its NP-completeness. ■

Complexity 6 $(\mathcal{AE} | STN^\neq)$ is NP-Complete.

Proof: The special case where all the quantities q of the pairs (x, y) are different can be reformulated as a problem $(\mathcal{R}_1^{+-}, \mathcal{L}_1 | STN^\neq)$ (cf above). ■

Complexity 7 $(\mathcal{R}^+, \mathcal{E} | PA)$ is NP-Complete.

Proof: The Subset Sum Problem (SSP), which is known to be NP-complete [Garey and Johnson, 1979] can be reduced to this problem. **SSP:** Given a finite set R ; and for each element $x \in R$ a positive integer size $q(x)$; a positive integer Q . Is there a subset S of R such that $\sum_{x \in S} q(x) = Q$? Let the RTN composed of the resource relative changes $R(q(x), t(x))$ for $x \in R$ and only one equal condition $E(Q, \tau, \tau')$ with $\tau \prec \tau'$. It's clear that if there exists a solution S to the SSP then this solution can clearly be transformed into a solution to the RTN; for example by choosing $\tau = 0, \tau' = 1, \forall x \in S, t(x) = 0, \forall x \in R \setminus S, t(x) = 1$. Reciprocally, for any solution σ to the RTN, a solution to the corresponding SSP can be built by taking $S = \{x \in R / \sigma \circ t(x) \leq \sigma(\tau)\}$. Note that SSP is not NP-Complete in the strong sense. $(\mathcal{R}^+, \mathcal{E} | PA)$ can be proved to be NP-Complete in the strong sense by a transformation from the 3-Partition problem [Garey and Johnson, 1979] very similar to the one used in this proof. ■

Complexity 8 $(\mathcal{A}, \mathcal{R}^+, \mathcal{L}_Q | PA)$ is NP-Complete.

Proof: The Bin Packing Problem (BPP), which is known to be NP-complete [Garey and Johnson, 1979] can be reduced to this problem. **BPP:** Given a finite set R of m items; for each item $x \in R$ a positive integer size $q(x)$; a positive integer Q and $k \leq m$. Can R be partitioned into k disjoint sets $R_1, \dots, R_k$ such that $\forall i \in [1, k], \sum_{x \in R_i} q(x) \leq Q$? Let the RTN composed of the resource relative changes $\{R(q(x), t(x))\}_{x \in R}$, a temporal chain of $k - 1$ absolute changes $\{A(0, \tau_i)\}_{i=1..k-1}$ with the precedence constraint

$\tau_i \prec \tau_{i+1}$ and the lower-than condition $L(Q, -\infty, +\infty)$. If there exists a solution $R_1, \dots, R_k$ to the BPP, it's clear that a solution σ to the RTN can be built as follows: $\sigma(\tau_i) = 2i, \forall x \in R_i, \sigma \circ t(x) = 2i - 1$. Reciprocally, a solution to the RTN problem can be used to build a solution to the BPP by defining the sets $R_1 = \{x \in R / \sigma \circ t(x) < \sigma(\tau_1)\}$ and, for $1 < i < k, R_i = \{x \in R / \sigma(\tau_{i-1}) \leq \sigma \circ t(x) < \sigma(\tau_i)\}$ and $R_k = \{x \in R / \sigma(\tau_{k-1}) \leq \sigma \circ t(x)\}$. ■

Complexity 9 $(\mathcal{R}_1^{+-}, \mathcal{L}_Q | PA)$ is NP-Complete.

Proof: The Sequencing to Minimize Maximum Cumulative Cost Problem (SMMCCP) with tasks costs in $\{-1, 1\}$, which is known to be NP-complete [Garey and Johnson, 1979] can be reduced to this problem. **SMMCCP:** Given a set of tasks T , a partial order between tasks $\prec$, a fixed cost $c(t) \in \{-1, 1\}$ for each task $t \in T$ and a constant K , find a total order $<$ on T that obeys the partial order $\prec$ and such that $\forall t \in T, \sum_{t' < t} c(t') \leq K$. Given a SMMCCP, we can build an RTN as follows. Each task t corresponds to a time-point in the RTN. The precedence constraints correspond to the partial order $\prec$ between tasks. Two additional time-points $-\infty$ and $+\infty$ are respectively added before and after all the other time points $t \in T$. For a task t such that $c(t) = +1$, we add the two relative changes $R(+1, t)$ and $R(-1, +\infty)$. For a task t such that $c(t) = -1$, we add the two relative changes $R(+1, -\infty)$ and $R(-1, t)$. Let N be the number of tasks t such that $c(t) = -1$. We then add the lower-than condition $L(K + N, -\infty, +\infty)$. Let $i(t) \in [1, |T|]$ be an arbitrary indexing of the set of tasks T . Let σ be a solution to the RTN. We build a total order between tasks as follows: $t' < t$ if and only if $\sigma(t') < \sigma(t)$ or $[\sigma(t') = \sigma(t) \text{ and } c(t') = -1 \text{ and } c(t) = +1]$ or $[\sigma(t') = \sigma(t) \text{ and } c(t') = c(t) \text{ and } i(t') < i(t)]$. It's easy to see that $<$ is a solution to the original SMMCCP. ■

4.3 Necessary Truth Criterion

The Necessary Truth Criterion (NTC) is the problem of deciding whether a given RTN is such that any of its time-consistent instantiation is also resource-consistent (and thus, is a solution). NTCs were introduced by David Chapman in a very influential paper [Chapman, 1987] where, in the context of planning with resources, he mentions the existence of “clever polynomial algorithms using network flow techniques that compute the exact least value the state variable could take on”. One of these algorithms was described in [Rhys, 1970] and more recently in [Muscettola, 2002]. In this section, we develop this idea and give some complexity results for NTCs in the context of resource temporal networks. Notice that, due to lack of space, proofs are mostly sketched. Note also that all the problems studied in this section are NP-Complete (see section 4.2). Without lack of generality we assume in this section that the RTNs do not contain any potentially conflicting pairs of changes. A pair of absolute changes $(x, x') \in \mathcal{A}^2$ is said to be potentially conflicting if and only if $q(x) \neq q(x')$ and $\mathcal{N}$ does not subsume $t(x) \neq t(x')$. A pair $(x, x') \in \mathcal{A} \times \mathcal{R}$ is said to be potentially conflicting if and only if $\mathcal{N}$ does not subsume $t(x) \neq t(x')$. Indeed, if such conflicting pairs exist, it is easy to add the corresponding non-simultaneity constraints $t(x) \neq t(x')$ on $\mathcal{N}$.

Lemma 1 *NTC for $(\mathcal{R}, \mathcal{L}_Q | STN^\neq)$ can be solved in $O(\text{maxflow}(n, m))$.*

Proof: First, we recap a result of [Rhys, 1970] showing that NTC for $(\mathcal{R}, \mathcal{L}_Q | PA)$ is polynomial. The problem can be expressed as a Maximum Weight Closure Problem (MWCP) on the directed graph where the arcs $\prec, \preceq$ of the PA network have been reversed and all arcs $\neq$ are ignored. **MWCP:** Given a directed graph $G = (V, A)$ and a node weight $q(x) \in \mathbb{Q}$ for each node in $x \in V$. A subset $S \subseteq V$ is called a closure if and only if it does not contain any outgoing arc that is $\forall x \in S, (x, y) \in A \Rightarrow y \in S$. The problem is to find the maximum weight closure. The RTN satisfies the NTC if and only if the weight of the maximum weight closure is lower than or equal to Q . Indeed, if S^* denotes the maximum weight closure, there exists an instantiation of the RTN such that all the changes in S^* are scheduled before all the other changes and the resource level after all the changes in S^* is the maximum resource level reachable in all time-consistent instantiations. This level is exactly the weight of S^* . As shown in [Ahuja et al., 1993], the MWCP is equivalent to the min-cut/max-flow problem. For the more general problem $(\mathcal{R}, \mathcal{L}_Q | STN^\neq)$, similar arguments as in [Muscatella, 2002] (separation schedule) can be used to show that the RTN satisfies the NTC if and only if the symbolical PA temporal constraint network subsumed by the $STN^\neq$ satisfies the NTC. ■

Lemma 2 *NTC for $(\mathcal{A}, \mathcal{R}^+, \mathcal{L}_Q | PA)$ can be solved in $O(\text{maxflow}(n, m))$.*

Proof: We can note that for any instantiation σ , the set of chronologically ordered absolute changes $\mathcal{A} = \{x_i\}_{i=1..n}$ with $\sigma \circ t(x_i) \leq \sigma \circ t(x_{i+1})$ partitions the set of changes $\mathcal{A} \cup \mathcal{R}^+$ into $n+1$ subsets: $S_0 = \{y \in \mathcal{R}^+, \sigma \circ t(y) < \sigma \circ t(x_1)\}$, for $i = 1..n-1$, $S_i = \{x_i\} \cup \{y \in \mathcal{R}^+, \sigma \circ t(x_i) < \sigma \circ t(y) < \sigma \circ t(x_{i+1})\}$, $S_n = \{x_n\} \cup \{y \in \mathcal{R}^+, \sigma \circ t(x_n) < \sigma \circ t(y)\}$. S_i is called the layer of x_i . In this context, the maximal level of the resource in instantiation σ is the weight of the maximum weight layer $S \subseteq \mathcal{A} \cup \mathcal{R}^+$ defined as $\sum_{x \in S} q(x)$. Given an instance of RTN, we build an undirected graph G as follows. Each change $x \in \mathcal{A} \cup \mathcal{R}^+$ is associated a unique node in G whose weight is equal to $q(x)$. Informally, there will be an edge between two changes $\{x, x'\}$ if and only if they cannot belong to the same layer in a time-consistent instantiation. More precisely, there is an edge in G between each pair of absolute change. Furthermore, if $x \in \mathcal{A} \cup \mathcal{R}^+$ is a change and $x' \in \mathcal{R}^+$ is a relative change, there is an edge $\{x, x'\}$ in G if and only if there exists an absolute change $y \in \mathcal{A}$ such that $x \preceq y \preceq x'$. It is not difficult to show that the maximal level of the resource is equal to the weight of the maximum independent set of G . Furthermore G is a comparability graph as the edges involving at least one relative change can naturally be oriented with the direction of a path containing an absolute change between them and the remaining of the edges (between pairs of absolute changes) forms a clique. Computing the maximum weight independent set of a comparability graph is polynomial and can be solved as a min-flow problem [Golumbic, 1980]. Thus, the NTC only needs to compare the weight of the maximum weight independent set of G with the maximal resource level Q . ■

The following lemma is a generalization of Lemma 2 when all changes are allowed including consumers ($q < 0$).

Lemma 3 *NTC for $(\mathcal{A}, \mathcal{R}, \mathcal{L}_Q | PA)$ can be solved in $O(n \text{maxflow}(n, m))$.*

Proof: We prove that the problem can be solved by solving n_A Maximum Weight Convex Set Problems (MWCSPP) where $n_A \leq n$ is the number of absolute changes. **MWCSPP:** Given a poset $G = (V, \leq)$ where $\leq$ is a partial order on V (reflexive, antisymmetric and transitive relation) and a weight function $w(x) \in \mathbb{Q}$. A subset $S \subseteq V$ is said to be convex if and only if $\forall (x, y) \in S^2, \forall z \in V, x \leq z \leq y \Rightarrow z \in S$. The problem is to find a convex subset $S \subseteq V$ of maximum total weight. As shown in [Gröflin, 1985], this problem can be solved in polynomial time by a min-flow/max-cut algorithm. For a given absolute change $x_i \in \mathcal{A}$, we transform the problem of computing the maximum level of the resource in the layer of x_i (see sketch of proof of Lemma 2) as an instance of MWCSPP. The poset $G_i = (V_i, \leq)$ of this MWCSPP is built as follows: $V_i = \{x_i\} \cup \{x \in \mathcal{A} \cup \mathcal{R}, \neg(x \preceq x_i)\}$ and $x \leq y$ if and only if relation $x \preceq y$ is subsumed by the PA network. $w(x_i) = K + q(x_i)$ where K is big enough ($K > \sum_{x \in \mathcal{R}} |q(x)|$), $w(x) = -\infty$ if $x \in \mathcal{A} \setminus \{x_i\}$ and $w(x) = q(x)$ if $x \in \mathcal{R}$. One can show that for any convex set S of G_i containing x_i , there exists a time-consistent instantiation σ and a time t such that $l_\sigma(t) = w(S) - K$ and that for any time-consistent instantiation σ and a time t , the set $\{x \in V_i / \sigma \circ t(x) \leq t\}$ is convex. Thus, the maximum resource level reachable in the layer of x_i is $w(S_i^*) - K$ where S_i^* is the maximum weight convex set of G_i . Note that due to the choice of K , $x_i \in S_i^*$. ■

We can see that Lemmas 1, 2 and 3 also hold when the unique lower-than condition does not span over the complete horizon $(-\infty, +\infty)$ but on some variable time interval $[t_s, t_e]$ where $\{t_s, t_e\} \subset \mathcal{T}$. Indeed, in this case, the problem can be transformed into a problem where the lower-than condition spans over the complete horizon and three additional relative changes are considered: $R(-K, -\infty)$, $R(+K, t_s)$, $R(-K, t_e)$ where K is big enough.

Furthermore, Lemmas 1 and 3 also hold for a greater-than condition $\mathcal{G}_Q$ as, here again, the problem can easily be transformed into a problem with a lower-than condition $\mathcal{L}_{-Q}$ and with opposite resource changes.

Theorem 1 *Let $\Pi = (\mathcal{T}, \mathcal{A}, \mathcal{R}, \mathcal{L}, \mathcal{G}, \mathcal{N})$ be an RTN. For $y \in \mathcal{L} \cup \mathcal{G}$, let Π_y denote the RTN where y is the only condition in other words $\Pi_y = (\mathcal{T}, \mathcal{A}, \mathcal{R}, \mathcal{L} \cap \{y\}, \mathcal{G} \cap \{y\}, \mathcal{N})$. $NTC(\Pi)$ is true if and only if for each condition statement $y \in \mathcal{L} \cup \mathcal{G}$, $NTC(\Pi_y)$ is true.*

Proof: The theorem gives a necessary condition for $NTC(\Pi)$ to be true because if the condition is not met, a time-consistent instantiation can be built that is not resource-consistent. Reciprocally, if $NTC(\Pi)$ is false, it means that there exists a time-consistent instantiation that is not resource-consistent. This time-consistent instantiation violates at least one condition statement $y \in \mathcal{L} \cup \mathcal{G}$. ■

The following complexity results use Theorem 1 to extend Lemmas 1 and 3 to the more general case of any set of condition statements. For each of these results, one can show that

problem Π_y can be transformed into one of the corresponding lemma and thus, its time complexity is polynomial.

Complexity 10 *NTC for $(\mathcal{R}, \mathcal{L}, \mathcal{G} | STN^\neq)$ can be solved in $O(n \maxflow(n, m))$.*

Complexity 11 *NTC for $(\mathcal{A}, \mathcal{R}, \mathcal{L}, \mathcal{G} | PA)$ can be solved in $O(n^2 \maxflow(n, m))$.*

5 Conclusion and Future Work

In this paper, we introduce the notion of RTN to express a large panel of possible evolutions of a given numerical state variable over time. RTNs allow modeling on the same fluent features of classical AI Planning (absolute changes, conditions) and Scheduling (relative changes). We show that computing a solution to an RTN is in general NP-Complete whereas determining whether all time-consistent instantiations of an RTN are solutions is polynomial. This last result indicates that efficient solving methods based on such polynomial Necessary Truth Criteria can be developed. Indeed, when an RTN does not satisfy the NTC, all the algorithms we mention can exhibit a subset of changes sufficient to explain why some time-consistent instantiations are not a solution. Just like in classical planning or scheduling, these potential conflicts can be used to branch in a search tree until the NTC is true and to perform constraint propagation.

A direction for future work will consist in studying the complexity of the only problem whose complexity is still open: NTC for $(\mathcal{A}, \mathcal{R}, \mathcal{L}, \mathcal{G} | STN^\neq)$. Based on the fairly optimistic results described in this paper, we also plan to work on the development of practical and efficient algorithms for solving NTC and for finding solutions (branching schemes, heuristics, computation of resource envelopes, constraint propagation). We think that the numerous and well-studied combinatorial problems we found tightly related with RTNs (one-machine scheduling problems, subset sum, bin packing, sequencing to minimize maximum cumulative costs, maximum weight closure, maximum weight independent set, maximum weight convex set) can also help to solve RTNs. For instance, state-of-the-art bin packing algorithms could be a source of inspiration for $(\mathcal{A}, \mathcal{R}^+, \mathcal{L}_Q | STN^\neq)$. Extension of the framework to integrate and mix continuous relative changes (see continuous reservoirs [ILOG, 2002]) and absolute changes (see [Penberthy and Weld, 1994; Trinquart and Ghallab, 2001]) is also clearly of interest.

6 Acknowledgements

I am very grateful to André Kezdy for pointing me to some relevant articles about the maximum weight convex set problem as well as to many other graph theorists who got interested in this problem. Special thanks to Emmanuel Guéré and Pascal Massimino for enlightening discussions on the RTN semantics and complexity issues and to Francis Sourd for carefully proofreading the paper.

References

[Ahuja et al., 1993] R. Ahuja, T. Magnanti, and J. Orlin. *Network Flows: Theory, Algorithms and Applications*. Prentice-Hall, 1993.

- [Chapman, 1987] D. Chapman. Planning for Conjunctive Goals. *Artificial Intelligence*, 32:333–377, 1987.
- [Finta and Liu, 1996] L. Finta and Z. Liu. Single Machine Scheduling Subject to Precedence Delays. *Discrete Applied Mathematics*, 70:247–266, 1996.
- [Fox and Long, 2002] M. Fox and D. Long. PDDL2.1 : An Extension to PDDL for Expressing Temporal Planning Domains, Feb. 2002.
- [Garey and Johnson, 1979] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.
- [Gerevini and Cristani, 1997] A. Gerevini and M. Cristani. On finding a solution in temporal constraint satisfaction problems. In *Proc. IJCAI 97*, pages 1460–1465, 1997.
- [Ghallab and Laruelle, 1994] M. Ghallab and H. Laruelle. Representation and Control in IxTeT, a Temporal Planner. In *Proc. AIPS-94*, pages 61–67, 1994.
- [Golumbic, 1980] M.C. Golumbic. *Algorithmic graph theory and perfect graphs*. Academic Press, New York, 1980.
- [Gröflin, 1985] H. Gröflin. On Node Constraint Networks. *Networks*, 15:469–475, 1985.
- [Helmert, 2002] M. Helmert. Decidability and Undecidability Results for Planning with Numerical State Variables. In *Proc. AIPS-02*, pages 44–53, 2002.
- [Hochbaum, 1998] D. Hochbaum. The pseudoflow algorithm and the pseudoflow-based simplex for the maximum flow problem. In *Proc. IPCO-98*, pages 325–337, Houston Texas, June 1998.
- [ILOG, 2002] ILOG. ILOG Scheduler 5.3 Reference Manual and Users Manual, 2002. <http://www.ilog.com/>.
- [Laborie, 2003] P. Laborie. Algorithms for Propagating Resource Constraints in AI Planning and Scheduling: Existing Approaches and New Results. *Artificial Intelligence*, 143(2):151–188, 2003.
- [Muscettola, 2002] N. Muscettola. Computing the Envelope for Stepwise-Constant Resource Allocations. In *Proc. CP-2002*, pages 139–154, 2002.
- [Penberthy and Weld, 1994] J.S. Penberthy and D.S. Weld. Temporal Planning with Continuous Change. In *Proc. AIPS-94*, pages 1010–1015, 1994.
- [Rhys, 1970] J. Rhys. A selection problem of shared fixed costs and network flows. *Management Science*, 17(3):200–207, 1970.
- [Trinquart and Ghallab, 2001] R. Trinquart and M. Ghallab. An Extended Functional Representation in Temporal Planning: Towards Continuous Change. In *Proc. ECP-01*, Sept 2001.
- [Vilain and Kautz, 1986] M. Vilain and H. Kautz. Constraint propagation algorithms for temporal reasoning. In *Proc. Fifth National Conference on Artificial Intelligence*, pages 377–382, 1986.

Generalizing GraphPlan by Formulating Planning as a CSP

Adriana Lopez

Dept. of Computer Science
University of Toronto
Toronto, Ontario
Canada, M5S 3G4
alopez@cs.toronto.edu

Fahiem Bacchus

Dept. Of Computer Science
University of Toronto
Toronto, Ontario
Canada, M5S 3G4
fbacchus@cs.toronto.edu

Abstract

We examine the approach of encoding planning problems as CSPs more closely. First we present a simple CSP encoding for planning problems and then a set of transformations that can be used to eliminate variables and add new constraints to the encoding. We show that our transformations uncover additional structure in the planning problem, structure that subsumes the structure uncovered by GRAPHPLAN planning graphs. We solve the CSP encoded planning problem by using standard CSP algorithms. Empirical evidence is presented to validate the effectiveness of this approach to solving planning problems, and to show that even a prototype implementation is more effective than standard GRAPHPLAN. Our prototype is even competitive with far more optimized planning graph based implementations. We also demonstrate that this approach can be more easily lifted to more complex types of planning than can planning graphs. In particular, we show that the approach can be easily extended to planning with resources.

1 Introduction

A powerful technique for solving planning problems, first used in [Kautz and Selman, 1996], is to impose a fixed bound on plan length. This converts the problem to one that lies in NP. The resulting problem can then be solved by encoding it in any of a number of alternative NP-complete formalisms, e.g., SAT [Kautz and Selman, 1996], integer programming [Vossen *et al.*, 2001], or *constraint satisfaction problems* (CSPs) [van Beek and Chen, 1999; Do and Kambhampati, 2001], and then using the powerful algorithms that have been designed to solve these general problems.

In this paper we examine the approach of encoding planning problems as CSPs more closely. We use a very simple encoding to translate the planning problem into a CSP. The advantage of using CSPs in general, and our encoding in particular, is that it preserves much of the structure of the original planning problem, and thus various transformations that utilize this specific structure can be applied to create a new CSP that is easier to solve.

The utility of exploiting planning specific structure has been convincingly demonstrated by the BLACKBOX [Kautz

and Selman, 1998] and GP-CSP [Do and Kambhampati, 2001] planners. These systems encode a transformed representation of the planning problem called the *planning graph* to SAT and CSP respectively. The planning graph is a planning specific construction, due to [Blum and Furst, 1997], that computes and represents some important additional structural information about the planning problem. Thus by encoding the planning graph rather than the original planning problem, BLACKBOX and GP-CSP are able to capture more of the structure of the planning problem in their encoding, which results in a significant improvement in performance.

In our approach we bypass the construction of a planning graph. Instead, we study techniques that can equally well exploit the structure of the planning problem and at the same time work directly on the CSP encoding. We demonstrate that it is possible to develop transformations that uncover all of the additional structure obtained from planning graphs.

The advantage of our approach is that it works directly on a much richer and more robust representation. Consequently we obtain at least three important advances over the planning graph construction: (1) Our approach allows us to *generalize planning graphs*. In particular, we can enhance our approach to extract other kinds of structure to allow us to solve the planning problem more efficiently. (2) We can more easily extend our approach to more *complex types of planning problems*, e.g., those involving resource usage. (3) Since the final result is a CSP we can *automatically utilize CSP solution techniques* which can be more powerful than standard GRAPHPLAN searching methods. In the paper we will provide evidence for all of these points.

First, we present our CSP encoding of a planning problem. Then we present a set of transformations that can be used to eliminate variables and add new constraints to the encoding, and show that these transformations subsume and generalize planning graphs. Then we present empirical evidence to validate the effectiveness of our approach to solving planning problems. To demonstrate that our approach can be more easily lifted to more complex types of planning, we show how it can be easily extended to planning with resources. Finally we close with some conclusions and a description of future work.

2 Generating the BASE-CSP

To encode a planning problem as a CSP we impose a bound k on the “size” of the plan. In this paper we will measure

plan size by the number of GraphPlan-concurrent steps (GP-steps) in the plan. It will become apparent in the sequel what GP-steps means.

In k GP-steps each proposition or action can change at most k times. So to encode a k GP-step plan in a CSP we can utilize $k + 1$ sets of propositional variables P_i^s , and k sets of action variables A_j^s , where s ranges from 0 to k for propositions and 0 to $k - 1$ for actions, and i and j range over the number of distinct propositions and action instances in the planning problem respectively. Each of these variables will be boolean. Intuitively, P_i^s means that proposition P_i is true at GraphPlan step s , and A_j^s means that action instance A_j was executed at GraphPlan step s ($\neg P_i^s$ and $\neg A_j^s$ denote the opposite).

Clearly by setting each of these variables we can capture any k GP-step plan and its effects. However, many illegal plans and nonsensical outcomes are also captured. We need constraints to ensure that any solution to the CSP (i.e., any setting of the variables that satisfies all of the constraints) is a legal plan. There are various possible sets of constraints that can serve our purpose. Here we present one particular set:

1. *Unary initial state and goal constraints.* The propositional variables from step zero, P_i^0 , and from step k , P_i^k are required to have values compatible with the initial and goal states of the planning problem.
2. *Precondition constraints.* An action A_j^s cannot be true unless its precondition is satisfied. This gives the constraint $A_j^s \rightarrow \text{Pre}(A_j)^s$, where $\text{Pre}(A_j)^s$ is A_j 's precondition relative to GP-step s .
3. *Successor state constraints.* Implicit in classical planning representations is the frame assumption: predicates not mentioned in the action's description are unaffected by the action. We can capture the effects of actions, including the implicit non-effects in a number of different ways. Here we use Reiter's formulation of successor state axioms [Reiter, 2001]. In particular, for each propositional variable P_i^s at GP-step s ($s > 0$) we have a constraint between it and the same proposition at GP-step $s-1$. The constraint says that P_i^s is true if and only if some action made it true, or it was previously true and no action made it false. Thus the successor state axiom constraints take the form:

$$P_i^s \leftrightarrow \bigvee_{A_j \in \text{Create}(P_i)} A_j^{s-1} \vee (P_i^{s-1} \wedge \bigwedge_{A_j \in \text{Delete}(P_i)} \neg A_j^{s-1}),$$

where $\text{Create}(P_i)$ and $\text{Delete}(P_i)$ are the set of actions that create and delete P_i . It is easy to automatically generate a set of successor state constraints from a set of STRIPS action descriptions. Additionally, by using successor state constraints that mention additional propositions from step $s-1$, it is easy to encode ADL [Pednault, 1989] operators as a set of successor state axioms.¹

¹Note that the successor state constraints do not encode the action preconditions. These are encoded as separate constraints. Thus, the only complication with ADL actions has to do with conditional effects. If P_i is a conditional effect of action A_j , subject to the condition ϕ , the successor state constraint will have a disjunct " $A_j^{s-1} \wedge \phi^{s-1}$ ". That is, P_i^s will be true if A_j was executed and condition ϕ held at the previous step.

4. *GP-Concurrency Constraints.* The above successor state axiom allows the unintended model in which we have two actions at the same step, one creating a proposition and one deleting it. To avoid such unintended solutions we must restrict concurrent actions in some way. The most natural way is a serial constraint, which says that only one action variable can be true at any step.² Another type of constraint is one that imposes GraphPlan concurrency. Basically it asserts that two actions cannot be simultaneously true if they interfere with each other. In this work we have chosen to use the GraphPlan (GP) concurrency constraints.

5. *Non-Null Steps Constraints.* We impose the constraint that for every step s at least one action variable should be true. This blocks null steps in the plan, in contrast [Blum and Furst, 1997] and [Do and Kambhampati, 2001] both allow null plan steps.

We will refer to this set of variables and constraints as the BASE-CSP. Any solution to the BASE-CSP will contain a setting of the action variables that comprises a valid GP-concurrent plan. If the BASE-CSP has no solution then no k GP-step plan exists for the problem.

3 Reduction of the BASE-CSP

Given a BASE-CSP representing a k -step planning problem we can use various transformations to modify it, generating a new CSP that is empirically easier to solve, and that is equivalent to the BASE-CSP in the sense that any solution to the new CSP can easily be extended to a solution to the original BASE-CSP. These transformations include inferring new constraints that can be added to the CSP and eliminating various single valued variables. Our transformations are related to the well known techniques of enforcing local consistency on a CSP prior to solving it [Freuder, 1985], however they are based on taking advantage of the specific logical form of the above set of constraints.

3.1 Adding GraphPlan Mutex Constraints

A well known technique for making CSPs easier to solve is to add redundant constraints, e.g., [Getoor *et al.*, 1997], which are constraints that make explicit additional relations between the variables. Redundant constraints that are useful to add are usually determined by examining the structure of the particular CSP. This is precisely what is done in planning graphs, where insights into the manner in which actions and propositions interact, are used to generate a new set of binary constraints called mutexes.

New binary constraints can be added to a CSP by enforcing *2- j consistency* [Freuder, 1985]. A CSP is *2- j consistent* if for every valid assignment to a pair of variables, there is a set of values for every j additional variables such that the $2+j$ assignments satisfy all of the constraints over these variables. Making a CSP *2- j consistent* can be very expensive as it involves testing all sets of $2+j$ variables, and each test in the worst case takes time exponential in j . In planning problems there can be thousands of variables, so it would be impossible

²Serial actions can be used to represent true concurrency by adding timestamps to the states [Bacchus and Ady, 2001; Pirri and Reiter, 2000].

to make the entire BASE-CSP 2- j consistent even for small j . The contribution of GRAPHPLAN is that it demonstrated a technique that quickly achieves a very effective partial form of 2- j consistency over a limited collection of $2+j$ variables.

We can use the CSP representation to directly compute the binary mutex constraints generated by planning graphs. In this manner we lift the mutex computation to a more general framework where it can be more easily generalized.

Mutexes in GraphPlan are generated by three simple rules: (1) *Base Action Mutexes*. Actions in the same GP-step with interfering effects and preconditions are mutex. (2) *Additional Action Mutexes*. Two actions in the same GP-step are mutex if there is a pair of propositions, one from each action's preconditions, that are mutex at this GP-step. (3) *Propositional Mutexes*. Two propositions, P_i^s and P_j^s , at the same GP-step, are mutex if every action that achieves P_i^s is mutex with every action that achieves P_j^s . We can create all of these binary constraints by testing for very similar conditions in the CSP encoded planning problem:

Base Action Mutexes. These mutexes are already present in the BASE-CSP; they are the GP-concurrency constraints.

Additional Action Mutexes. The GRAPHPLAN condition for detecting additional action mutexes only handles actions with conjunctive preconditions, e.g., STRIPS actions. Thus if we have a pair of preconditions P_i^s for A_i^s and P_j^s for A_j^s , the corresponding precondition constraints $A_i^s \rightarrow P_i^s$ and $A_j^s \rightarrow P_j^s$, and a mutex constraint $\neg P_i^s \vee \neg P_j^s$, a simple 2-2 consistency test with these variables and constraints allows us to derive the new mutex constraint $\neg A_i^s \vee \neg A_j^s$. In fact, we can short-circuit this test by using the presence of the mutex constraint between P_i^s and P_j^s to immediately mark all actions with precondition P_i^s as being mutex with all actions with precondition P_j^s . Furthermore, if an action A_k^s contains both P_i^s and P_j^s in its precondition we obtain the constraint $\neg A_k^s \vee \neg A_k^s$, i.e., we obtain a unary constraint that forces A_k^s to be false. As we will describe below, single valued variables can be used to further reduce the CSP. A similar approach can be followed with more general types of preconditions using more general local consistency tests, e.g., with ADL actions where the GRAPHPLAN test is no longer valid.

Propositional Mutexes. For propositional mutexes we have one added complication, which is that our encoding, unlike planning graphs, does not utilize no-ops. Nevertheless, a general condition for when two propositions P_i^s and P_j^s are mutex can be given by examining the three cases.

1. $\neg P_i^{s-1} \wedge \neg P_j^{s-1}$ holds. In this case for P_i^s and P_j^s both to be true in the next state they must both be created by an action. In particular, neither proposition can remain true by inertia. For a mutex between P_i^s and P_j^s to hold we must have for every action A_i^{s-1} creating P_i^s that $\neg A_i^{s-1} \vee \bigwedge_{A_j \in \text{Create}(P_j)} (A_i^{s-1} \rightarrow \neg A_j^{s-1})$. That is, either the action is not possible, or it must be mutex with

all actions that create P_j^s .³ A similar condition must be satisfied by the actions creating P_i^s .

2. $\neg P_i^{s-1} \wedge P_j^{s-1}$ (or analogously $P_i^{s-1} \wedge \neg P_j^{s-1}$) holds. In this case for both P_i^s and P_j^s to hold P_i^s must be created by an action. For the mutex to hold we must have for every action A_i^{s-1} creating P_i^s that $\neg A_i^{s-1} \vee (A_i^{s-1} \rightarrow \neg P_j^s)$. That is, either the action is not possible or it must delete P_j^s (else P_j^s can be true by inertia).
3. $P_i^{s-1} \wedge P_j^{s-1}$ holds. In this case we must have for every action A^{s-1} that $\neg A^{s-1} \vee (A^{s-1} \rightarrow \neg P_i^{s-1}) \vee (A^{s-1} \rightarrow \neg P_j^{s-1})$. That is, either the action is not possible, or it must delete at least one of P_i or P_j . Note that this condition relies on the fact that at least one action must be executed at every step (the non-null steps constraint).

As in planning graphs, We can use the constraints added at step i to test for new mutex constraints at step $i+1$. The following result can then be easily proved (we omit the proof for reasons of brevity).

Proposition 1 *If the planning graph construction has detected a mutex constraint between two variables V_i^s and V_j^s (action or propositional variables), then after transforming the BASE-CSP to add the above mutex constraints we will have that either (a) both variables in the CSP will have become single valued with not both being true or (b) there will be a binary constraint in the CSP between these variables that prohibits them from both being true.*

That is, the transformed CSP will capture all of the planning graph mutexes. Furthermore, it might contain *more* mutexes than those inferred by the planning graph. In the simplest case this can arise from the last case presented above where every action deletes one of P_i or P_j . This can create a new mutex in the middle of the plan. In planning graphs, on the other hand, once P_i and P_j appear in the graph without a mutex between them, they can never become mutex again (due to the presence of no-ops and the possibility of null steps in the plan).

3.2 Symbolic Reduction of Single Valued Variables

In any CSP $\mathcal{P}$, if $\mathcal{P}$ has a variable V with only a single value a in its domain, we can reduce $\mathcal{P}$ by making the assignment $V \leftarrow a$, to produce a new CSP $\mathcal{P}'$ that does not contain the variable V . $\mathcal{P}'$ has the property that for any solution S of $\mathcal{P}'$, $S \cup \{V \leftarrow a\}$ is a solution to the original $\mathcal{P}$. To reduce $\mathcal{P}$, we replace every constraint mentioning V , e.g., $C(V, V_1, \dots, V_k)$ by a new constraint $C'(V_1, \dots, V_k)$ such that $C'(V_1 \leftarrow x_1, \dots, V_k \leftarrow x_k)$ is true of a tuple of assignments $(x_1, \dots, x_k)$ if and only if $C(V \leftarrow a, V_1 \leftarrow x_1, \dots, V_k \leftarrow x_k)$ is true.

Our translation from a planning problem produces a BASE-CSP in which every constraint C is represented *symbolically* by a logical formula. We can simplify this logical formula directly by replacing the single valued variable V by its value, and then performing standard logical reductions, e.g.,

³Note that if A_i^{s-1} deletes P_j^s then it will be mutex with all actions creating P_j^s via the GP-concurrency constraints.

$\text{FALSE} \wedge \alpha \Rightarrow \text{FALSE}$, $\text{FALSE} \vee \alpha \Rightarrow \alpha$, etc. This yields a new logical formula that represents the reduced constraint C' .

In the case of the BASE-CSP we also know the form of its different constraints. This allows us to precompute many types of constraint reductions, so we can realize many types of single valued variable reductions more efficiently.

The reduction of a single valued variable, or the application of some of the other transformations described below, might generate new single valued variables that can in turn be reduced. It can also interact with the generation of mutex constraints, allowing new mutexes to be detected. For example, say that we have determined that a propositional variable P_j^s must be false. If P_j^s appears in any conjunctive preconditions, e.g., $A_j^s \rightarrow P_j^s$, we will also immediately infer $\neg A_j^s$, i.e., that A_j^s has also become single valued (false). With A_j^s false it can no longer be a candidate for producing some other proposition P_i^{s+1} , and we could possibly infer a new mutex involving P_i^{s+1} since one of the ways of making it true is now blocked. With mutexes, single valued variable reduction subsumes reachability analysis of planning graphs.

Proposition 2 *The variables that remain in the CSP after single valued variables have been reduced, are a subset of the variables appearing in the planning graph.*

4 Beyond GRAPHPLAN

Besides subsuming, and slightly generalizing, the planning graph inferences, further transformations can be utilized on the CSP. These transformations either add more constraints to the CSP or cause further variables to become single valued. In either case, like the planning graph inferences, they are used to make the CSP easier to solve. These additional constraints can also feed into the standard GraphPlan reductions described above, and thus generate even further mutexes.

4.1 Additional Binary Constraints

Graphplan mutexes only prohibit the single pair of values for two variables (TRUE, TRUE). Our representation treats negated propositions and actions in an entirely symmetric fashion. This means that the analysis for mutexes given above will allow us to compute mutexes between any two values. For example, when applied to P_i^s and $\neg P_j^s$, P_i^s being mutex with $\neg P_j^s$ means that P_i^s is “mutin” with P_j^s ; i.e., these variables must both be true or both be false. Such binary constraints can be detected and added to the BASE-CSP.

4.2 Single Valued Variable Reduction beyond GRAPHPLAN Reachability

As demonstrated above, single valued variable reduction is as powerful as the reachability analysis inherent in the planning graph construction. However, in the CSP encoding it is more powerful, obtaining, for example, the following additional reductions:

1. Propositions that are never modified by any action, are propagated without change in the planning graph. Since these propositions have no creators nor deletors, their successor state axiom reduces all of them to being equivalent to their status in the initial state, i.e., they will all

become single valued. Thus they can be reduced from the CSP prior to searching for a solution.

2. If all possible actions at step s create proposition P_i^{s+1} then P_i^{s+1} is forced to be true. Similarly if it is deleted by all possible actions then it is forced to be false. P_i^{s+1} can then be reduced from the CSP. This particular reduction is not possible in planning graphs due to the presence of no-ops.
3. Variables that are forced to be true further reduce all variables that are mutex with them to be false. This does not help reduce a planning graph since in planning graphs mutexes are monotonically decreasing and, other than the initial state variables, variables are never forced to be true.⁴ But in the CSP encoding the previous rules for forcing a variable to be true, and for creating a new mutex when it did not exist at the previous step, enable additional single valued variable reductions.

4.3 Sequence Constraints

There are some additional constraints that can be imposed on plans to rule out obvious inefficiencies. For example, it never makes sense in the plan to immediately follow an action by its inverse (e.g., a load followed by an unload). Adding this constraint allows eliminating different invalid combinations of actions by making mutex any inverse pairs of actions at consecutive time steps.

Blocking the immediate sequencing of inverse actions also allows us to infer additional sequence constraints. Say that actions A_i and A_j are inverses of each other, and that A_i is the only action that produces P_k while A_j is the only action that deletes it. Then another constraint that can be inferred from the mutex between A_i^s and A_j^{s+1} is that when P_k first becomes true, it must remain true for at least one more time step:

$$\neg P_k^s \wedge P_k^{s+1} \rightarrow P_k^{s+2}.$$

There is also a similar constraint for when P_k first becomes false.

Our system automatically detects action pairs that are inverses of each other as well as any predicates over which a sequence constraint can be imposed.

5 CSP-PLAN: Implementation

Based on the transformations explained before, we have developed a planner called CSP-PLAN that will find a solution to a planning problem. CSP-PLAN builds the BASE-CSP incrementally up to step s (beginning with $s = 1$), and then sends it to a CSP Solver for its solution. If the solver reports no solution to it, then the BASE-CSP is constructed for $s+1$. This cycle can continue until some termination condition has been reached or a plan has been found. It is important to notice CSP-PLAN solves these problems incrementally, thus ensuring that it finds a GP-step optimal plan.

The CSP solver used by CSP-PLAN is EFC [Katsirelos and Bacchus, 2001] which allows the use of different techniques

⁴This means that the variable that is mutex with the true variable never appears in the planning graph until the mutex no longer holds.

Problem	GAC	GACCBJ	GACvsCBJ	RecordNoGoods
Bw-large-12	0.46	0.10	0.10	0.11
Bw-large-a	1.44	0.49	0.49	0.50
Rocket-a	21.66	13.74	14.01	10.67
Rocket-b	5.26	4.44	4.48	3.39
DriverLog2	59.13	21.09	21.28	10.38
DriverLog3	0.42	0.23	0.23	0.15
DriverLog4	71.13	29.96	29.96	16.54
DriverLog5	139.98	62.98	63.65	29.59
DriverLog6	0.06	0.05	0.06	0.06

Table 1: Effect of changing the search algorithm on the performance of the CSP solver for planning problems. Experiments were run using the standard CSP heuristic dom+deg.

to solve a CSP problem, including: the ability to create and use tailored heuristics for dynamic variable ordering (DVO), the use of different state of the art CSP algorithms and a novel method to record nogoods. CSP-PLAN uses these methods as follows to efficiently solve general planning problems:

DVO: CSP-PLAN uses a dynamic variable ordering heuristic called BSEARCH, specially tailored for planning problems. This heuristic instantiates first the variables that propagate the most changes in the domain.

Search Algorithm: The CSP algorithm uses a version of GAC propagation [Bacchus, 2000] during backtracking along with nogood recording. In addition, for each constraint with an arity higher than 2, special purpose propagators were developed to increase the efficiency of GAC. It can be noted that although nogood recording is a generic CSP technique, in the form implemented it is strictly more general than the memoization of bad goal agendas described in [Blum and Furst, 1997].

6 Experimental Results

To test the performance of CSP-PLAN, we used STRIPS problems from IPC2 and IPC3 (International Planning Competition 2000/2002). All experiments run on a Xeon 2.40GHz machine with 400MB of RAM. Times are in CPU seconds.

6.1 Effect of CSP techniques on CSP-PLAN

Table 1 shows the effect of a few different CSP algorithms on the performance of CSP-PLAN on some representative problems (using a standard CSP variable ordering heuristic dom+deg).⁵ We can see that GACCBJ and GACvsCBJ⁶ perform over 2 times faster than plain GAC. GACCBJ and GACvsCBJ have comparable results on these problems, but we found that GACvsCBJ performed better on more complex problems, e.g., for Gripper-04, GACvsCBJ finds a plan in 877 secs while GACCBJ required 1,133 secs. The table also shows that nogood recording yields another factor of almost 2 improvement over GACvsCBJ. Hence, we choose GACvsCBJ+RecordNoGood as our default CSP search algorithm.

6.2 Effect of planning specific techniques

Some of the CSP simplification techniques explained in this article use planning related information, e.g. planning specific

⁵Other CSP algorithms were experimented with, but they did not provide adequate performance.

⁶GACvsCBJ is described in [Bacchus, 2000].

Problem	dom+deg	BSEARCH
Bw-large-12	0.10	0.08
Bw-large-a	0.51	0.56
Rocket-a	10.65	0.54
Rocket-b	3.38	1.24
DriverLog2	10.38	0.46
DriverLog3	0.15	0.03
DriverLog4	16.54	0.19
DriverLog5	24.59	0.17
DriverLog6	0.06	0.01

Table 2: Effect of changing the DVO heuristic on the performance of the CSP solver for planning problems. Experiments were run using GACvsCBJ+RecordNoGoods

Problem	No redundant constraints	Inverse mutexes	Sequence constraints
Log-a	0.06	0.09	0.06
Log-b	16.14	15.17	213.50
Rocket-a	1.41	1.48	1.31
Rocket-b	1.85	1.77	2.15
Gripper-01	0.12	0.06	0.10
Gripper-02	17.11	13.20	17.01

Table 3: Effect of adding mutexes between inverse actions and sequence constraints to the BASE-CSP.

DVO heuristics, the addition of redundant constraints, the removal of single valued variables, etc. The addition of redundant binary constraints, particularly mutexes, and the detection and removal of single valued variables has a clear and dramatic effect on the efficiency of CSP-PLAN, even for simple problems. For example, on bw-large-12, the BASE-CSP has 1,289 variables and 10,426 constraints, but only 657 variables and 2,847 constraints after single valued variables are removed. Similarly, without the inferred mutex constraints and the removal of single valued variables, CSP-PLAN required 154.3 sec. to solve the problem. Adding the mutex constraints reduced this time down to 0.33 sec, and removing single valued variables brought the time down even further to 0.04 secs. The effect of some of the other planning specific techniques is described below.

DVO heuristic: The order in which variables are instantiated during search greatly affects the solution times for a CSP. Table 2 shows the difference in performance between the standard CSP heuristic dom+deg, and our BSEARCH heuristic which was designed specifically to take advantage of the planning based structure of the CSPs that CSP-PLAN is solving. These representative problems show that BSEARCH performs much better than dom+deg.

Redundant Constraints: Table 3 shows the performance of CSP-PLAN when mutexes between inverse actions and sequence constraints are added to the BASE-CSP. On these problems we see that there is a slight improvement in CSP-PLAN when mutexes between inverse actions are present. Interestingly, adding sequence constraints did not help. Minor improvements on some problems were negated by significant degradation on others.

6.3 Comparison with similar approaches

The results in Table 4 show how CSP-PLAN (using the same setting for all problems)⁷ compares with GRAPHPLAN [Blum

⁷A number of planning domains could be solved faster with different settings; e.g., the heuristics sometimes performed differently

Problem	CSP-PLAN	GRAPHPLAN	GP-CSP	BLACKBOX	IPP(4.1)
Bw-large-a	0.18	0.21	0.63	0.23	0.06
Bw-large-b	15.18	19.53	106.00	17.50	2.84
Gripper-02	15.36	2.17	62.00	1.87	0.10
Gripper-03	1,328.63	135.00	> 3 hrs.	2,485.00	1.08
Log-a	0.05	1,733.00	47.75	5.50	165.20
Log-b	14.87	675.00	143.00	5.72	120.64
Rocket-a	1.51	46.46	40.50	5.44	5.96
Rocket-b	1.79	128.00	22.68	5.57	21.10
Depots3	11.03	2.41	15.75	2.36	0.22
Depots4	4.59	3.80	82.00	3.72	0.44
DriverLog8	0.78	16.05	0.38	5.70	2.67
DriverLog9	28.99	270.00	90.00	6.27	6.50
DriverLog10	77.75	123.00	22.23	7.89	11.08
ZenoTravel6	0.24	0.64	0.55	0.43	CNP
ZenoTravel7	0.18	0.67	0.61	0.44	CNP
ZenoTravel8	0.52	1.40	1.24	0.90	CNP
ZenoTravel9	37.16	4.84	4.10	3.40	CNP
ZenoTravel10	3.55	9,228.00	28.53	7.52	CNP
Satellite3	0.13	0.23	0.20	0.23	0.04
Satellite4	5.08	271.00	77.02	6.26	4.83
Satellite5	262.72	9,256.00	5,759.00	6.49	175.27
Freecell1	0.10	14.24	183.00	12.90	0.88

Table 4: Comparison of CSP-PLAN with GRAPHPLAN, GP-CSP, BLACKBOX and IPP on a set of problems from the STRIPS domain. “CNP” means could not parse. Lowest times are in bold.

and Furst, 1997], GP-CSP [Do and Kambhampati, 2001], BLACKBOX [Kautz and Selman, 1998] and IPP 4.1 [Koehler *et al.*, 1997]. These planners have in common the construction of a planning graph. The planning graph is either then compiled into a CSP (GP-CSP) or a SAT formula (BLACKBOX), or searched directly for a solution (GRAPHPLAN and IPP 4.1). CSP-PLAN, on the other hand, bypasses the planning graph construction and directly exploits the CSP encoding of the problem as explained above.

It can be seen that CSP-PLAN yields significantly better performance in almost all cases over the standard planning graph planners GRAPHPLAN and GP-CSP. This provides evidence of the effectiveness of the additional constraints and reductions computed by CSP-PLAN. The results against BLACKBOX are mixed. This version of BLACKBOX utilizes the highly engineered Zchaff [Moskewicz *et al.*, 2001] SAT solver. Zchaff utilizes very different heuristics and techniques for learning nogoods (clauses) than CSP-PLAN and is more efficient than our current CSP code on larger problems. Nevertheless, there is considerable scope for making our CSP-PLAN implementation more efficient, and our heuristics are at this stage very simple. Even so, our approach demonstrates its usefulness on problems like gripper.

Similarly against IPP our results are mixed. A key component of IPP is a method for solving planning problems by dividing them into a set of smaller problems using a goal agenda [Koehler, 1998b]. This technique is particularly effective on domains like BlocksWorld, Depots and Gripper, and it can easily be recast in terms of CSPs. Hence, CSP-PLAN could be extended to utilize this technique. On other domains like logistics CSP-PLAN is more effective. This is due particularly to the presence of mutexes between inverse actions in these domains that can be effectively used by CSP-PLAN.

on different domains.

MIPSOpt. Level	Time	Steps	Acts.	Res.
1	3.52	110	175	11,761
3	7.09	102	178	11,505
5	28.02	108	197	13,130
7	80.44	105	221	10,853
9	96.44	104	225	11,630
CSP-PLAN	774.86	75	194	11,345

Table 5: Effect of changing the optimization value for MIPS. Each value in this table is the sum of the results for DriverLog1 to DriverLog10. “Res”: resource usage, “Steps”: number of GP-steps, and “Acts”: number of actions in the plan.

7 Extensions: Planning with Resources

A major advantage of encoding planning problems as CSPs is that we can easily extend the encoding to represent more complex planning problems. For example, planning problems with resources can simply be represented as CSPs with numeric variables. To demonstrate the flexibility of our approach we implemented an extension of CSP-PLAN to deal with such planning problems. Previous approaches based on planning graphs (e.g., [Koehler, 1998a]) have had to develop from scratch methods for dealing with the resource variables in the planning graph, like interval arithmetic and bounds propagation. These techniques are already well known in the CSP field.

To handle planning with resources we introduced the following into CSP-PLAN: (1) Mathematical operation constraints, e.g., $x = z + y$; (2) comparison constraints, e.g., $x > y$; (3) precondition constraints containing comparisons, e.g., $\text{fuel}(\text{truck}) > 0$; (4) successor state constraints to define how the numeric variables could change between states, e.g., if no action affecting it is true, its value cannot change; (5) and finally, additional concurrency constraints to block actions from simultaneously altering a numeric variable. These new constraints were very easy to add to the BASE-CSP, and the reduction transformations described above were used almost unchanged to simplify the BASE-CSP.

We compared CSP-PLAN with MIPS [Edelkamp, 2002]⁸ to evaluate its performance on resource planning problems. MIPS combines two paradigms, model checking and heuristic search, to find totally ordered plans that are then made concurrent using a post-processing procedure. This combination of paradigms introduces additional complexity to the algorithm, in contrast with the uniform paradigm used in our approach. MIPS tries to minimize the number of steps in the final plan, using an optimization parameter that ranges from 1 to 10. Table 5 shows this how parameter affects the quality of the final plans in MIPS. Increasing the parameter causes MIPS to spend more time trying to find plans with fewer steps. However, this search is not always successful. In fact MIPS can spend more time and generate worse results. CSP-PLAN, on the other hand, is an optimal planner always finding the shortest plan in terms of number of GP-steps. This is a slightly different optimization criteria from that used by MIPS. However, even under MIPS’s optimization criteria CSP-PLAN’s plans are much better than those found by MIPS.⁹

Table 6 compares both planners (MIPS running with its optimization flag set to 5). The results in this table show that

⁸MIPS competed in IPC3. No planning graph based planner was

Problem	CSP-PLAN				MIPS			
	Time	Steps	Acts	Res	Time	Steps	Acts	Res
Depots1	0.05	7	11	32	0.04	9	11	32
Depots2	1.55	10	16	43	0.06	10	18	44
DriverLog1	0.01	6	8	1099	0.04	6	8	1099
DriverLog2	0.47	9	23	979	0.08	12	22	1605
DriverLog3	0.08	7	12	907	0.04	7	14	1051
DriverLog4	0.13	7	18	703	5.77	9	19	757
DriverLog5	0.21	8	18	654	1.21	13	22	838
DriverLog6	0.04	5	13	965	0.15	7	15	1667
DriverLog7	0.14	6	18	1060	0.25	7	13	866
DriverLog8	5.24	8	27	1920	15.78	17	29	2217
DriverLog9	73.27	11	32	2485	1.67	14	30	2787
DriverLog10	695.27	8	25	573	3.03	16	25	243

Table 6: Performance comparison of CSP-PLAN and MIPS on resource planning problems. “Res”, “Steps” and “Acts” as explained in table 5.

CSP-PLAN has a performance comparable to MIPS, and in most cases, the quality of the plans (number of steps) is considerably better. However, CSP-PLAN does not scale as well as MIPS, e.g., it takes much longer than MIPS on DriverLog10. But is not surprising since CSP-PLAN is an optimal planner while MIPS is not. Achieving optimality quickly becomes very difficult. It is also important to note that our prototype implementation uses very simple techniques to propagate numeric variables. These techniques are far from the current state of the art in CSPs. Nonetheless, we see that we still obtain results competitive with the competition version of MIPS.

8 Conclusions

Our implementation of CSP-PLAN has demonstrated that the structure of planning problems can be exploited directly in a CSP encoding. By lifting the planning problem to a CSP we obtain a richer and more robust representation for which many sophisticated and effective solution techniques have been developed. In particular, we are able to capture and generalize the important inferences generated by planning graphs directly in the CSP representation. This provides excellent performance for CSP-PLAN in comparison with similar techniques. Additionally, by showing how easily we can extend our representation to planning with resources, we have provided evidence for the benefits of lifting the problem to a CSP encoding.

This work naturally leads to a number of future research topics. For example, the fastest current planners are based on local search. Nevertheless, the heuristics used in these planners utilize notions from the planning graph construction. It seems feasible that instead of planning graphs, CSP techniques could be utilized directly to compute better heuristics. This could be particularly beneficial for computing more informative admissible heuristics. Better heuristics and additional transformations could also be explored. In particular, there is still much scope for improving the performance of this approach. Finally, the approach is well suited to resource usage optimization using branch and bound techniques during the search, and this could be further investigated.

extended to deal with resources in this competition.

⁹Note that neither planner is actually trying to optimize resource usage, just the size of the plan.

References

- [Bacchus and Ady, 2001] Fahiem Bacchus and Michael Ady. Planning with resources and concurrency, a forward chaining approach. In *IJCAI-2001*, pages 417–424.
- [Bacchus, 2000] Fahiem Bacchus. Extending forward checking. In *CP2000*, number 1894 in LNCS, pages 35–51. Springer-Verlag.
- [Blum and Furst, 1997] Avrim Blum and Merrick Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90:281–300, 1997.
- [Do and Kambhampati, 2001] Minh Binh Do and Subbarao Kambhampati. Planning as constraint satisfaction: Solving the planning graph by compiling it into CSP. *Artificial Intelligence*, 132(2):151–182, 2001.
- [Edelkamp, 2002] Stefan Edelkamp. Mixed propositional and numerical planning in the model checking integrated planning system. In *AIPS-2002, Workshop on Temporal Planning*, 2002.
- [Freuder, 1985] E. Freuder. A sufficient condition for backtrack-bounded search. *Journal of the ACM*, 32(4):755–761, 1985.
- [Getoor et al., 1997] L. Getoor, G. Ottosson, M. Fromherz, and B. Carlson. Effective redundant constraints for online scheduling. In *AAAI-1997*, pages 302–307.
- [Katsirelos and Bacchus, 2001] George Katsirelos and Fahiem Bacchus. A library for solving CSPs, 2001. www.cs.toronto.edu/~gkatsi/efc.tar.gz.
- [Kautz and Selman, 1996] Henry Kautz and Bart Selman. Pushing the envelope: planning, propositional logic, and stochastic search. In *AAAI-1996*, pages 1194–1201.
- [Kautz and Selman, 1998] H. Kautz and B. Selman. Blackbox: A new approach to the application of theorem proving to problem solving. *Workshop on Planning as Combinatorial Search*, AIPS-1998.
- [Koehler et al., 1997] J. Koehler, B. Nebel, J. Hoffmann, and Y. Dimopoulos. Extending planning graphs to an ADL subset. In *European Conference on Planning*, pages 273–285, 1997.
- [Koehler, 1998a] J. Koehler. Planning under resource constraints. In *ECAI-1998*, pages 489–493.
- [Koehler, 1998b] J. Koehler. Solving complex planning tasks through extraction of subproblems. In *AIPS-1998*, pages 62–69.
- [Moskewicz et al., 2001] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient sat solver. In *Proc. of the Design Automation Conference (DAC)*, 2001.
- [Pednault, 1989] E. Pednault. ADL: Exploring the middle ground between STRIPS and the situation calculus. In *KR-1989*, pages 324–332.
- [Pirri and Reiter, 2000] F. Pirri and R. Reiter. Planning with natural actions in the situation calculus. In Jack Minker, editor, *Logic-Based Artificial Intelligence*. Kluwer Press, 2000. in press.
- [Reiter, 2001] R. Reiter. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press, 2001.
- [van Beek and Chen, 1999] P. van Beek and X. Chen. CPlan: A constraint programming approach to planning. In *AAAI-1999*, pages 585–590.
- [Vossen et al., 2001] T. Vossen, M. Ball, A. Lotem, and D. Nau. Applying integer programming to AI planning. *Knowledge Engineering Review*, 16:85–100, 2001.

In Defense of PDDL Axioms*

Sylvie Thiébaux

Computer Sciences Laboratory
The Australian National University
Canberra, ACT 0200, Australia
Sylvie.Thiebaux@anu.edu.au

Jörg Hoffmann and Bernhard Nebel

Institut für Informatik
Universität Freiburg
D-79110 Freiburg, Germany
<last-name>@informatik.uni-freiburg.de

Abstract

There is controversy as to whether explicit support for PDDL-like axioms and derived predicates is needed for planners to handle real-world domains effectively. Many researchers have deplored the lack of precise semantics for such axioms, while others have argued that it might be best to compile them away. We propose an adequate semantics for PDDL axioms and show that they are an essential feature by proving that it is impossible to compile them away if we restrict the growth of plans and domain descriptions to be polynomial. These results suggest that adding a reasonable implementation to handle axioms inside the planner is beneficial for the performance. Our experiments confirm this suggestion.

1 Motivation

It is not uncommon for planners to support *derived* predicates, whose truth in the current state is inferred from that of some *basic* predicates via some *axioms* under the closed world assumption. While basic predicates may appear as effects of actions, derived ones may only be used in preconditions, effect contexts and goals. Planners in this family include the partial order planner UCPOP [3], the HTN planner SHOP [18], and the heuristic search planner GPT [4], to cite but a few. The original version of PDDL [17], the International Planning Competition language, also featured such axioms and derived predicates. However, these were never used in competition events, and did not survive PDDL2.1, the extension of the language to temporal planning [8].

This is unfortunate, as the lack of axioms impedes the ability to elegantly and concisely represent real-world domains. Such domains typically require checking complex conditions which are best built hierarchically, from elementary conditions on the state variables to increasingly abstract ones. Without axioms, preconditions and effect contexts quickly become unreadable, or postconditions are forced to include supervenient properties which are just logical consequences of the basic ones—that is when extra actions do not need to be introduced or action descriptions customised. Moreover, axioms provide a natural way of capturing the effects of actions on common real world structures such as paths or flows, as we then need to reason about how the transitive closure of

a basic relation is affected.¹ There is no intuitive way to update transitive closures in the body of a PDDL action, while it is easy to axiomatize them recursively by means of PDDL axioms (see [4] for a power flow example).

The most common criticism of the original PDDL axioms was that their semantics was ill-specified, and that the conditions under which the truth of the derived predicates could be uniquely determined were unclear. We remedy this by providing a clear semantics for PDDL axioms while remaining consistent with the original description in [17]. In particular, we identify conditions that are sufficient to ensure that the axioms have an unambiguous meaning, and explain how these conditions can efficiently be checked.

Another common view is that axioms are a non-essential language feature which it might be better to compile away than to deal with explicitly, compilation offering the advantage of enabling the use of more efficient, simple, standard planners without specific treatment [10; 9; 7]. We bring new insight to this issue. We give evidence that axioms add significant expressive power to PDDL. We take “expressive power” to be a measure of how concisely domains and plans can be expressed in a formalism and use the notion of compilability to analyse that [19]. As it turns out, axioms are an essential feature because it is impossible to compile them away—provided we require the domain descriptions to grow only polynomially and the plans to grow only polynomially in the size of the original plans and domain descriptions. Of course, if we allow for exponential growth, then compilations become possible and we specify one such transformation, which, unlike those previously published [10; 9; 7], works without restriction. However, the above mentioned results suggest that it might be much more efficient to deal with axioms inside the planner than to compile them away. In fact, our experiments with FF [11] suggest that adding even a simple implementation of axioms to a planner clearly outperforms the original version of the planner solving the compiled problem.

2 Syntax and Semantics

We remain in the sequential planning setting of PDDL2.1 level 1, which is essentially that of the version of PDDL with ADL actions used in the 2000 planning competition. See [2] for a description of the syntax. For clarity we do not consider types. Although we see axioms with conditions on numeric

*The full proofs of the theorems appear in the techreport: <http://cs1.anu.edu.au/~thiebaux/papers/trarp0103.pdf>.

¹In that respect, PDDL axioms offer advantages over the use of purely logical axioms as in the original version of STRIPS [14].

fluents, such as those featured in PDDL2.1 level 2, as very desirable, we do not consider them here for simplicity.

Let $\mathcal{B}$ and $\mathcal{D}$ be two sets of predicate symbols with $\mathcal{B} \cap \mathcal{D} = \emptyset$, called the set of basic and derived predicates, respectively. Symbols in $\mathcal{D}$ are not allowed to appear in the initial state description and in atomic effects of actions, but may appear in preconditions, effect contexts, and goals. The domain description features a set of axioms A . These have the form $(: \text{derived } (d \ ?\vec{x}) (f \ ?\vec{x}))$, where $d \in \mathcal{D}$, and where f is a first-order formula built from predicate symbols in $\mathcal{B} \cup \mathcal{D}$ and whose free variables are those in the vector $\vec{x}$.

Intuitively, an axiom $(: \text{derived } (d \ ?\vec{x}) (f \ ?\vec{x}))$ means that when $(f \ ?\vec{x})$ is true at the specified arguments in a given state, we should *derive* that $(d \ ?\vec{x})$ is true at those arguments in that same state. Unlike traditional implications, these derivations are not to be contraposed (the negation of f is not derived from the negation of d), and what cannot be derived as true is false (closed world assumption). Because of the closed world assumption, there is never any need to explicitly derive negative literals, so the constraint that the consequent of axioms be *positive* literals does not make us lose generality.

In sum, axioms are essentially (function free) logic program statements [15]. For example, from the basic predicate `on` and the predicate `holding` in Blocks World, we can define the predicate `clear`, as follows:

```
(:derived (clear ?x)
  (and (not (holding ?x))
    (forall (?y) (not (on ?y ?x))))))
```

Another classic is `above`, the transitive closure of `on`, e.g.:

```
(:derived (above ?x ?y)
  (or (on ?x ?y)
    (exists (?z) (and (above ?x ?z)
                      (above ?z ?y)))))
```

In a planning context, it is natural and convenient to restrict attention to so-called *stratified* axiom sets—stratified logic programs avoid unsafe use of negation and have an unambiguous, well-understood semantics [1]. The idea behind stratification is that some derived predicates should first be defined in terms of the basic ones possibly using negation, or in terms of themselves (allowing for recursion) but *without* using negation. Next, more abstract predicates can be defined building on the former, possibly using their negation, or in terms of themselves but without negation, and so on. Thus, a stratified axiom set is partitionable into strata, in such a way that the negation normal form² (NNF) of the antecedent of an axiom defining a predicate belonging to a given stratum uses arbitrary occurrences of predicates belonging to strictly lower strata and *positive* occurrences of predicates belonging to the same stratum. Basic predicates may be used freely.

Definition 1 An axiom set A is stratified iff there exists a partition (stratification) of the set of derived predicates $\mathcal{D}$ into (non-empty) subsets $\{\mathcal{D}_i, 1 \leq i \leq n\}$ such that for all $(: \text{derived } (d_i \ ?\vec{x}) (f \ ?\vec{x})) \in A$:

1. if d_j appears in $\text{NNF}(f \ ?\vec{x})$, then $d_i \in \mathcal{D}_i$ and $d_j \in \mathcal{D}_j$ such that $j \leq i$,
2. if d_j appears negated in $\text{NNF}(f \ ?\vec{x})$, then $d_i \in \mathcal{D}_i$ and $d_j \in \mathcal{D}_j$ such that $j < i$.

²In a formula in NNF, negation occurs only in literals.

Note that any stratification $\{\mathcal{D}_i, 1 \leq i \leq n\}$ of $\mathcal{D}$ induces a stratification $\{A_i, 1 \leq i \leq n\}$ of A in the obvious way: $A_i = \{(: \text{derived } (d_i \ ?\vec{x}) (f \ ?\vec{x})) \in A \mid d_i \in \mathcal{D}_i\}$. Note also that when no derived predicate occurs negated in the NNF of the antecedent of any axiom, a single stratum suffices. Several planning papers have considered this special case [10; 9; 7].

Working through the successive strata, applying axioms in any order within each stratum until a fixed point is reached and then only proceeding to the next stratum, always leads to the same final fixed point independently of the chosen stratification [1, p. 116]. It is this final fixed point which we take to be the meaning of the axiom set.

We now spell out the semantics formally. Since we have a finite domain and no functions, we identify the objects in the domain with the ground terms (constants) that denote them, and states with finite sets of ground atoms. More precisely, a state is taken to be a set of ground *basic* atoms: the derived ones will be treated as elaborate descriptions of the basic state. In order to define the semantics, however, we first need to consider an extended notion of “state” consisting of a set S of basic atoms and an arbitrary set D of atoms in the derived vocabulary. The modeling conditions for extended states are just the ordinary ones of first order logic, as though there were no relationship between S and D . Where $?\vec{x}$ denotes a vector of variables and $\vec{t}$ denotes a vector of ground terms, we define:

Definition 2

$$\begin{aligned} \langle S, D \rangle &\models (b \ \vec{t}) \text{ for } b \in \mathcal{B} \text{ iff } (b \ \vec{t}) \in S \\ \langle S, D \rangle &\models (d \ \vec{t}) \text{ for } d \in \mathcal{D} \text{ iff } (d \ \vec{t}) \in D \\ \langle S, D \rangle &\models (\text{not } f) \text{ iff } \langle S, D \rangle \not\models f \\ \langle S, D \rangle &\models (\text{and } f_1 \ f_2) \text{ iff } \langle S, D \rangle \models f_1 \text{ and } \langle S, D \rangle \models f_2 \\ \langle S, D \rangle &\models (\text{or } f_1 \ f_2) \text{ iff } \langle S, D \rangle \models f_1 \text{ or } \langle S, D \rangle \models f_2 \\ \langle S, D \rangle &\models (\text{forall } (?x) (f \ ?x)) \text{ iff } \langle S, D \rangle \models (f \ \vec{t}) \text{ for all } \vec{t} \\ \langle S, D \rangle &\models (\text{exists } (?x) (f \ ?x)) \text{ iff } \langle S, D \rangle \models (f \ \vec{t}) \text{ for some } \vec{t} \end{aligned}$$

Applying axiom $a \equiv (: \text{derived } (d \ ?\vec{x}) (f \ ?\vec{x}))$ in a state S augmented with derived atoms D , results in the set $\llbracket a \rrbracket(S, D)$ of further derived atoms:

Definition 3

$$\llbracket a \rrbracket(S, D) = \{ (d \ \vec{t}) \mid \langle S, D \rangle \models (f \ \vec{t}), \vec{t} \text{ is ground} \}$$

Given this, we associate stratum A_i with the function $\llbracket A \rrbracket_i$ which maps a given basic state S to the least fixed point attainable by applying the axioms in A_i starting from the extended state consisting of S and of the set of ground derived atoms returned at the previous stratum by $\llbracket A \rrbracket_{i-1}$. The stratified axiom set A denotes the function $\llbracket A \rrbracket = \llbracket A \rrbracket_n$:

Definition 4 Let $\{A_i, 1 \leq i \leq n\}$ be an arbitrary stratification for a stratified axiom set A . For each state S , let:

$$\begin{aligned} \llbracket A \rrbracket_0(S) &= \emptyset, \text{ and for all } 1 \leq i \leq n \\ \llbracket A \rrbracket_i(S) &= \bigcap \left\{ D \mid \bigcup_{a \in A_i} \llbracket a \rrbracket(S, D) \cup \llbracket A \rrbracket_{i-1}(S) \subseteq D \right\} \end{aligned}$$

Then $\llbracket A \rrbracket(S)$ is defined as $\llbracket A \rrbracket_n(S)$.

Finally, given a stratified axiom set A , we write $S \models_A f$ to indicate that a formula f composed of both basic and derived predicates holds in state S :

Algorithm 1 Stratification

```

1. function STRATIFY( $\mathcal{D}, A$ )
2.   for each  $i \in \mathcal{D}$  do
3.     for each  $j \in \mathcal{D}$  do
4.        $R[i, j] \leftarrow 0$ 
5.   for each  $(: \text{derived } (j \text{ ? } \bar{x}) (f \text{ ? } \bar{x})) \in A$  do
6.     for each  $i \in \mathcal{D}$  do
7.       if  $i$  occurs negatively in  $\text{NNF}(f \text{ ? } \bar{x})$  then
8.          $R[i, j] \leftarrow 2$ 
9.       else if  $i$  occurs positively in  $\text{NNF}(f \text{ ? } \bar{x})$  then
10.         $R[i, j] \leftarrow \text{MAX}(1, R[i, j])$ 
11.   for each  $j \in \mathcal{D}$  do
12.     for each  $i \in \mathcal{D}$  do
13.       for each  $k \in \mathcal{D}$  do
14.         if  $\text{MIN}(R[i, j], R[j, k]) > 0$  then
15.            $R[i, k] \leftarrow \text{MAX}(R[i, j], R[j, k], R[i, k])$ 
16.   if  $\forall i \in \mathcal{D} R[i, i] \neq 2$  then
17.     stratification  $\leftarrow \emptyset$ , remaining  $\leftarrow \mathcal{D}$ , level  $\leftarrow 1$ 
18.     while remaining  $\neq \emptyset$  do
19.       stratum  $\leftarrow \emptyset$ 
20.       for each  $j \in \text{remaining}$  do
21.         if  $\forall i \in \text{remaining } R[i, j] \neq 2$  then
22.           stratum  $\leftarrow \text{stratum} \cup \{j\}$ 
23.       remaining  $\leftarrow \text{remaining} \setminus \text{stratum}$ 
24.       stratification  $\leftarrow \text{stratification} \cup \{(\text{level}, \text{stratum})\}$ 
25.       level  $\leftarrow \text{level} + 1$ 
26.   return stratification
27.   else fail
  
```

Definition 5 $S \models_A f$ iff $\langle S, \llbracket A \rrbracket(S) \rangle \models f$

This modeling relation is used when applying an action in state S to check preconditions and effect contexts, and to determine whether S satisfies the goal. This is the only change introduced by the axioms into the semantics of PDDL and completes our statement of the semantics. The rest carries over verbatim from [2].

Checking that the axiom set in a domain description is stratified and computing a stratification can be done in polynomial time in the size of the domain description, using Algorithm 1. The algorithm starts by building a $|\mathcal{D}| \times |\mathcal{D}|$ matrix³ R such that $R[i, j] = 2$ when it follows from the axioms that predicate i 's stratum must be strictly lower than predicate j 's stratum, $R[i, j] = 1$ when i 's stratum must be lower than j 's stratum but not necessarily strictly, and $R[i, j] = 0$ when there is no constraint between the two strata (lines 2-15). R is first filled with the values encoding the status (strict or not) of the base constraints obtained by direct examination of the axioms (lines 5-10). Then, the consequences of the base constraints are computed, similarly as one would compute the transitive closure of a relation (lines 11-15). There exists a stratification iff the strict relation encoded in R is irreflexive, that is iff $R[i, i] \neq 2$ for all $i \in \mathcal{D}$ (line 16). In that case, the stratification corresponding to the smallest pre-order consistent with R is extracted, i.e. predicates are put in the lowest stratum consistent with R (lines 17-26).

3 Axioms Add Significant Expressive Power

It is clear that axioms add something to the expressive power of PDDL. In order to determine how much power is added, we will use the *compilability approach* [19]. Basically, what we want to determine is how concisely a planning task can be represented if we compile the axioms away. Furthermore, we want to know how long the corresponding plans in the compiled planning task will become.

³By $|\cdot|$ we denote the cardinality of a set.

In the following, we take a PDDL $_{\mathcal{X}}$ planning domain description to be a tuple $\Delta = \langle \mathcal{C}, \mathcal{B}, \mathcal{D}, A, O \rangle$, where $\mathcal{C}$ is the set of constant symbols, $\mathcal{B}$ is the set of basic predicates, $\mathcal{D}$ is the set of derived predicates, A is a stratified axiom set as in Definition 1, and O is a set of action descriptions (with the mentioned restriction on the appearance in atomic effects of the symbols in $\mathcal{D}$). A PDDL $_{\mathcal{X}}$ *planning instance* or *task* is a tuple $\Pi = \langle \Delta, \mathcal{I}, \mathcal{G} \rangle$, where Δ is the domain description, and $\mathcal{I}$ and $\mathcal{G}$ are the initial state (a set of ground basic atoms) and goal descriptions (a formula), respectively. The result of applying an action in a (basic) state and what constitutes a valid plan (sequence of actions) for a given planning task are defined in the usual way [2], except that the modeling relation in Definition 5 is used in place of the usual one. By a PDDL domain description and planning instances we mean those without any axioms and derived predicates, i.e., a PDDL domain description has the form $\langle \mathcal{C}, \mathcal{B}, \emptyset, \emptyset, O \rangle$.

We now use *compilation schemes* [19] to translate PDDL $_{\mathcal{X}}$ domain descriptions to PDDL domain descriptions. Such schemes are functions that translate domain descriptions between planning formalisms without any restriction on their computational resources but the constraint that the target domain should be only polynomially larger than the original.⁴

Definition 6 A compilation scheme from $\mathcal{X}$ to $\mathcal{Y}$ is a tuple of functions $\mathbf{f} = \langle f_{\delta}, f_i, f_g \rangle$ that induces a function F from $\mathcal{X}$ -instances $\Pi = \langle \Delta, \mathcal{I}, \mathcal{G} \rangle$ to $\mathcal{Y}$ -instances $F(\Pi)$ as follows:

$$F(\Pi) = \langle f_{\delta}(\Delta), \mathcal{I} \cup f_i(\Delta), \mathcal{G} \wedge f_g(\Delta) \rangle$$

and satisfies the following conditions:

1. there exists a plan for Π iff there exists a plan for $F(\Pi)$,
2. and the size of the results of f_{δ} , f_i , and f_g is polynomial in the size of their argument Δ .

In addition, we measure the size of the corresponding plans in the target formalism.⁵

Definition 7 If a compilation scheme $\mathbf{f}$ has the property that for every plan P solving an instance Π , there exists a plan P' solving $F(\Pi)$ such that $\|P'\| \leq c \times \|P\| + k$ for positive integer constants c and k , then $\mathbf{f}$ is a compilation scheme preserving plan size linearly, and if $\|P'\| \leq p(\|P\|, \|\Pi\|)$ for some polynomial p , then $\mathbf{f}$ is a compilation scheme preserving plan size polynomially.

From a practical point of view, one can regard compilability preserving *plan size linearly* as an indication that the target formalism is *at least as expressive* as the source formalism. Conversely, if a *super-linear* blowup of the plans in the target formalism is required, this indicates that the source formalism is *more expressive* than the target formalism—a planning algorithm for the target formalism would be forced to generate significantly longer plans for compiled instances, making it probably infeasible to solve such instances. If plans are required to grow even *super-polynomially*, then the increase of expressive power must be dramatic. Incidentally, exponential growth of plan size is necessary to compile axioms away.

⁴We use here a slightly simplified definition of compilability.

⁵The size of an instance, domain description, plan, etc. is denoted by $\|\cdot\|$.

In order to investigate the compilability between PDDL and PDDL_χ, we will analyze restricted planning problems such as the *1-step planning problem* and the *polynomial step planning problem*. The former is the problem of whether there exists a 1-step plan to solve a planning task, the latter is the problem whether there exists a plan polynomially sized (for some fixed polynomial) in the representation of the domain description. From the results on the computational complexity of pure DATALOG and DATALOG with stratified negation [6], the next theorem is immediate.

Theorem 1 *The 1-step planning problem for PDDL_χ is EXPTIME-complete, even if all axioms are in pure DATALOG.*

If we now consider PDDL planning tasks, it turns out that the planning problem is considerably easier, even if we allow for polynomial length plans. Since guessing a plan of polynomial size and verifying it can easily be done in polynomial space, the polynomial step PDDL planning problem is obviously in PSPACE. Taking in addition Vardi's [20] result into account that first-order query evaluation over a finite database is PSPACE-complete, hardness follows as well.

Theorem 2 *The polynomial step planning problem for PDDL is PSPACE-complete.*

From these two statements it follows immediately that it is very unlikely that there exists a *polynomial time* compilation scheme from PDDL_χ to PDDL preserving plan size polynomially. Otherwise, it would be possible to solve all problems requiring exponential time in polynomial space, which is considered as quite unlikely. As argued, however, by Nebel [19], if we want to make claims about *expressiveness*, then we should not take the computational resources of the compilation scheme into account but allow for computationally unconstrained transformations. Interestingly, even allowing for such unconstrained compilation schemes changes nothing.

Theorem 3 *Unless EXPTIME = PSPACE, there is no compilation scheme from PDDL_χ (even restricted to pure DATALOG axioms) to PDDL preserving plan size polynomially.*

Proof Sketch. We use a proof idea similar to the one Kautz and Selman [13] used to prove that approximations of logical theories of a certain size are not very likely to exist. By using a DATALOG theory in order to describe all instances of the *linearly bounded alternating Turing machine* acceptance problem up to a certain size, which in its general form is EXPTIME-complete [5], we get a *polynomial advice string* [12] if a compilation scheme from PDDL_χ to PDDL preserving plan size polynomially exists. This would imply that EXPTIME ⊆ PSPACE/poly. However, by Karp and Lipton's [12] results, this implies that EXPTIME = PSPACE. ■

4 Compilations with Exponential Results

While it is impossible to find a concise equivalent PDDL planning instance that guarantees short plans, it is possible to come up with a poly-size instance which may have exponentially longer plans in the worst case. Such compilation schemes have been described by e.g. Gazen and Knoblock [10] and Garagnani [9] under severe restrictions on the use of negated derived predicates. Specifically, these schemes do not work if negated derived predicates appear anywhere in the

Figure 1 PDDL instances induced by **f**

```

1. (: predicates ; all predicates in  $\mathcal{B} \cup \mathcal{D}$ 
2.      (done1) ... (donen)
3.      (fixed0) ... (fixedn)
4.      (new))
   for each  $i \in \{1, \dots, n\}$ 
5. (: action stratumi
6.      : parameters ()
7.      : precondition (and (fixedi-1) (not (fixedi)))
8.      : effect (and (donei)
9.                  (forall ( $\vec{x}_{i,1}$ )
10.                     (when (and ( $f_{i,1} ?\vec{x}_{i,1}$ ) (not ( $d_{i,1} ?\vec{x}_{i,1}$ )))
11.                         (and ( $d_{i,1} ?\vec{x}_{i,1}$ ) (new))))))
12.                  ...
13.                  (forall ( $\vec{x}_{i,|A_i|}$ )
14.                     (when (and ( $f_{i,|A_i|} ?\vec{x}_{i,|A_i|}$ ) (not ( $d_{i,|A_i|} ?\vec{x}_{i,|A_i|}$ )))
15.                         (and ( $d_{i,|A_i|} ?\vec{x}_{i,|A_i|}$ ) (new))))))
16. (: action fixpointi
17.      : parameters ()
18.      : precondition (donei)
19.      : effect (and (when (not(new)) (fixedi))
20.                  (not(new))
21.                  (not (donei))))
   for each  $o \in \mathcal{O}$ 
22. (: action NAME( $o$ )
23.      : parameters PARAMETERS( $o$ )
24.      : precondition (and PRECONDITION( $o$ ) (fixedk))
25.      : effect (and EFFECT( $o$ )
26.                (not (fixedm)) ... (not (fixedn))
27.                (not (donem)) ... (not (donen))
28.                (forall ( $\vec{x}_{m,1}$  (not ( $d_{m,1} ?\vec{x}_{m,1}$ )))
29.                    ...
30.                    (forall ( $\vec{x}_{n,|A_n|}$  (not ( $d_{n,|A_n|} ?\vec{x}_{n,|A_n|}$ ))))))
   Where  $k = \max(\{i \mid \text{some } d_{i,j} \text{ occurs in PRECONDITION}(o)\} \cup \{0\})$  and
    $m = \min(\{i \mid \text{a predicate in some } f_{i,j} \text{ is modified in EFFECT}(o)\} \cup \{n+1\})$ 
31. (: init  $\mathcal{I} \cup \{\text{fixed}_0\}$ )
32. (: goal (and  $\mathcal{G}$  (fixedn)))

```

planning task,⁶ and the latter scheme [9] is further restricted to pure DATALOG axioms.

An interesting contrasting approach is that of Davidson and Garagnani [7]. They propose to compile pure DATALOG axioms solely into conditional effects, which means that the resulting plans will have exactly the same length. However, as is implied by Theorem 3, their domain description suffers a super-polynomial growth.

We now specify a generally applicable compilation scheme producing poly-size instances, which we will use as a baseline in our performance evaluation. In contrast to the schemes mentioned above, it complies with the stratified semantics specified in Section 2 while dealing with negated occurrences of derived predicates anywhere in the planning task.

Theorem 4 *There exists a polynomial time compilation scheme $\mathbf{f} = \langle f_\delta, f_i, f_g \rangle$, such that for every PDDL_χ domain description $\Delta = \langle \mathcal{C}, \mathcal{B}, \mathcal{D}, A, O \rangle$: $\|f_i(\Delta)\| = c_1$ and $\|f_g(\Delta)\| = c_2$ for some constants c_1 and c_2 , and $f_\delta(\Delta) = \langle \mathcal{C}, \mathcal{B}', \emptyset, \emptyset, O' \rangle$ is a PDDL domain with $|\mathcal{B}'| \leq |\mathcal{B}| + 3 \cdot |\mathcal{D}| + 2$ and with $\|O'\| \leq p(\|O\|, \|A\|)$ for some polynomial p .*

Proof Sketch. Figure 4 shows the main elements of the PDDL instances induced by **f**. **f** computes a stratification $\{A_i, 1 \leq i \leq n\}$ of the set of axioms A , as explained in Section 2, where in stratum i , each axiom $a_{i,j}$ is of the form $(: \text{derived } (d_{i,j} ?\vec{x}_{i,j}) (f_{i,j} ?\vec{x}_{i,j}))$ for $1 \leq j \leq |A_i|$. **f**

⁶This remains true even if negation is compiled away as per the Gazen and Knoblock method [10].

encodes each stratum as an extra action stratum_i (see lines 5-15 in Figure 4) which applies all axioms $a_{i,j}$ at this stratum in parallel, records that this was done (done_i) and whether anything new (new) was derived in doing so. Each $a_{i,j}$ is encoded as a universally quantified and conditional effect of stratum_i —see lines 9-15. To ensure that the precedence between strata is respected, stratum_i is only applicable when the fixed point for the previous stratum has been reached (i.e. when fixed_{i-1}) and the fixed point for the current stratum has not (i.e. when $(\text{not } (\text{fixed}_i))$)—see line 7. $\mathbf{f}$ encodes the fixpoint computation at each stratum i using an extra action fixpoint_i , which is applicable after a round of one or more applications of stratum_i (i.e., when done_i is true), asserts that the fixed point has been reached (i.e. fixed_i) whenever nothing new has been derived during this last round, and resets new and done_i for the next round—see lines 16-21. Next, the precondition and effect of each action description $o \in O$ are augmented as follows (see lines 22-30). Let $0 \leq k \leq n$ be the highest stratum of any derived predicate appearing in the precondition of o , or 0 if there is no such predicate. Before applying o , we must make sure that the fixed point for that stratum has been computed by adding fixed_k to the precondition. Similarly, let $1 \leq m \leq n+1$ be the lowest stratum such that some predicate in the antecedent of some axiom in A_m is modified in the effect of o , or $n+1$ if there is none. After applying o , we may need to re-compute the fixed points for the strata above m , that is, the effect must reset fixed , done , and the value of all derived propositions, at strata m and above. Finally, fixed_0 holds initially, and the goal requires fixed_n to be true. The fact that $\mathbf{f}$ preserves domain description size polynomially, and the bounds given in theorem 4, follow directly from the construction. ■

It is obvious that a plan P for a planning task Π can be recovered from a plan P' for the compiled planning task $F(\Pi)$, by simply stripping all occurrences of stratum and fixpoint actions. In the worst case of course, there is no polynomial p such that $\|P'\| \leq p(\|P\|, \|\Pi\|)$. Indeed, the worst-case is obtained when, initially and after each action from P , all derived predicates need to be (re)computed and only one proposition is ever derived per application of stratum_i actions. Even if the planner is able to interleave as few fixpoint_i actions as possible with the stratum_i actions, this still leads to a plan of length $\|P'\| = \|P\| + (\|P\| + 1)(\sum_{i=1}^n (\overline{\mathcal{D}}_i + 3)) = \|P\| + (\|P\| + 1)(3n + |\overline{\mathcal{D}}|)$, where $\overline{\mathcal{D}}$ denotes the set of all instances of predicates in $\mathcal{D}$. Observe that $\overline{\mathcal{D}}$ is not polynomially bounded in $|\mathcal{D}|$ and $|\mathcal{C}|$.

5 Planning: With or Without Axioms?

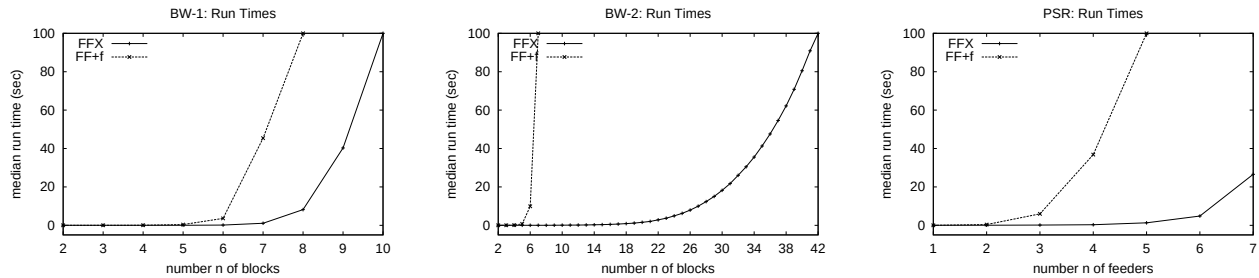
The absence of a polynomial time compilation scheme preserving plan size linearly not only indicates that axioms bring (much needed) expressive power, but it also suggests that extending a planner to explicitly deal with axioms may lead to much better performance than using a compilation scheme with the original version of the planner. To confirm this hypothesis, we extended the FF planner [11] with a straightforward implementation of axioms—we call this extension $\text{FF}_{\mathcal{X}}$ —and compared results obtained by $\text{FF}_{\mathcal{X}}$ on PDDL $_{\mathcal{X}}$ instances with those obtained by FF on the PDDL instances produced via compilation with $\mathbf{f}$.

$\text{FF}_{\mathcal{X}}$ transforms each axiom $(\text{derived}(d \text{ ?}\vec{x})(f \text{ ?}\vec{x}))$ into an operator with parameters $(\text{?}\vec{x})$, precondition $(f \text{ ?}\vec{x})$ and effect $(d \text{ ?}\vec{x})$, with a flag set to distinguish it from a “normal” operator. During the relaxed planning process that FF performs to obtain its heuristic function, the axiom actions are treated as normal actions and can be chosen for inclusion in a relaxed plan. However, the heuristic value only counts the number of *normal* actions in the relaxed plan. During the forward search FF performs, only normal actions are considered; after each application of such an action, the axiom actions are applied so as to obtain the successive fixed points associated with the stratification computed by Algorithm 1.

One domain we chose for our experiments is the usual Blocks World (BW) with 4 operators. In contrast to most other common benchmarks, in BW there is a natural distinction between basic and derived predicates; in particular BW is the only common benchmark domain we are aware of where the stratification of the axioms requires more than one stratum. The basic predicates are *on* and *ontable*, and the derived ones are *above* and *holding* (stratum 1), as well as *clear* and *handempty* (stratum 2) whose axiomatisations use the negation of *holding*. *above* is only used in goal descriptions. For the experiment labelled BW-1 in the figures below, we generated 30 random initial states for each size $n = 2 \dots 10$ and took the goal that any block initially on the table had to be above all those that were initially not. Note that expressing the resulting goal using *on* and *ontable* would require exponential space, highlighting once more the utility of derived predicates. As shown in the figure, the median run-time of $\text{FF}_{\mathcal{X}}$ shows a significant improvement over that of $\text{FF}+\mathbf{f}$. The plans found by $\text{FF}+\mathbf{f}$ in this experiment were an order of magnitude longer than those found by $\text{FF}_{\mathcal{X}}$. The experiment labelled BW-2 shows, for $n = 2 \dots 42$, the special case of those instances for which the initial state has only one tower. Here the improvement in run time is dramatic, as $\text{FF}_{\mathcal{X}}$ finds the optimal plans whose length is only linear in n .

Another domain we ran experiments on is the challenging Power Supply Restoration (PSR) benchmark [4], which is derived from a real-world problem in the area of power distribution. The domain description requires a number of complex, recursive, derived predicates to axiomatize the power flow, [4]. We considered a version of the benchmark without any uncertainty for which the goal is to resupply all resuppliable lines. For each number $n = 1$ to 7 feeders, we generated 100 random networks with a maximum of 3 switches per feeder and with 30% faulty lines. The third figure above compares the median run times of $\text{FF}_{\mathcal{X}}$ and $\text{FF}+\mathbf{f}$ as a function of n . Again the improvement in performance resulting from handling axioms explicitly is undeniable. In this experiment, the plan length does not vary much with n : with our parameters for the random instances generation, it is clustered around 5 actions for PDDL $_{\mathcal{X}}$ instances, and around 50 for the compiled instances. Yet this makes all the difference between what is solvable in reasonable time and what is not.

Although the domains in these experiments are by no means chosen to show off the worst-case for the compilation scheme, they nevertheless illustrate its drawbacks. The difference of performance we observe is due to the facts that compilation increases the branching factor, increases the plan length, and obscures the computation of the heuristic.



Other published compilation schemes [10; 9] are not applicable to the above domains whose descriptions involve negated derived predicates. The exponential space transformation by Davidson and Garagnani [7] is applicable to BW, but was unable to cope with problems larger than $n = 4$ because it substitutes for their definitions all occurrences of non-recursive derived predicates until none remains. This turns preconditions into ADL constructs that quickly become too complex for FF's pre-processing step to compile them away in reasonable time, in difference to the experiments described above where pre-processing time was negligible.

6 Conclusion

As reflected by recent endeavours in the international planning competitions, there is a growing (and, in our opinion, desirable) trend towards more realistic planning languages and benchmark domains. In that context, it is crucial to determine which additional language features are particularly relevant. The main contribution of this paper is to give theoretical and empirical evidence of the fact that axioms *are* important, from both an expressivity and efficiency perspective. In addition, we have provided a clear formal semantics for PDDL axioms, identified a general and easily testable criterion for axiom sets to have an unambiguous meaning, and given a compilation scheme which is more generally applicable than those previously published (and also more effective in conjunction with forward heuristic search planners like FF).

Future work will include more extensive empirical studies involving a more elaborate treatment of axioms within FF and planners of different types, as well as the extension of derived predicates and axioms to the context of the numerical and temporal language features recently introduced with PDDL 2.1. Axioms have long been an integral part of action formalisms in the field of reasoning about action and change where, much beyond the inference of derived predicate considered here, they form the basis for elegant solutions to the frame and ramification problems, see e.g. [16]. It is our hope that the adoption of PDDL axioms will eventually encourage the planning community to make greater use of these formalisms.

Acknowledgements

Thanks to Blai Bonet, Marina Davidson, Stefan Edelkamp, Maria Fox, John Lloyd, and John Slaney for fruitful discussions which helped to improve this paper.

References

- [1] K. Apt, H. Blair, and A. Walker. Towards a theory of declarative knowledge. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*. Morgan Kaufmann, 1987.
- [2] F. Bacchus. Subset of PDDL for the AIPS2000 planning competition. www.cs.toronto.edu/aips2000, 2000.
- [3] A. Barrett, D. Christianson, M. Friedman, C. Kwok, K. Golden, S. Penberthy, Y. Sun, and D. Weld. UCPOP User's Manual. Technical Report 93-09-06d, The University of Washington, Computer Science Department, 1995.
- [4] B. Bonet and S. Thiébaux. GPT Meets PSR. In *Proc. ICAPS*, to appear, 2003.
- [5] A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer. Alternation. *Journal of the ACM*, 28(1):114–133, 1981.
- [6] E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity and expressive power of logic programming. *ACM Computing Surveys*, 33(3):374–425, 2001.
- [7] M. Davidson and M. Garagnani. Pre-processing planning domains containing language axioms. In *Proc. UK PlanSIG workshop*, 2002.
- [8] M. Fox and D. Long. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. www.dur.ac.uk/d.p.long/IPC/pddl.html, 2002.
- [9] M. Garagnani. A correct algorithm for efficient planning with preprocessed domain axioms. In *Research and Development in Intelligent Systems XVII*. Springer, 2000.
- [10] C. Gazeau and C. Knoblock. Combining the expressivity of UCPOP with the efficiency of Graphplan. In *Proc. ECP*, 1997.
- [11] J. Hoffmann and B. Nebel. The FF planning system: Fast plan generation through heuristic search. *JAIR*, 14:253–302, 2001.
- [12] R. M. Karp and R. J. Lipton. Turing machines that take advice. *L'Enseignement Mathématique*, 28:191–210, 1982.
- [13] H. A. Kautz and B. Selman. Forming concepts for fast inference. In *Proc. AAAI*, 1992.
- [14] V. Lifschitz. On the semantics of STRIPS. In *Proc. Workshop on Reasoning about Actions and Plans*, pages 1–9, 1987.
- [15] J. Lloyd. *Foundations of Logic Programming*. Springer, 1993.
- [16] N. McCain and H. Turner. A causal theory of ramifications and qualifications. In *Proc. IJCAI*, 1995.
- [17] D. McDermott. PDDL – The Planning Domain Definition Language. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, 1998.
- [18] D. Nau, Y. Cao, A. Lotem, and H. Munoz-Avila. SHOP: simple hierarchical ordered planner. In *Proc. IJCAI*, 1999.
- [19] B. Nebel. On the compilability and expressive power of propositional planning formalisms. *JAIR*, 12:271–315, 2000.
- [20] M. Y. Vardi. The complexity of relational query languages. In *Proc. STOC*, 1982.

PROBABILISTIC INFERENCE

Optimal Time–Space Tradeoff in Probabilistic Inference

David Allen and Adnan Darwiche

University of California
Los Angeles, CA 90095
{dlallen,darwiche}@cs.ucla.edu

Abstract

Recursive Conditioning, RC, is an any–space algorithm for exact inference in Bayesian networks, which can trade space for time in increments of the size of a floating point number. This smooth tradeoff is possible by varying the algorithm’s cache size. When RC is run with a constrained cache size, an important problem arises: Which specific results should be cached in order to minimize the running time of the algorithm? RC is driven by a structure known as a dtree, and many such dtrees exist for a given Bayesian network. In this paper, we examine the problem of searching for an optimal caching scheme for a given dtree, and present some optimal time–space tradeoff curves for given dtrees of several published Bayesian networks. We also compare these curves to the memory requirements of state–of–the–art algorithms based on jointrees. Our results show that the memory requirements of these networks can be significantly reduced with only a minimal cost in time, allowing for exact inference in situations previously impractical. They also show that probabilistic reasoning systems can be efficiently designed to run under varying amounts of memory.

1 Introduction

Recursive Conditioning, RC, was recently proposed as an any–space algorithm for exact inference in Bayesian networks [Darwiche, 2001]. The algorithm works by using conditioning to decompose a network into smaller subnetworks that are then solved independently and recursively using RC. It turns out that many of the subnetworks generated by this decomposition process need to be solved multiple times redundantly, allowing the results to be stored in a cache after the first computation and then subsequently fetched during further computations. This gives the algorithm its any–space behavior since any number of results may be cached. This also leads to an important question, which is the subject of this paper: “Given a limited amount of memory, which results should be cached in order to minimize the running time of the recursive conditioning algorithm?”

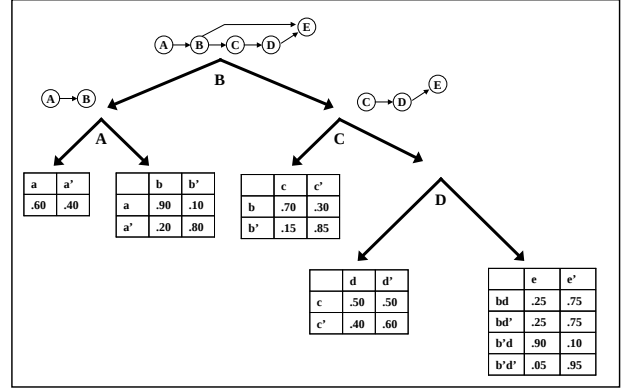


Figure 1: An example dtree.

We approach this problem by formulating it as a systematic search problem. We then use the developed method to construct time–space tradeoff curves for some real–world Bayesian networks, and put these curves in perspective by comparing them to the memory requirements of state–of–the–art methods based on jointrees [Jensen *et al.*, 1990; Shafer and Shenoy, 1990]. The curves produced illustrate that a significant amount of memory can be reduced with only a minimal cost in time. In fact, for much of their domains, the time–space curves we produce appear close to linear, with exponential behavior appearing only near the extreme case of no caching. This dramatic space reduction, without a significant time penalty, allows one to practically reason with Bayesian networks that would otherwise be impractical to handle.

This paper is structured as follows. We start in Section 2 by providing some background on recursive conditioning and the cache allocation problem. We then formulate this problem in Section 3 as a systematic search problem. Time–space tradeoff curves for several published Bayesian networks are then presented in Section 4. Finally, in Section 5, we provide some concluding remarks and discuss some future work.

2 Any–Space Inference

The RC algorithm for exact inference in Bayesian networks works by using conditioning and case analysis to decompose a network into smaller subnetworks that are solved independently and recursively. The algorithm is driven by a structure

known as a decomposition tree (dtree), which controls the decomposition process at each level of the recursion. We will first review the dtree structure and then discuss RC.

2.1 Dtrees

Definition 1 [Darwiche, 2001] A *dtree* for a Bayesian network is a full binary tree, the leaves of which correspond to the network conditional probability tables (CPTs). If a leaf node t corresponds to a CPT ϕ , then $\text{vars}(t)$ is defined as the variables appearing in CPT ϕ .

Figure 1 depicts a simple dtree. The root node t of the dtree represents the entire network. To decompose this network, the dtree instructs us to condition on variable B , called the cutset of root node t . Conditioning on a set of variables leads to removing edges outgoing from these variables, which for a cutset is guaranteed to disconnect the network into two sub-networks, one corresponding to the left child of node t and another corresponding to the right child of node t ; see Figure 1. This decomposition process continues until a boundary condition is reached, which is a subnetwork that has a single variable.

We will now present some notation needed to define additional concepts with regard to a dtree. The notation t_l and t_r will be used for the left child and right child of node t , and the function vars will be extended to internal nodes t : $\text{vars}(t) \stackrel{\text{def}}{=} \text{vars}(t_l) \cup \text{vars}(t_r)$. Each node in a dtree has three more sets of variables associated with it. The first two of these sets are used by the RC algorithm, while the third set is used to analyze the complexity of the algorithm.

Definition 2 The *cutset* of internal node t in a dtree is: $\text{cutset}(t) \stackrel{\text{def}}{=} \text{vars}(t_l) \cap \text{vars}(t_r) - \text{acutset}(t)$, where $\text{acutset}(t)$ is the union of cutsets associated with ancestors of node t in the dtree. The *context* of node t in a dtree is: $\text{context}(t) \stackrel{\text{def}}{=} \text{vars}(t) \cap \text{acutset}(t)$. The *cluster* of node t in a dtree is: $\text{cutset}(t) \cup \text{context}(t)$ if t is a non-leaf, and as $\text{vars}(t)$ if t is a leaf.

The *width* of a dtree is the size of its maximal cluster -1 .

The cutset of a dtree node t is used to decompose the network associated with node t into the smaller networks associated with the children of t . That is, by conditioning on variables in $\text{cutset}(t)$, one is guaranteed to disconnect the network associated with node t . The context of dtree node t is used to cache results: Any two computations on the network associated with node t will yield the same result if these computations occur under the same instantiation of variables in $\text{context}(t)$. Hence, a cache is associated with each dtree node t , which stores the results of such computations (probabilities) indexed by instantiations of $\text{context}(t)$. This means that the size of a cache associated with dtree node t can grow as large as the number of instantiations of $\text{context}(t)$.

For a given Bayesian network, many different dtrees exist and the quality of the dtree significantly affects the resource requirements of RC. The width is one important measure of this, as RC's time complexity is exponential in this value. The construction of dtrees is beyond the scope of this paper, but in [Darwiche, 2001; Darwiche and Hopkins, 2001] it was shown how to create them from elimination orders, jointrees,

Algorithm 1 $\text{RC}(t)$: Returns the probability of evidence $\mathbf{e}$ recorded on the dtree rooted at t .

```

1: if  $t$  is a leaf node then
2:   return LOOKUP( $t$ )
3: else
4:    $\mathbf{y} \leftarrow$  recorded instantiation of  $\text{context}(t)$ 
5:   if  $\text{cache?}(t)$  and  $\text{cache}_t[\mathbf{y}] \neq \text{nil}$  then
6:     return  $\text{cache}_t[\mathbf{y}]$ 
7:   else
8:      $p \leftarrow 0$ 
9:     for instantiations  $\mathbf{c}$  of uninstantiated vars in  $\text{cutset}(t)$  do
10:      record instantiation  $\mathbf{c}$ 
11:       $p \leftarrow p + \text{RC}(t_l)\text{RC}(t_r)$ 
12:      un-record instantiation  $\mathbf{c}$ 
13:   when  $\text{cache?}(t)$ ,  $\text{cache}_t[\mathbf{y}] \leftarrow p$ 
14:   return  $p$ 

```

Algorithm 2 LOOKUP(t).

```

 $\phi \leftarrow$  CPT of variable  $X$  associated with leaf  $t$ 
if  $X$  is instantiated then
   $x \leftarrow$  recorded instantiation of  $X$ 
   $\mathbf{u} \leftarrow$  recorded instantiation of  $X$ 's parents
  return  $\phi(x|\mathbf{u})$  //  $\phi(x|\mathbf{u}) = \Pr(x|\mathbf{u})$ 
else
  return 1

```

or directly by using the hMeTiS [Karypis and Kumar, 1998] hypergraph partitioning program.

2.2 Recursive Conditioning

Given a Bayesian network and a corresponding dtree with root t , the RC algorithm given in Algorithms 1 and 2 can be used to compute the probability of evidence $\mathbf{e}$ by first “recording” the instantiation $\mathbf{e}$ and then calling $\text{RC}(t)$, which returns the probability of $\mathbf{e}$.

Our main concern here is with Line 5 and Line 13 of the algorithm. On Line 5, the algorithm checks whether it has performed and cached this computation with respect to the subnetwork associated with node t . A computation is characterized by the instantiation of t 's context, which also serves as an index into the cache attached to node t . If the computation has been performed and cached before, its result is simply fetched. Otherwise, the computation is performed and its result is possibly cached on Line 13.

When every computation is cached, RC uses $O(n \exp(w))$ space and $O(n \exp(w))$ time, where n is the number of nodes in the network and w is the width of the dtree. This corresponds to the complexity of jointree algorithm, assuming that the dtree is generated from a jointree [Darwiche, 2001]. When no computations are cached, the memory requirement of RC is reduced to $O(n)$, in which case the time requirement increases to $O(n \exp(w \log n))$. Any amount of memory between these two extremes can also be used in increments of the size of a floating point number, a cache value.

Suppose now that the available memory is limited and we can only cache a subset of the computations performed by RC. The specific subset that we cache can have a dramatic effect on the algorithm's running time. A key question is then to choose that subset which minimizes the running time, which

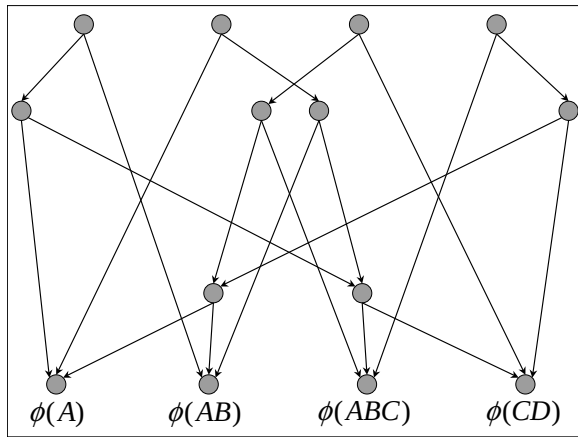


Figure 2: An example dgraph.

is the main objective of this paper. We refer to this as the *secondary optimization problem*, with the first optimization problem being that of constructing an optimal dtree.

Most of our results in this paper are based on a version of RC which not only computes the probability of evidence e , but also posterior marginals over families and, hence, posterior marginals over individual variables. This version of RC uses a *decomposition graph* (dgraph), which is basically a set of dtrees that share structure.

An example dgraph for a network with four variables can be seen in Figure 2. It should be noted that each of the four root nodes corresponds to a valid dtree, so this dgraph actually contains four dtrees which share a significant portion of their structure. Creation of dgraphs is discussed in [Darwiche, 2002].

The code in Algorithms 1 and 2 is also used in this version of RC, where $RC(t)$ is called once on each root t of the dgraph (the posterior marginal of each family is computed as a side effect of each of these calls). This version of RC uses more memory as it maintains more caches. But it is more meaningful when it comes to comparing our time-space tradeoff curves with the memory requirements of jointree algorithms, as this version of RC is equally powerful to these algorithms.

3 The Cache Allocation Problem

The total number of computations that a dgraph (or dtree) node t needs to cache equals the number of instantiations of $context(t)$. Given a memory constraint, however, one may not be able to cache all these computations, and we need a way to specify which results in particular to cache. A *cache factor* cf for a dgraph is a function which maps each internal node t in the dgraph into a number $cf(t)$ between 0 and 1. Hence, if $cf(t) = .75$, then node t can only cache 75% of these total computations. A *discrete* cache factor is one which maps every internal dgraph node into either 1 or 0: all of the node's computations are cached, or none are cached. The RC code in Algorithms 1 and 2 assumes a discrete cache factor, which is captured by the flag $cache?(t)$, indicating whether caching will take place at dgraph node t .

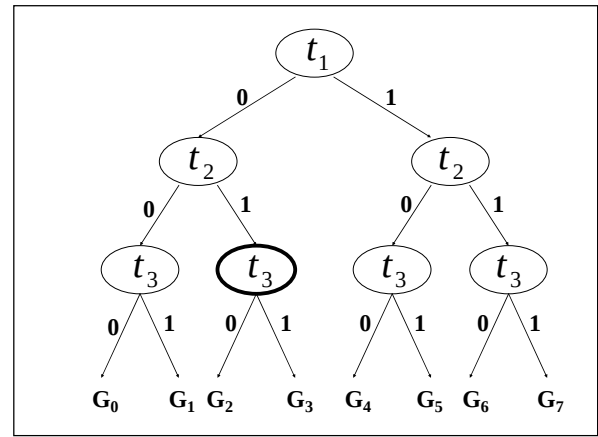


Figure 3: Search tree for a dgraph with 3 internal nodes.

One can count the number of recursive calls made by RC (and, hence, compute its running time) given any discrete cache factor. Specifically, if t^p denotes a parent of node t in a dgraph, and $S^\#$ denotes the number of instantiations of variables S , the number of recursive calls made to node t is [Darwiche, 2001; 2002]:

$$calls(t) = \sum_{t^p} cutset(t^p)^\# [cf(t^p)context(t^p)^\# + (1 - cf(t^p))calls(t^p)]. \quad (1)$$

If the cache factor is not discrete, the above formula gives the average number of recursive calls, since the actual number of calls will depend on the specific computations cached. This equation is significant as it can be used to predict the expected time requirement of RC under a given caching scheme.

We focus in this paper on searching for an optimal discrete cache factor, given a limited amount of memory, where optimality is with respect to minimizing the number of recursive calls. To this end, we will first define a search problem for finding an optimal discrete cache factor and then develop a depth-first branch-and-bound search algorithm. We will also use the developed algorithm to construct the time-space tradeoff curves for some published Bayesian networks from various domains, and compare these curves to the memory demands and running times of jointree algorithms.

3.1 Cache Allocation as a Search Problem

The cache allocation problem can be phrased as a search problem in which *states* in the search space correspond to partial cache factors that do not violate the given memory constraint, and where an *operator* extends a partial cache factor by making a caching decision on one more dgraph node. The *initial state* in this problem is the empty cache factor, in which no caching decisions have been made for any nodes in the dgraph. The *goal states* correspond to complete cache factors, where a caching decision has been made for every dgraph node, without violating the given memory constraint. Suppose for example that we have a dgraph with three internal nodes t_1, t_2, t_3 . This will then lead to the search tree in Figure 3. In this figure, each node n in the search tree represents a partial cache factor cf . For example, the node in bold

corresponds to the partial cache factor $cf(t_1) = 0$, $cf(t_2) = 1$ and $cf(t_3) = ?$. Moreover, if node n is labeled with a dgraph node t_i , then the children of n represent two possible extensions of the cache factor cf : one in which dgraph node t_i will cache all computations (1-child), and another in which dgraph node t_i will cache no computations (0-child).

According to the search tree in Figure 3, one always makes a decision on dgraph node t_1 , followed by a decision on dgraph node t_2 , and then node t_3 . A fixed ordering of dgraph nodes is not necessary, however, as long as the following condition is met: A decision should be made on a dgraph node t_i only after decisions have been made on all its ancestors in the dgraph. We will explain the reason for this constraint later on when we discuss cost functions.

In the search tree depicted in Figure 3, the leftmost leaf represents no caching, while the rightmost leaf represents full caching. The search trees for this problem have a maximum depth of d , where d is the number of internal nodes in the dgraph. Given this property, depth-first branch-and-bound search is a good choice given its optimality and linear space complexity [Papadimitriou and Steiglitz, 1998]. It is also an anytime algorithm, meaning that it can always return its best result so far if interrupted, and if run to completion will return the optimal solution. Hence, we will focus on developing a depth-first branch-and-bound search algorithm.

3.2 Cost Functions

The depth-first branch-and-bound (DFBnB) algorithm requires a cost function f which assigns a cost $f(n)$ to every node n in the search tree. The function $f(n)$ estimates the cost of an optimal solution that passes through n . The key here is that $f(n)$ must not overestimate that cost; otherwise, one loses the optimality guarantee offered by the search algorithm. We will now develop such a cost function $f(n)$ based on the following observations. Since each node n represents a partial cache factor cf , function $f(n)$ must estimate the number of recursive calls made to RC based on an optimal completion of cache factor cf . Consider now the completion cf' of cf in which we decide to cache at each dgraph node that cf did not make a decision on. This cache factor cf' is the best completion of cf from the viewpoint of running time, but it may violate the constraint given on total memory. Yet, we will use it to compute $f(n)$ as it guarantees that $f(n)$ will never overestimate the cost of an optimal completion of cf .

One important observation in this regard is that once the caching decision is made on the ancestors of dgraph node t , we can compute exactly the number of recursive calls that will be made to dgraph node t (see Equation 1). Therefore, when extending a partial cache factor, we will always insist on making a decision regarding a dgraph node t for which decisions have been made on all its ancestors. This improves the quality of the estimate $f(n)$ as n gets deeper in the tree. It also allows us to incrementally compute this estimate based on the estimate of n 's parent in the search tree.

3.3 Pruning

As depicted by the search tree in Figure 3, there is potentially an exponential number of goal nodes in the search tree and the combinatorial explosion of exhaustive search can become

unmanageable very quickly. Hence the search algorithm must eliminate portions of the search space while still being able to guarantee an optimal result. One of the key methods of doing this is by pruning parts of the search tree which are known to contain non-optimal results. The DFBnB algorithm does this by pruning search tree nodes when the cost function $f(n)$ is larger than or equal to the current best solution. Hence, more accurate cost functions will allow more pruning. Another major source of pruning is the given constraint on total memory. This is accomplished by pruning a search tree node and all its descendants once it attempts to assign more memory to caches than is permitted by the memory constraint.

3.4 Search Decisions

Now that we have chosen a cost function, we are still left with two important choices in our search algorithm: (1) which child of a search tree node to expanded first, and (2) in what order to visit dgraph nodes during search. Expanding the 1-child first is a greedy approach, as it attempts to fully cache at a dgraph node whenever possible. Results on many different networks have shown that in many cases, expanding the 1-child before the 0-child appears to be equal to or better than the opposite, and it is this choice that we adopt in our experiments. The specific order in which we visit dgraph nodes in the search tree turns out to have an even more dramatic effect on the efficiency of search. Even though we make caching decisions on parent dgraph nodes before their children, there is still a lot of flexibility. Our experimentation on many networks has shown that choosing the dgraph node t with the largest context(t)[#] is orders of magnitude more efficient than some other basic ordering heuristics [Allen and Darwiche, 2002]. This choice corresponds to choosing the dgraph node with the largest cache, and it is the one we use in our search algorithm.

4 Time-Space Tradeoff

The main goal of this section is to present time-space tradeoff curves for a number of benchmark Bayesian networks, some of which are obtained from [Bayesian Network Repository, URL] and others are included in the distributions of [Hugin Expert, URL; GeNIe, URL]. The main points to observe with respect to each curve is the slope of the curve, which provides information on the time penalty one pays when reducing space in probabilistic inference. The second main point is to compare the produced curves with the time and space requirement of jointree methods, as the version of RC we are using provides the same functionality as these algorithms (that is, probability of evidence and posterior marginals over families and variables). This baseline comparison is important as it places our results in the context of state-of-the-art inference systems.

Time-space tradeoff curves. Figures 4, 5, and 6 depict optimal discrete time-space tradeoff curves for three networks. These curves were generated as follows. A jointree was first generated for the network using Hugin.¹ The

¹We used Hugin's default setting: the minimum fill-in weight heuristic in conjunction with prime component analysis.

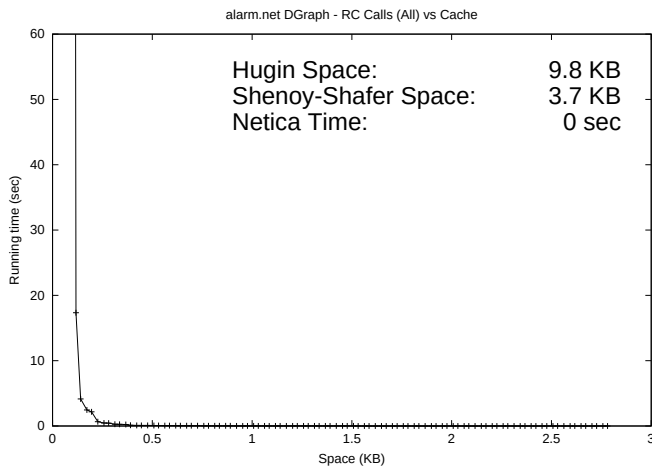


Figure 4: Time-space tradeoff on Alarm.

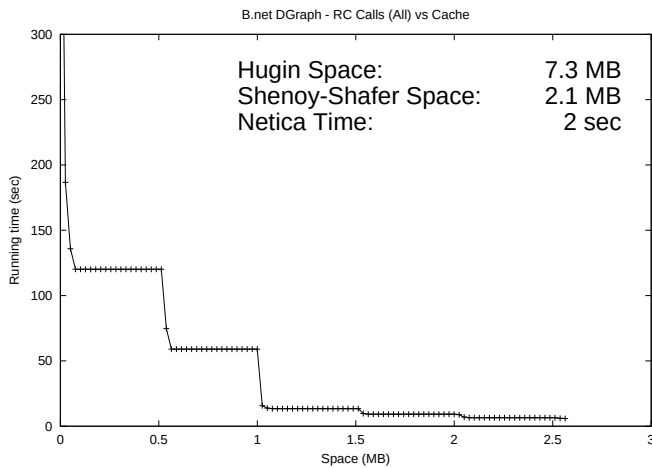


Figure 5: Time-space tradeoff on B.

jointree was then converted into a dtree as described in [Darwiche, 2001]. The dtree was finally converted into a dgraph as described in the full paper. Two sets of results were then generated:

- We computed the space requirements for jointree algorithms, using both the Hugin [Jensen *et al.*, 1990] and Shenoy-Shafer [Shafer and Shenoy, 1990] architectures. For the first architecture, we assumed one table for each clique and one table for each separator. For the second, we assumed two tables for each separator (no tables for cliques). We also performed propagation on the jointree using Netica [Norsys Software Corp., URL] and recorded the running time.
- We then ran our search algorithm to find an optimal cache factor under different memory constraints, where we generated 100 data points for each curve. For each caching factor that we identified, we computed the number of recursive calls that will be made by RC under that factor and converted the calls to seconds.²

²Our Java implementation of RC on a Sun Ultra 10, 440 MHz

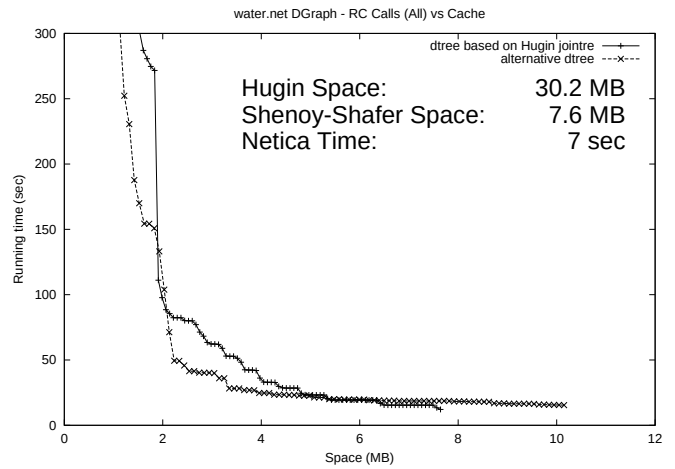


Figure 6: Time-space tradeoff on Water.

A number of observations are in order here. First, these time-space tradeoff curves show that the amount of memory used by RC under full caching is very close to that required by the Shenoy-Shafer architecture (the Hugin architecture takes much more space). Second, the curves show that a significant amount of memory can sometimes be reduced from full caching with only a limited increase in the time required; in fact, the exponential growth appears to be occurring only near the lower extreme of no caching. The space requirement for Water (Figure 6), for example, can be reduced to 30% while only increasing the running time by a factor of 6. Moreover, the space requirements for B (Figure 5) can be reduced to about 2.5% while increasing the running time by a factor of 20. Finally, we note that each optimal search for the Water and B networks took less than two seconds, and each optimal search for Alarm took less than two minutes. We stress though that such searches need to be done only once for a network, and their results can then be used for many further queries.

Non-optimal tradeoffs. On some networks, the search space is too large to solve optimally using our search algorithm, but the anytime nature of the algorithm allows us to interrupt the search at any point and ask for the best result obtained thus far. Figures 7 and 8 were generated by allowing the search to run for an hour. Even though these curves are not optimal, they are useful practically. For example, according to these curves, the memory requirement of Barley can be reduced from about 22 MB to about 6.5 MB while only increasing the running time from about 1 to 4 minutes. Moreover, the space requirement of Munin1 can be reduced from about 370 MB to 150 MB, while increasing the running time from about 22 minutes to about 6 hours. Encouraged by such results, we are planning to investigate other (non-optimal) search methods, such as local search.

Dtrees vs dgraphs. Running RC on a dtree takes less space than running it on a dgraph, but produces much less

computer with 256 MB of RAM, makes an average number of three million recursive calls per second.

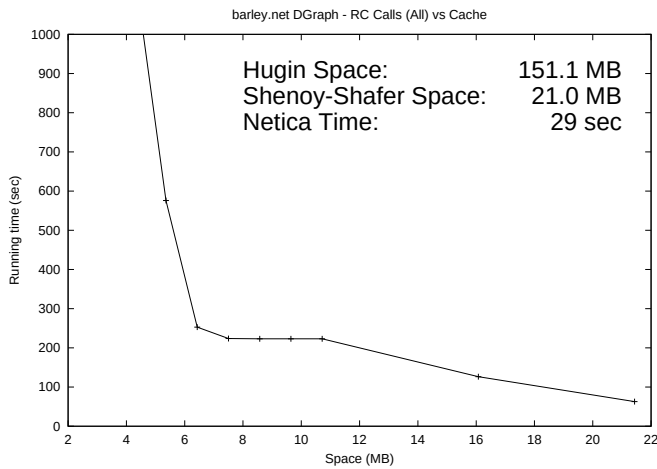


Figure 7: Time-space tradeoff on Barley.

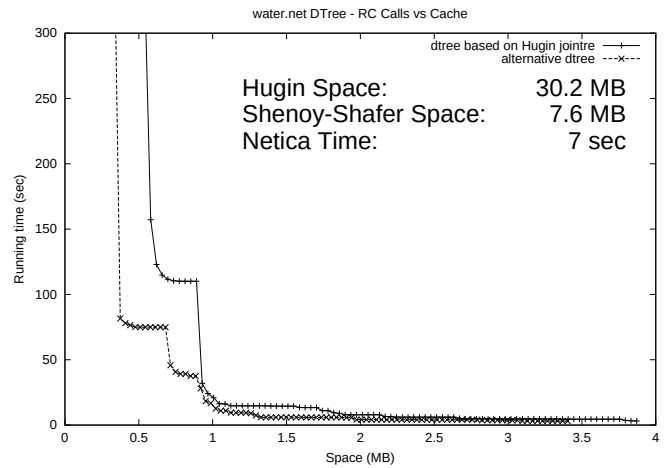


Figure 9: Time-space tradeoff on Water for computing probability of evidence.

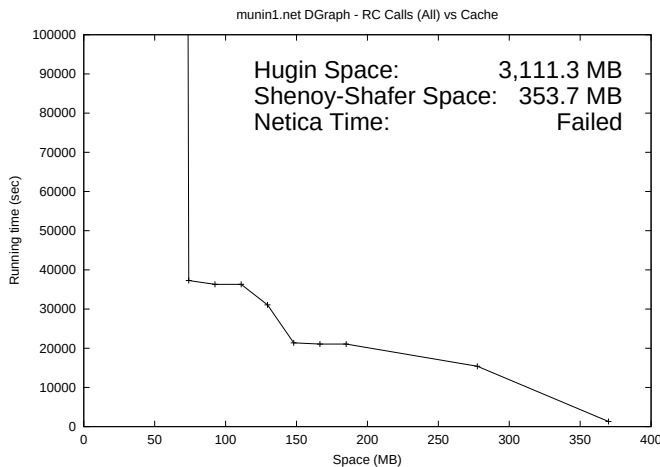


Figure 8: Time-space tradeoff on Munin1.

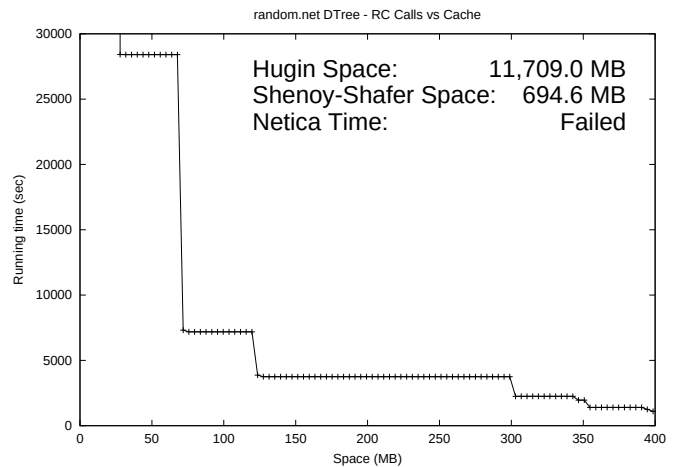


Figure 10: Time-space tradeoff on Random for computing probability of evidence.

information (probability of evidence instead of posterior marginals). To illustrate this difference concretely, we present in Figure 9 two tradeoff curves for the Water network, assuming a dtree version of RC, which require much less memory compared to the curves in Figure 6. Suppose now that we only have 1 MB of memory, instead of the 7.6 MB or 30.2 MB required by jointtree algorithms, and we want to compute the posterior marginals for all variables. According to Figure 6, we can do this in 669 seconds using the dgraph version of RC. The dtree version takes 16.7 seconds to compute the probability of evidence under this amount of memory, and we would have to run it 85 times to produce all posterior marginals for the Water network (given variable cardinalities in Water).

Effect of dtree/dgraph on tradeoff. Our notion of optimality for tradeoff is based on a given dtree/dgraph; hence, generating different dtrees/dgraphs could possibly lead to better time-space tradeoff curves. To illustrate this point, we generated tradeoff curves for the Water network based on

multiple dtrees/graphs, as shown in Figures 6 and 9. One observation that we came across is that dtrees/graphs that are based on jointtrees tend to require less memory under full caching, but are not necessarily best for tradeoff towards the no caching region; see Figures 6 for an example. Yet, we used such dtrees/graphs in this paper in an effort to provide a clear baseline for comparison with jointtree methods. If we relax this constraint, however, we can obtain better tradeoff curves than is generally reported here, as illustrated by Figures 6 and 9. The specific way in which properties of a dtree/dgraph influence the quality of corresponding time-space tradeoff curves is not very well understood, however, and we hope to shed more light on this in future work.

Size of search space. It should be noted that the difficulty of obtaining an optimal time-space tradeoff curve on some networks is not due to a large space requirement, but is due mostly to the number of nodes in the Bayesian network as that is what decides the size of search space. To further illustrate

this point, we generated a network randomly with 40 nodes (many of them non-binary), 86 edges, and a width of 14. This network requires extensive memory but has a relatively small number of variables. In fact, both Netica and Hugin were unable to compile the network requiring about 6 GB and 11 GB respectively. We were able, however, to produce an optimal time-space tradeoff curve for this network. The curve for the dtree version of RC is shown in Figure 10. According to this curve, we can compute the probability of any evidence on this network in about 2 hours using only about 75 MB.

Related work. We close this section by a note on related work for time-space tradeoff in probabilistic reasoning, which takes a different approach [Dechter and Fattah, 2001]. In this work, large separators in a jointree are removed by combining their adjacent clusters, which has the effect of reducing the space requirements of the Shenoy-Shafer architecture (as we now have fewer separators), but also increasing its running time (as we now have larger clusters). The tradeoffs permitted by this approach, however, are coarser than those permitted by RC as discussed in [Darwiche, 2001]. Furthermore, the secondary optimization problem of which separators to remove in order to minimize running time is not addressed in [Dechter and Fattah, 2001] for the proposed approach, as we do in this paper for the RC approach.

5 Conclusions and Future Work

The main contribution of this paper is a formal framework, and a corresponding working system, for trading space for time when designing probabilistic reasoning systems based on Bayesian networks. The proposal is based on the algorithm of recursive conditioning, and is accompanied with a set of experimental results showing that a significant amount of memory can sometimes be reduced while only incurring a reasonable penalty in running time. The proposed framework is then beneficial for designing reasoning systems with limited memory, as in embedded systems, and for reasoning with challenging networks on which jointree algorithms can exhaust the system memory.

Recursive conditioning and the described time-space tradeoff system have been implemented in JAVA in the SAMIAM tool, which is available publicly [UCLA Automated Reasoning Group, URL].

Acknowledgments

This work has been partially supported by NSF grant IIS-9988543 and MURI grant N00014-00-1-0617.

References

- [Allen and Darwiche, 2002] David Allen and Adnan Darwiche. Optimal time-space tradeoff in probabilistic inference. In *Proceedings of the First European Workshop on Probabilistic Graphical Models*, 1–8, 2002.
- [Bayesian Network Repository, URL] Bayesian Network Repository. <http://www.cs.huji.ac.il/labs/compbio/Repository/>, URL.
- [Darwiche and Hopkins, 2001] Adnan Darwiche and Mark Hopkins. Using recursive decomposition to construct elimination orders, jointrees, and dtrees. In *ECSQARU'01*, 180–191, 2001.
- [Darwiche, 2001] Adnan Darwiche. Recursive conditioning. *Artificial Intelligence*, 126(1–2):5–41, 2001.
- [Darwiche, 2002] Adnan Darwiche. Decomposition graphs. Technical Report D-134, Computer Science Department, UCLA, 2002.
- [Dechter and Fattah, 2001] Rina Dechter and Yousri El Fattah. Topological parameters for time-space tradeoff. *Artificial Intelligence*, 125(1–2):93–118, 2001.
- [GeNIe, URL] GeNIe. <http://www2.sis.pitt.edu/~genie/>, URL.
- [Hugin Expert, URL] Hugin Expert. <http://www.hugin.com/>, URL.
- [Jensen *et al.*, 1990] Finn V. Jensen, Steffen L. Lauritzen, and Kristian G. Olesen. Bayesian updating in causal probabilistic networks by local computations. *Computational Statistics Quarterly*, 4:269–282, 1990.
- [Karypis and Kumar, 1998] George Karypis and Vipin Kumar. Hmetis: A hypergraph partitioning package. <http://www.cs.umn.edu/karypis>, 1998.
- [Norsys Software Corp., URL] Norsys Software Corp. <http://www.norsys.com/>, URL.
- [Papadimitriou and Steiglitz, 1998] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization*. Dover Publications, Inc., 1998.
- [Shafer and Shenoy, 1990] Glenn R. Shafer and Prakash P. Shenoy. Probability propagation. *Annals of Mathematics and Artificial Intelligence*, 2:327–352, 1990.
- [UCLA Automated Reasoning Group, URL] UCLA Automated Reasoning Group. SamIam: Sensitivity Analysis, Modeling, Inference And More. <http://reasoning.cs.ucla.edu/samiam>, URL.

Variable Resolution Particle Filter

Vandi Verma, Sebastian Thrun and Reid Simmons

Carnegie Mellon University
5000 Forbes Ave. Pittsburgh PA 15217
{vandi, thrun, reids}@cs.cmu.edu

Abstract

Particle filters are used extensively for tracking the state of non-linear dynamic systems. This paper presents a new particle filter that maintains samples in the state space at dynamically varying resolution for computational efficiency. Resolution within statespace varies by region, depending on the belief that the true state lies within each region. Where belief is strong, resolution is fine. Where belief is low, resolution is coarse, abstracting multiple similar states together. The resolution of the statespace is dynamically updated as the belief changes. The proposed algorithm makes an explicit bias-variance tradeoff to select between maintaining samples in a biased generalization of a region of state space versus in a high variance specialization at fine resolution. Samples are maintained at a coarser resolution when the bias introduced by the generalization to a coarse resolution is outweighed by the gain in terms of reduction in variance, and at a finer resolution when it is not. Maintaining samples in abstraction prevents potential hypotheses from being eliminated prematurely for lack of a sufficient number of particles. Empirical results show that our variable resolution particle filter requires significantly lower computation for performance comparable to a classical particle filter.

1 Introduction

A number of problems in AI and robotics require estimation of the state of a system as it changes over time from a sequence of measurements of the system that provide noisy (partial) information about the state. Particle filters have been extensively used for Bayesian state estimation in nonlinear systems with noisy measurements [Isard and Blake, 1998; Fox *et al.*, 1999; Doucet *et al.*, 2001]. The Bayesian approach to dynamic state estimation computes a posterior probability distribution over the states based on the sequence of measurements. Probabilistic models of the change in state over time (state transition model) and relationship between the measurements and the state capture the noise inherent in these domains. Computing the full posterior in real time is often intractable. Particle filters approximate the posterior distribution over the states with a set of particles drawn from the posterior. The particle approximation converges to the true

Bayesian posterior in the limit as the number of samples go to infinity. For real-time state estimation it is impractical to have an infinitely large number of particles. With a small number of particles, the variance of the particle based estimate can be high, particularly when there are a large number of possible state transitions.

This paper presents a new particle filter, the variable resolution particle filter (VRPF), for tracking large state spaces efficiently with low computation. The basic idea is to represent a set of states¹ rather than a single state with a single particle and thus increase the number of states (or hypotheses) that may be tracked with the same number of particles. This makes it possible to track a larger number of states for the same computation, but results in a loss of information that differentiates between the states that are lumped together and tracked in abstraction. We formalize this tradeoff in terms of bias and variance. Tracking in abstraction at a coarser resolution introduces a bias over tracking at a finer resolution, but it reduces the variance of the estimator given a limited number of samples. By dynamically varying the resolution in different parts of the state space to minimize the bias-variance error, a finite set of samples are used in a highly efficient manner. Initial experimental results support our analysis and show that the bias is more than compensated for by the reduction in variance from not eliminating potential hypotheses prematurely for lack of a sufficient number of particles.

2 Bayesian Filtering

We denote the multivariate state at time t as X_t and measurements or observations as O_t . As is commonly the case, we concentrate on the discrete time, first order Markov formulation of the dynamic state estimation problem. Hence, the state at time t is a sufficient statistic of the history of measurements, i.e. $p(X_t|X_{0:t-1}) = p(X_t|X_{t-1})$ and the observations depend only on the current state, i.e. $p(O_t|X_{0:t}) = p(O_t|X_t)$. In the Bayesian framework, the posterior distribution at time t , $p(X_{0:t}|O_{1:t})$, includes all the available information upto time t and provides the optimal solution to the state estimation problem. In this paper we are interested in estimating recursively in time a marginal of this distribution, the filtering distribution, $p(X_t|O_{1:t})$, as the measurements be-

¹or regions of a continuous state space

come available. Based on the Markovian assumption, the recursive filter is derived as follows:

$$\begin{aligned}
p(X_t|O_{1:t}) &= \eta_t p(O_t|X_t, O_{1:t-1}) p(X_t|O_{1:t-1}) \\
&= \eta_t p(O_t|X_t) p(X_t|O_{1:t-1}) \\
&= \eta_t p(O_t|X_t) \int \{ p(X_t|O_{1:t-1}, X_{t-1}) \\
&\quad p(X_{t-1}|O_{1:t-1}) \} dX_{t-1} \\
&= \eta_t p(O_t|X_t) \int p(X_t|X_{t-1}) p(X_{t-1}|O_{1:t-1}) dX_{t-1}
\end{aligned} \tag{1}$$

This process is known as Bayesian filtering, optimal filtering or stochastic filtering and may be characterized by three distributions: (1) a transition model $p(X_t|X_{t-1})$, (2) an observation model $p(O_t|X_t)$, and, (3) an initial prior distribution, $p(X_0)$. In a number of applications, the state space is too large to compute the posterior in a reasonable time. Particle filters are a popular method for computing tractable approximations to this posterior.

2.1 Classical Particle Filter

A Particle filter (PF) [Metropolis and Ulam, 1949; Gordon *et al.*, 1993; Kanazawa *et al.*, 1995] is a Monte Carlo approximation of the posterior in a Bayes filter. PFs are non-parametric and can represent arbitrary distributions. PFs approximate the posterior with a set of N fully instantiated state samples or particles, $P_t = \{x_t^{[1]} \dots x_t^{[N]}\}$ as follows:

$$p(X_t|O_{1:t}) \approx \frac{1}{N} \sum_{i=1}^N \delta(X_t - x_t^{[i]}) \tag{2}$$

where $\delta(\cdot)$ denotes the Dirac delta function. It can be shown that as $N \rightarrow \infty$ the approximation in (2) approaches the true posterior density [Doucet and Crisan, 2002]. In general it is difficult to draw samples from, $p(X_t|O_{1:t})$; instead, samples are drawn from a more tractable distribution, $q(\cdot)$, called the proposal or importance distribution. Each particle is assigned a weight, $w_t^{[i]}$ to account for the fact that the samples were drawn from a different distribution [Rubin, 1988; Rubinstein, 1981]. There are a large number of possible choices for the proposal distribution, the only condition being that its support must include that of the posterior. The common practice is to sample from the transition probability, $p(X_t|X_{t-1})$, in which case the importance weight is equal to the likelihood, $p(O_t|X_t)$. The experiments in this paper were performed using this proposal distribution. The PF algorithm is:

```

1      set  $P_t = P_t^{\text{aux}} = \emptyset$ 
2      for  $i = 1$  to  $N$  do
3          pick the  $i$ -th sample  $x_{t-1}^{[i]} \in P_{t-1}$ 
4          draw  $x_t^{[i]} \sim p(X_t|x_{t-1}^{[i]})$ 
5          set  $w_t^{[i]} = p(O_t|x_t^{[i]})$ 
6          add  $\langle x_t^{[i]}, w_t^{[i]} \rangle$  to  $P_t^{\text{aux}}$ 
```

```

7      end
8      for  $i = 1$  to  $N$  do
9          draw  $x_t^{[i]}$  from  $P_t^{\text{aux}}$ 
          with probability proportional to  $w_t^{[i]}$ 
10         add  $x_t^{[i]}$  to  $P_t$ 
11     end
```

3 Variable Resolution Particle Filter

A well known problem with particle filters is that a large number of particles are often needed to obtain a reasonable approximation of the posterior distribution. For real-time state estimation maintaining such a large number of particles is typically not practical. However, the variance of the particle based estimate can be high with a limited number of samples, particularly when the process is not very stochastic and parts of the state space transition to other parts with very low, or zero, probability. Consider the problem of diagnosing locomotion faults on a robot. The probability of a stalled motor is low and wheels on the same side generate similar observations. Motors on any of the wheels may stall at any time. A particle filter that produces an estimate with a high variance is likely to result in identifying some arbitrary wheel fault on the same side, rather than identifying the correct fault.

The variable resolution particle filter introduces the notion of an abstract particle, in which particles may represent individual states or sets of states. With this method a single abstract particle simultaneously tracks multiple states. A limited number of samples are therefore sufficient for representing large state spaces. A bias-variance tradeoff is made to dynamically refine and abstract states to change the resolution, thereby abstracting a set of states and generalizing the samples or specializing the samples in the state into the individual states that it represents. As a result reasonable posterior estimates can be obtained with a relatively small number of samples. In the example above, with the VRPF the wheel faults on the same side of the rover would be aggregated together into an abstract fault. Given a fault, the abstract state representing the side on which the fault occurs would have high likelihood. The samples in this state would be assigned a high importance weight. This would result in multiple copies of these samples on resampling proportional to weight. Once there are sufficient particles to populate all the refined states represented by the abstract state, the resolution of the state would be changed to the states representing the individual wheel faults. At this stage, the correct hypothesis is likely to be included in this particle based approximation at the level of the individual states and hence the correct fault is likely to be detected.

For the variable resolution particle filter we need: (1) A variable resolution state space model that defines the relationship between states at different resolutions, (2) an algorithm for state estimation given a fixed resolution of the state space, (3) a basis for evaluating resolutions of the state space model, and (4) an algorithm for dynamically altering the resolution of the state space.

3.1 Variable resolution state space model

We could use a directed acyclic graph (DAG) to represent the variable resolution state space model, which would consider every possible combination of the (abstract)states to aggregate or split. But this would make our state space exponentially large. We must therefore constrain the possible combinations of states that we consider. There are a number of ways to do this. For the experiments in this paper we use a multi-layered hierarchy where each physical (non-abstract) state only exists along a single branch. Sets of states with similar state transition and observation models are aggregated together at each level in the hierarchy. In addition to the physical state set $\{X_k\}$, the variable resolution model, M consists of a set of abstract states $\{S_j\}$ that represent sets of states and or other abstract states.

$$S_j = \left\{ \begin{array}{l} \{X_k\} \\ \cup_i S_i \end{array} \right\} \quad (3)$$

Figure 1(a) shows an arbitrary Markov model and figure 1(b) shows an arbitrary variable resolution model for 1(a). Figure 1(c) shows the model in 1(b) at a different resolution.

We denote the measurement at time t as z_t and the sequence of measurement $z_1 \dots z_t$ as z^t . From the dynamics, $p(x_t|x_{t-1})$, and measurement probabilities $p(z_t|x_t)$, we compute the stationary distribution (Markov chain invariant distribution) of the physical states $\pi(x_k)$ [Behrendts, 2000].

3.2 Belief state estimation at a fixed resolution

This section describes the algorithm for estimating a distribution over the state space, given a fixed resolution for each state, where different states may be at different fixed resolutions. For each particle in a physical state, a sample is drawn from the predictive model for that state $p(x_t|x_{t-1})$. It is then assigned a weight proportional to the likelihood of the measurement given the prediction, $p(z_t|x_t)$. For each particle in an abstract state, S_j , one of the physical states, x_t , that it represents in abstraction is selected proportional to the probability of the physical state under the stationary distribution, $\pi(x_t)$. The predictive and measurement models for this physical state are then used to obtain a weighted posterior sample. The particles are then resampled proportional to their weight. Based on the number of resulting particles in each physical state a Bayes estimate with a Dirichlet(1) prior is obtained as follows:

$$\hat{p}(x_t|z^t) = \frac{n(x_t) + \pi(x_t)}{|N_t| + 1}, \quad \sum_{x_t} \pi(x_t) = 1 \quad (4)$$

where, $n(x_t)$ represents the number of samples in the physical state x_t and $|N_t|$ represents the total number of particles in the particle filter. The distribution over an abstract state S_j at time t is estimated as:

$$\hat{p}(S_j|z^t) = \sum_{x_t \in S_j} \hat{p}(x_t|z^t) \quad (5)$$

3.3 Bias-variance tradeoff

The loss l , from a particle based approximation $\hat{p}(x_t|z^t)$, of the true distribution $p(x_t|z^t)$ is:

$$\begin{aligned} l &= \mathbb{E}[p(x_t|z^t) - \hat{p}(x_t|z^t)]^2 \\ &= \{p(x_t|z^t) - \mathbb{E}[\hat{p}(x_t|z^t)]\}^2 + \\ &\quad \{ \hat{p}(x_t|z^t)^2 - \mathbb{E}[\hat{p}(x_t|z^t)]^2 \} \\ &= b(\hat{p}(x_t|z^t))^2 + v(\hat{p}(x_t|z^t)) \end{aligned} \quad (6)$$

where, $b(\hat{p}(x_t|z^t))$ is the bias and $v(\hat{p}(x_t|z^t))$ is the variance.

The posterior belief state estimate from tracking states at the resolution of physical states introduces no bias. But the variance of this estimate can be high, specially with small sample sizes. An approximation of the sample variance at the resolution of the physical states may be computed as follows:

$$v(x_t) = \hat{p}(x_t|z^t) \frac{\hat{p}(x_t|z^t) [1 - \hat{p}(x_t|z^t)]}{n(x_t) + \pi(x_t)} \quad (7)$$

The loss of an abstract state S_j , is computed as the weighted sum of the loss of the physical states $x_t \in S_j$, as follows²:

$$l(S_j) = \sum_{x_t \in S_j} \hat{p}(x_t|z^t) l(x_t) \quad (8)$$

The generalization to abstract states biases the distribution over the physical states to the stationary distribution. In other words, the abstract state has no information about the relative posterior likelihood, given the data, of the states that it represents in abstraction. Instead it uses the stationary distribution to project its posterior into the physical layer. The projection of the posterior distribution $\hat{p}(S_j|z^t)$, of abstract state S_j , to the resolution of the physical layer $\tilde{p}(x_t|z^t)$, is computed as follows:

$$\tilde{p}(x_t|z^t) = \frac{\pi(x_t)}{\pi(S_j)} \hat{p}(S_j|z^t) \quad (9)$$

where, $\pi(S_j) = \sum_{x_t \in S_j} \pi(x_t)$.

As a consequence of the algorithm for computing the posterior distribution over abstract states described in section 3.2, an unbiased posterior over physical states x_t is available at no extra computation, as shown in equation (4). The bias $b(S_j)$, introduced by representing the set of physical states $x_t \in S_j$, in abstraction as S_j is approximated as follows:

$$b(S_j) = \sum_{x_t \in S_j} \hat{p}(x_t|z^t) [\hat{p}(x_t|z^t) - \tilde{p}(x_t|z^t)]^2 \quad (10)$$

It is the weighed sum of the squared difference between the unbiased posterior $\hat{p}(x_t|z^t)$, computed at the resolution of the physical states and the biased posterior $\tilde{p}(x_t|z^t)$, computed at the resolution of abstract state S_j .

An approximation of the variance of abstract state S_j is computed as a weighted sum of the projection to the physical states as follows:

$$v(S_j) = \sum_{x_t \in S_j} \hat{p}(x_t|z^t) \left[\frac{\pi(x_t)}{\pi(S_j)} \right]^2 \frac{\hat{p}(S_j|z^t) [1 - \hat{p}(S_j|z^t)]}{n(S_j) + \pi(S_j)} \quad (11)$$

²The relative importance/cost of the physical states may also be included in the weight

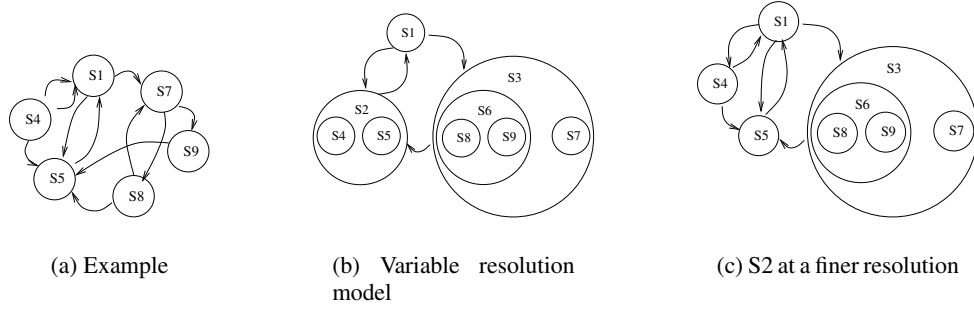


Figure 1: (a) Arbitrary Markov model (b) Arbitrary variable resolution model corresponding to the Markov model in (a). Circles that enclose other circles represent abstract states. S2 and S3 and S6 are abstract states. States S2, S4 and S5 form one abstraction hierarchy, and states S3, S6, S7, S8 and S9 form another abstraction hierarchy. The states at the highest level of abstraction are S1, S2 and S3. (c) The model in (b) with states S4 and S5 at a finer resolution.

The loss from tracking a set of states $x_t \in S_j$, at the resolution of the physical states is thus:

$$l_p = 0 + \sum_{x_t \in S_j} v(x_t) \quad (12)$$

The loss from tracking the same set of states in abstraction as S_j is:

$$l_a = b(S_j) + v(S_j) \quad (13)$$

There is a gain in terms of reduction in variance from generalizing and tracking in abstraction, but it results in an increase in bias. Here, a tradeoff between bias and variance refers to the process of accepting a certain increase in one term for a larger reduction in the other and hence in the total error.

3.4 Dynamically varying resolution

The variable resolution particle filter uses a bias-variance tradeoff to make a decision to vary the resolution of the state space. A decision to abstract to the coarser resolution of abstract state S_j , is made if the state space is currently at the resolution of states S_i , and the combination of bias and variance in abstract state S_j , is less than the combination of bias and variance of all its children S_i , as shown below:

$$b(S_j) + v(S_j) \leq \sum_{S_i \in \{\text{children}(S_j)\}} [b(S_i) + v(S_i)] \quad (14)$$

On the other hand if the state space is currently at the resolution of abstract state S_j , and the reverse of equation (14) is true, then a decision to refine to the finer resolution of states S_i is made. The resolution of a state is left unaltered if its bias-variance combination is less than its parent and its children. To avoid hysteresis, all abstraction decisions are considered before any refinement decisions.

Each time a new measurement is obtained the distribution of particles over the state space is updated. Since this alters the bias and variance tradeoff, the states explicitly represented at the current resolution of the state space are each evaluated for gain from abstraction or refinement. Any change in the current resolution of the state space is recursively evaluated for further change in the same direction.

4 Experimental results

The problem domain for our experiments involves diagnosing locomotion faults in a physics based simulation of a six wheel rover. Figure 2(a) shows a snapshot of the rover in the Darwin2K [Leger, 2000] simulator.

The experiment is formulated in terms of estimating discrete fault and operational modes of the robot from continuous control inputs and noisy sensor readings. The discrete state, x_t , represents the particular fault or operational mode. The continuous variables, z_t , provide noisy measurements of the change in rover position and orientation. The particle set P_t therefore consists of N particles, where each particle $x_t^{[i]}$ is a hypothesis about the current state of the system. In other words, there are a number of discrete fault and operational states that a particle may transition to based on the transition model. Each discrete fault state has a different observation and predictive model for the continuous dynamics. The probability of a state is determined by the density of samples in that state.

The Markov model representing the discrete state transitions consists of 7 states. As shown in figure 2(c) the normal driving (ND) state may transition back to the normal driving state or to any one of six fault states: right front (RF), right middle (RM), right rear (RR), left front (LF), left middle (LM) and left rear (LR) wheel stuck. Each of these faults cause a change in the rover dynamics, but the faults on each side (right and left), have similar dynamics.

Given that the three wheels on each side of the rover have similar dynamics, we constructed a hierarchy that clusters the fault states on each side together. Figure 2(d) shows this hierarchical model, where the abstract states right side fault (RS), and left side fault (LS) represent sets of states {RF, RM, RR} and {LF, LM, LR} respectively. The highest level of abstraction therefore consists of nodes {ND, RS, LS}. Figure 2(e) shows how the state space in figure 2(d) would be refined if the bias in the abstract state RS given the number of particles outweighs the reduction in variance over the specialized states RF, RM and RR at a finer resolution.

When particle filtering is performed with the variable resolution particle filter, the particles are initialized at the highest

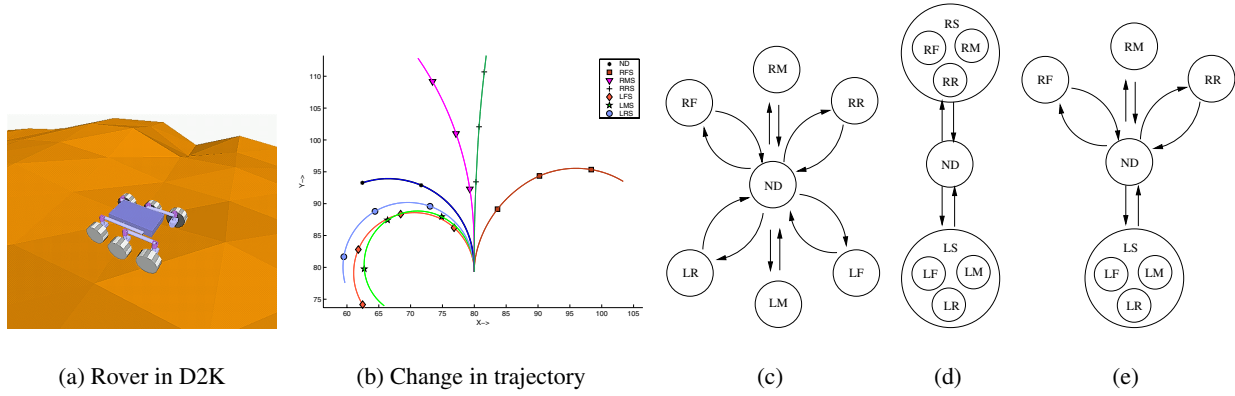


Figure 2: (a) Snapshot from the dynamic simulation of the six wheel rocker bogie rover in the simulator; (b) An example showing the normal trajectory (ND) and the change in the same trajectory with a fault at each wheel. (c) Original discrete state transition model. The discrete states are: Normal driving (ND), right and left, front, middle and rear wheel faulty (RF, RM, RR, LF, LM, LR) (d) Abstract discrete state transition model. The states, RF, RM and RR have been aggregated into the Right Side wheel faulty states and similarly LF, LM and LR into Left Side faulty states (RS and LS). (e) State space model where RS has been refined. All states have self transitions that have been excluded for clarity.

level in the abstraction hierarchy, i.e. in the abstract states ND, RS and LS. Say a RF fault occurs, this is likely to result in a high likelihood of samples in RS. These samples will multiply which may then result in the bias in RS exceeding the reduction in variance in RS over RF, RM and RR thus favoring tracking at the finer resolution. Additional observations should then assign a high likelihood to RF.

The model is based on the real-world and is not very stochastic. It does not allow transitions from most fault states to other fault states. For example, the RF fault does not transition to the RM fault. This does not exclude transitions to multiple fault states and if the model included multiple faults, it could still transition to a “RF and RM” fault, which is different from a RM fault. Hence, if there are no samples in the actual fault state, samples that end up in fault states with dynamics that are similar to the actual fault state may end up being identified as the fault state. The hierarchical approach tracks the state at an abstract level and does not commit to identifying any particular specialized fault state until there is sufficient evidence. Hence it is more likely to identify the correct fault state.

Figure 3(a) shows a comparison of the error from monitoring the state using a classical particle filter that tracks the full state space, and the VRPF that varies the resolution of the state space. The X axis shows the number of particles used, the Y axis shows the KL divergence from an approximation of the true posterior computed using a large number of samples. 1000 samples were used to compute an approximation to the true distribution. The KL divergence is computed over the entire length of the data sequence and is averaged over multiple runs over the same data set³. The data set included normal operation and each of the six faults. Figure 3(a) demonstrates that the performance of the VRPF is su-

perior to that of the classical filter for small sample sizes. In addition figure 3(b) shows the KL-divergence along the Y axis and wall clock time along the X axis. Both filters were coded in matlab and share as many functions as possible.

5 Discussion and Future Work

This paper presents a novel approach for state abstraction in particle filters that allows efficient tracking of large discrete, continuous or hybrid state spaces. The advantage of this approach is that it makes efficient use of the available computation by selecting the resolution of the state space based on an explicit bias-variance tradeoff. It performs no worse than a classical particle filter with an infinite number of particles⁴. But with limited particles we show that its performance is considerably superior. The VRPF does not prematurely eliminate potential hypotheses for lack of a sufficient number of particles. It maintains particles in abstraction until there are a sufficient number of particles to provide a low variance estimate at a lower resolution.

The VRPF generalizes from current samples to unsampled regions of the state space. Regions of the state space that had no samples at time t may acquire samples at time $t + 1$ on abstraction. This provides additional robustness to noise that may have resulted in eliminating a likely hypothesis. [Koller and Fratkina, 1998] addresses this by using the time t samples as input to a density estimation algorithm that learns a distribution over the states at time t . Samples at time $t + 1$ are then generated using this generalized distribution. [Ng *et al.*, 2002] uses a factored representation to allow a similar mixing of particles. The VRPF uses a bias-variance tradeoff to generalize more selectively and efficiently.

We have presented the variable resolution particle filter for a discrete state space. We plan to extend it to a continuous

³The results are an average over 50 to 5 runs with repetitions decreasing as the sample size was increased.

⁴There is a minor overhead in terms of evaluating the resolution of the state space

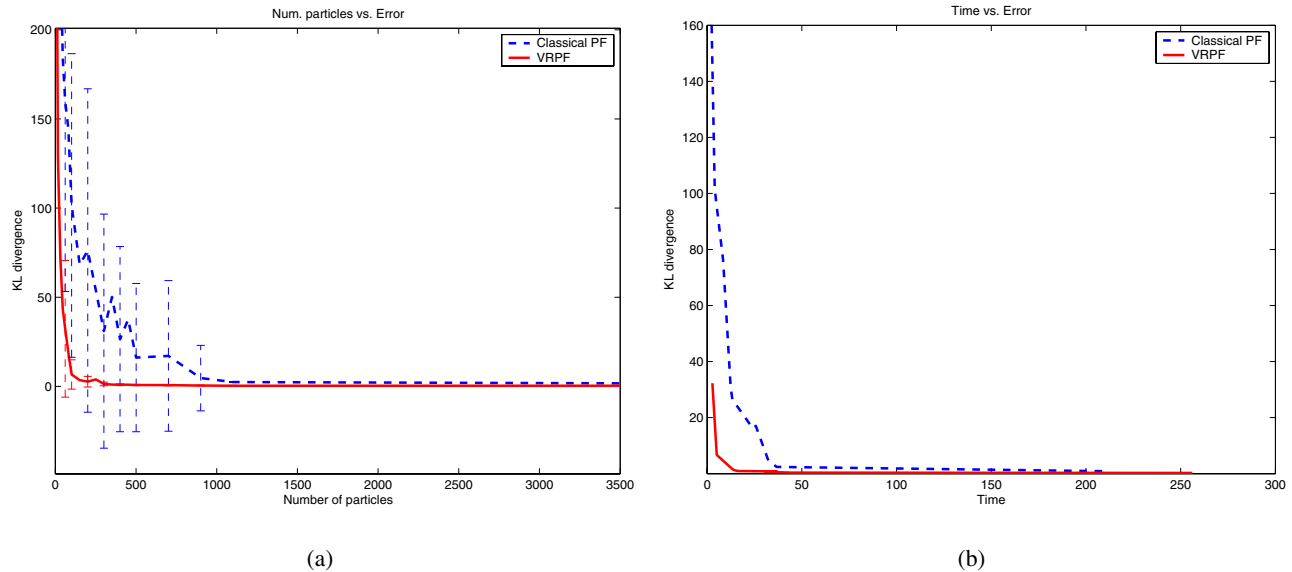


Figure 3: Comparison of the KL divergence from the true distribution for the classical particle filter and the VRPF, against (a) number of particles used, (b) wall clock time.

state space where regions of state space, rather than sets of states are maintained at different resolutions. A density tree may be used for efficiently learning the variable resolution state space model with abstract states composed of finitely many sub-regions.

The VRPF makes efficient use of available computation and can provide significantly improved posterior estimates given a small number of samples. However, it should not be expected to solve all problems associated with particle impoverishment. For example, it assumes that different regions of the state space are important at different times. We have generally found this to be the case, but the VRPF does not improve performance if it is not. Another situation is when the transition and observation models of all the states are highly dissimilar and there is no possibility for state aggregation.

6 Acknowledgments

We would like to thank Geoff Gordon, Tom Minka and the anonymous reviewers.

References

- [Behrends, 2000] E. Behrends. *Introduction to Markov Chains with Special Emphasis on Rapid Mixing*. Vieweg Verlag, 2000.
- [Doucet and Crisan, 2002] A. Doucet and D. Crisan. A survey of convergence results on particle filtering for practitioners. *IEEE Trans. Signal Processing*, 2002.
- [Doucet et al., 2001] A. Doucet, N. de Freitas, and N.J. Gordon, editors. *Sequential Monte Carlo Methods in Practice*. Springer-Verlag, 2001.
- [Fox et al., 1999] D. Fox, W. Burgard, and S. Thrun. Markov localization for mobile robots in dynamic environments. In *Journal of Artificial Intelligence*, volume 11, pages 391–427, 1999.
- [Gordon et al., 1993] N.J. Gordon, D.J. Salmond, and A.F.M. Smith. Novel approach to nonlinear/non gaussian bayesian state estimation. volume 140 of *IEE Proceedings-F*, pages 107–113, 1993.
- [Isard and Blake, 1998] M. Isard and A. Blake. Condensation: conditional density propagation for visual tracking. *International Journal of Computer Vision*, 29(1):5–28, 1998.
- [Kanazawa et al., 1995] K. Kanazawa, D. Koller, and S. Russell. Stochastic simulation algorithms for dynamic probabilistic networks. pages 346–351, 1995.
- [Koller and Fratkin, 1998] D. Koller and R. Fratkin. Using learning for approximation in stochastic processes. In *ICML*, 1998.
- [Leger, 2000] P.C. Leger. *Darwin2K: An Evolutionary Approach to Automated Design for Robotics*. Kluwer Academic Publishers, 2000.
- [Metropolis and Ulam, 1949] N. Metropolis and S. Ulam. The monte carlo method. volume 44 of *Journal of the American Statistical Association*, 1949.
- [Ng et al., 2002] B. Ng, L. Peshkin, and A. Pfeffer. Factored particles for scalable monitoring. In *UAI*, 2002.
- [Rubin, 1988] D. B. Rubin. *Using the SIR algorithm to simulate posterior distributions*. Oxford University Press, 1988.
- [Rubinstein, 1981] R. Y. Rubinstein. Simulation and Monte Carlo method. In *John Wiley and Sons*, 1981.

PROBABILISTIC INFERENCE

PROBABILISTIC INFERENCE: FIRST ORDER

First-order probabilistic inference

David Poole

Department of Computer Science

University of British Columbia

Vancouver, B.C., Canada V6T 1Z4

poole@cs.ubc.ca

<http://www.cs.ubc.ca/spider/poole/>

Abstract

There have been many proposals for first-order belief networks (i.e., where we quantify over individuals) but these typically only let us reason about the individuals that we know about. There are many instances where we have to quantify over all of the individuals in a population. When we do this the population size often matters and we need to reason about all of the members of the population (but not necessarily individually). This paper presents an algorithm to reason about multiple individuals, where we may know particular facts about some of them, but want to treat the others as a group. Combining unification with variable elimination lets us reason about classes of individuals without needing to ground out the theory.

1 Introduction

Belief networks or Bayesian networks [Pearl, 1988] are a popular representation for independencies amongst random variables. They are, however zeroth-order representations; to reason about multiple individuals, we have to make each property of each individual into a separate node.

There have been proposals to allow for ways to lift belief networks to a first order representation. The first tradition [Breese, 1992; Horsch and Poole, 1990; Wellman, Breese and Goldman, 1992] essentially allowed for parameterized belief networks, which could be grounded for each individual. The second tradition [Poole, 1993; Koller and Pfeffer, 1998; Pfeffer, Koller, Milch, and Takusagawa, 1999] allowed for richer first order probabilistic representations that have belief networks as special cases. In all of these the only individuals assumed to exist are those that we know about.

There are many cases where we want to reason about a set of individuals as a group. We'd like avoid explicit reasoning about each individual separately. There was a great advance in theorem proving in the 1960s with the invention of resolution and unification [Robinson, 1965]. The big advance was that doing resolution on clauses that contain free variables implements a potentially unbounded number of resolutions on the grounded representation. The goal of the current work is to allow for similar savings in probabilistic reasoning.

Probabilistic reasoning is more challenging than logical reasoning for a number of reasons:

- We have to use all of our information when making a probabilistic inference; new information can change old conclusions.
- We don't want to double count evidence. If we have some probabilistic information about all people and we use it for one particular individual, say Fred, then we can't reuse that information for Fred.
- There are cases where the size of the domain affects the probability. In the example developed below, determining the probability that a person is guilty of a particular crime depends on the population; the population size affects the number of other people who could be guilty.

The following example shows why we may need to reason about the individuals we don't know about as well as the individuals we do know about and shows why we need to consider population size.

Example 1 A person in a town was seen committing a crime. This person had the same (unusual) hair colour and car colour as Joe (both purple) and the person was very tall and we know Joe has big feet (and being tall is correlated with having big feet). What is the probability that Joe is guilty? We need to model the probabilities of the observations and their dependence, which would lead us to consider belief networks as a representation. The probability Joe is guilty also depends on the population. If the population of the town is very small then he is probably guilty (as it is unlikely there is anyone else fitting this description). If the town was a city containing a large number of people then he is probably innocent (as we would expect many other people to also fit this description).

This example points to a number of features that have been ignored in first-order probabilistic models. Not only do we need to take population sizes into account but we need to be able to reason about the individuals we know about as well as the individuals who we know exist, but we don't know anything particular about. We don't have to ground out our theory and reason about millions of people in a city separately, but we also cannot ignore the relevant information about the people we know something about.

2 Representation

We assume that the domain is represented as a parametrized belief network, where the nodes are parametrized with

domain-restricted types (or populations). Such an idea has been explored previously [Horsch and Poole, 1990; Kersting and De Raedt, 2000], and is similar to the plates of Buntine [1994]. We will not dwell on the semantics in this paper. We just assume that the program means the grounding: the substitution of constants for each individual in the domain of a parameter.

This paper is explicitly Bayesian. In terms of the first-order probability of Halpern [1990], the probabilities are all degrees of belief, not statistical assertions about proportions of a population. We are not quantifying over random individuals (as does Bacchus [1990]), but over all individuals (i.e., with standard universal quantification). All of the individuals about which we have the same information have the same probability. It is this property that we exploit computationally.

This paper is built on two different traditions, namely that of logic programming and theorem proving on one side and that of probabilistic reasoning on the other side. Unfortunately they use the same terms or notations (e.g., “variable”, “domain”, “=”) for very different things (or at least they are used in very different ways in this paper). In order to avoid confusion, we use neutral terms, but use the traditions of the different communities as appropriate.

A **population** is a set of **individuals**. A population corresponds to a domain in logic. The cardinality of the population is called the **population size**. For example, the population may be the set of people living in Vancouver, and the population size is 2 million people. The population size can be finite or infinite.

We allow for random variables to be parametrized, where parameters start with an upper case letter and constants start with a lower case letter. Parameters correspond to logical variables (e.g., as a variable in Prolog). All of the parameters are typed with a population. A parametrized random variable is of the form $f(t_1, \dots, t_k)$ where f is a functor (either a function symbol or a predicate symbol) and each t_i is a parameter or a constant. Each functor has a set of **values** called the **range** of the functor.

Examples of parametrized random variables are $hair_colour(X)$, $likes(X, Y)$, $likes(joe, X)$ or $town_conservativeness$ (the latter being a functor of no arguments), where $hair_colour$, $likes$ and $town_conservativeness$ are functors.

Given an assignment of a constant to each parameter, a parametrized random variable represents a random variable. For example, suppose hair colour has range $\{black, blond, red, grey, purple, orange, none, multicoloured\}$. Then $hair_colour(sam)$ is a random variable with domain $\{black, blond, red, grey, purple, orange, none, multicoloured\}$. Thus a parametrized random variable represents a set of random variables, one for each parameter assignment. Different parameter assignments for the variables in a parametrized random variable result in different random variables; for example, $hair_colour(fred)$ is a different random variable to $hair_colour(joe)$.

A **parametrized primitive proposition** is an expression of the form $t = v$, which means that parametrized random variable t has value v . A **parametrized proposition** is built from the parametrized primitive propositions using the normal

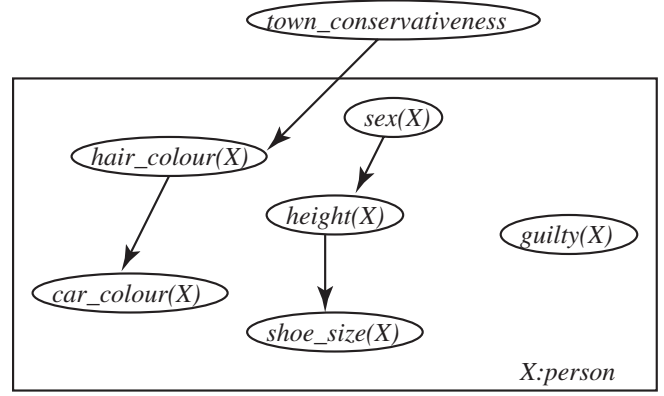


Figure 1: Robbing example parametrized belief network (Example 2) using plates and parametrized random variables.

logical connectives.

A **probabilistic assertion** is of the form:

$$\forall X_1 : d_1, \dots, \forall X_k : d_k; C \rightarrow P(\alpha|\beta) = p$$

where X_i are parameters, d_i are populations, C is a set of inequality constraints on $X_1, \dots, X_k$, α and β are parametrized propositions using only the parameters $X_1, \dots, X_k$, p specifies a probability distribution over α . We omit the corresponding syntactic constructs when $k = 0$, $C = \{\}$ or $\beta = true$.

A **parametrized belief network** consists of

- a DAG where the nodes are parametrized random variables,
- an association of a population to each parameter,
- an assignment of a range to each functor,
- a set of probability assertions for each node given its parents.

Example 2 To formalize Example 1, consider the parametrized belief network of Figure 1. Here we have shown it using the plates¹ [Buntine, 1994] as well as with parametrized random variables. We assume that the hair-colour of the different people are not independent; they depend on how conservative the town is. Associated with this network are conditional probabilistic assertions such as ²:

$$P(conservative) = 0.7$$

$$\forall X : person; P(guilty(X)) = 0.0000001$$

$$\forall X : person; P(very_tall(X)|male(X)) = 0.1$$

$$\forall X : person; P(hair_colour(X)=purple|$$

$$conservative) = 0.001$$

If we knew as background knowledge that *sam* was an exception and has purple hair with probability 0.5, we would replace the last probabilistic assertion with:

$$\forall X : person; X \neq sam \rightarrow$$

$$P(hair_colour(X)=purple|conservative) = 0.001$$

¹Here we use plates just as a visual help; the real meaning is as probabilistic assertions. The plates get difficult to interpret when there are complex interactions between the parameters, but we can always interpret these as probabilistic assertions.

²We use *conservative* as an abbreviation for $town_conservativeness = conservative$, $male(X)$ as an abbreviation for $sex(X) = male$ and $very_tall(X)$ for $height(X) = very_tall$.

$$P(\text{hair_colour}(\text{sam})=\text{purple}) = 0.5$$

(We will not use this in our continuing example).

A **grounding** of a parametrized belief network is a belief network that consists of all instances of the parametrized random variables where each parameter is replaced by an individual in the population (or a ground term denoting that individual).

Intuitively, a parametrized belief network represents a huge belief network where the parametrized random variables are repeated for each individual in the population associated with the parameter for which the constraint is true. The above example, if there were 10,000 people, would represent a belief network with 60,001 nodes.

3 First order variable elimination

The problem that we consider is: given a parametrized belief network, a set of observations, and a (possibly parametrized) query to determine the conditional probability of the instances of a query given the observations. We assume that the evidence is a conjunction of existentially quantified parametrized primitive propositions.

We consider the problem of parametric probabilistic inference in two stages: first, where every parameter that appears in the parents of a node also appears in the node and where the observations are variable-free. In this case, when we ground out the theory, each random variable only has a limited number of parents. In Section 3.5 we present the second case where we have parents that contain extra parameters; in this case we need a way to aggregate over populations. That section also considers existential observations and queries.

The algorithm is based on variable elimination, VE, [Zhang and Poole, 1996], where we eliminate the non-observed non-query random variables one at a time. For first-order VE (FOVE) we eliminate all the instances of a functor at once³.

There is a strong relationship between this work and lifting in theorem proving [Chang and Lee, 1973]: given a ground proof procedure, construct a proof procedure with logical variables (or in our case with parametrized random variables). In general, correctness can be shown by proving that we get the same answer as if we first grounded the theory and then carried out variable elimination. See Figure 2.

3.1 Parametric Factors

In VE, a factor is the unit of data used during computation. A VE factor is a function from a set of random variables into a non-negative real number. The initial factors are the conditional probabilities. The main operations are multiplying factors and summing out random variables from factors. After conditioning on the observed random variables and summing out the non-observed, non-query random variables, we can extract posterior probabilities from the remaining factors by multiplying them and normalizing the remaining factor.

In first-order variable elimination, we use a generalization of a factor where we want to treat the many instances of factors

³In general, we can potentially eliminate some instances of a functor, but not necessarily all instances, and leave the other instances to be eliminated later. In this paper we will only do this when eliminating all instances except for a query instance.

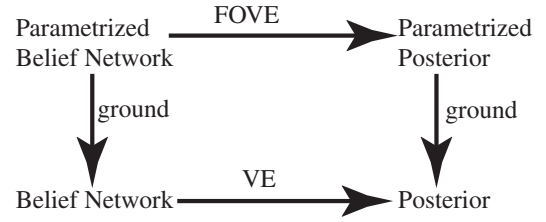


Figure 2: We design FOVE so that we get the same answer as if we had grounded the representation and carried out variable elimination.

as a unit. We only instantiate parameters when we need to. In general we reason with all of the individuals (except the ones that we know extra information about) as a unit.

A **parametric factor** or **parfactor** is a triple $\langle C, V, t \rangle$ where C is a set of constraints on parameters, V is a set of parametrized random variables and t is a table representing a factor from the random variables to the non-negative reals.

Intuitively the parametric factor represents all of the ground instances of the factor where the instantiation of the parameters satisfies the constraints.

Example 3 A parametric factor that represents one of the conditional probability tables of Figure 1 is:

$$\langle \{\}, \{\text{hair_colour}(X), \text{conservativeness}\}, t \rangle$$

where t is the table that represents a function from *hair_colour* and *conservativeness* into non-negative numbers. t is *not* indexed by X . t looks like:

<i>hair_colour</i>	<i>conservativeness</i>	Val
purple	conservative	0.001
purple	liberal	0.01
blue	conservative	0.1
blue	liberal	0.05
...		

When there are two instances of the same parametrized random variable in V ; in this case we need to mark the random variable in the table to distinguish these instances.

Example 4 Suppose that whether person X is a friend of Y depends on whether X likes Y and whether Y likes X . There are two distinct cases here, the first is when $X = Y$, in which case *friends*(X, Y) has one parent, and the second is when $X \neq Y$, in which case *friends*(X, Y) has two parents. To represent this, we could use

$$\langle \{\}, \{\text{friends}(X, X), \text{likes}(X, X)\}, t_1 \rangle$$

$$\langle \{X \neq Y\}, \{\text{friends}(X, Y), \text{likes}_1(X, Y), \text{likes}_2(Y, X)\}, t_2 \rangle$$

where the subscripts in *likes*₁ and *likes*₂ represent different instances of *likes* in table t_2 . That is, t_2 is a factor on *friends*, *likes*₁ and *likes*₂. When we eliminate all of the *likes* relationships, we have to consider *likes*₁ and *likes*₂.

3.2 Splitting

The foundation of parametric variable elimination is the splitting operation. Splitting plays the analogous role to applying substitutions in theorem proving, except that we not only have to be concerned about the instance created, but also about the instances left over.

Definition 1 Suppose parametric factor $\langle C, V, t \rangle$ contains parameter X . A **split** of $\langle C, V, t \rangle$ on $X = \gamma$, where γ is either a constant or another parameter, and C does not contain $X \neq \gamma$ (or $\gamma \neq X$), results in the two parametric factors:

$$\langle C[X/\gamma], V[X/\gamma], t \rangle \\ \langle \{X \neq \gamma\} \cup C, V, t \rangle$$

where $V[X/\gamma]$ is the same as V , but with γ replacing every instance of X (i.e., we substitute γ for X). The second parametric factor is called a **residual** parametric factor.

A **substitution** is of the form $\{X_1/t_1, \dots, X_k/t_k\}$ where the X_i are distinct variables and the t_i are terms. We assume that all substitutions are in normal form: t_i does not contain X_j for any i and j . The substitutions resulting from standard unification algorithms are in normal form [Chang and Lee, 1973]. If a substitution is in normal form, we get the same result from replacing each X_i by the corresponding t_i sequentially (in any order) or in parallel.

Instead of just applying substitutions as in normal theorem proving, we need to split. The general idea is that whenever applying a substitution to a parfactor restricts the set of ground instances, we need to split the parfactor. We don't need to split when just renaming variables or substituting a value for a variable that doesn't appear in the parfactor.

We can **split** a parfactor ϕ on substitution θ , by totally ordering the elements of the substitution⁴, $\{X_1/t_1, \dots, X_k/t_k\}$, by carrying out the following procedure which results in a final instance of ϕ and a set of residual parfactored:

For i from 1 to k
 If X_i is a variable in ϕ
 then if t_i is a variable in ϕ or t_i is not a variable
 then split ϕ on $X_i = t_i$
 else replace all occurrences of X_i by t_i ;
 t_i becomes a variable of ϕ .

Note that the final value of ϕ is the same as if we had applied the substitution to ϕ , but we also create residuals.

There is a close relationship between the splitting on equality in this paper and the splitting on the value of a variable in contextual variable elimination [Poole and Zhang, 2003].

3.3 Observations

When we observe a ground value, we carry out the analogous operation to VE. We project the tables onto the observed values. However we must first split to ensure that we only affect the appropriate ground variables.

Example 5 If we condition on the fact that *Joe* has purple hair, a purple car, and a shoe size of 12, we now reason separately about *Joe* than we do about the other individuals who can be treated as a group.

The parametric factor of example 3 becomes the two parametric factors:

$$\langle \{\}, \{conservativeness\}, t' \rangle \\ \langle \{X \neq joe\}, \{hair_colour(X), conservativeness\}, t \rangle$$

with the same t as above, and t' is t where we select $hair_colour = purple$ and project onto $conservativeness$.

⁴The total order does not affect the instance created but does affect which residuals are created.

3.4 Multiplying Parametric Factors

In VE, when we eliminate a variable, we multiply all of the factors that contain that variable then sum out the variable from the resultant factor.

As in variable elimination, we need to multiply the factors that contain the variable to be eliminated. In parametric variable elimination, given two parametric factors, some instances may need to be multiplied, and some instances may not. Also, the dimension of the resulting factors can be different for different instances.

The product of two parametric factors, in general, results in a set of parametric factors. Intuitively, we keep splitting the parametric factors (and renaming parameters) until we can guarantee that we get parametric factors that, if grounded, result in the same factors as if we were to first ground the factors and then multiply.

Determining which parfactored to multiply

Suppose there are two parfactored $\phi_1 = \langle C_1, \{p_1\} \cup V_1, t_1 \rangle$ and $\phi_2 = \langle C_2, \{p_2\} \cup V_2, t_2 \rangle$, with variables renamed to be different, where p_1 and p_2 are unifiable instances of $p(X_1, \dots, X_k)$, where p is to be eliminated. Let $\theta = mgu(p_1, p_2)$. We will split ϕ_1 on θ and split ϕ_2 on θ , putting all the residuals in the set Φ of all parfactored. All instances of the resulting non-residual parfactored would be multiplied in VE.

The following abstract example is designed to show what needs to be considered when multiplying parfactored. It isn't meant to be meaningful.

Example 6 Suppose we were to eliminate p and multiply the two parametric factors:

$$\langle \{\}, \{p(X, a), q(Y, c), s(X, Y)\}, t_1 \rangle \quad (1)$$

$$\langle \{W \neq d\}, \{p(b, Z), q(W, T), r(W, T)\}, t_2 \rangle \quad (2)$$

If we were to ground the parameters some of the instances of these would be multiplied in VE and some of them wouldn't. Unification finds the most general instances that are identical.

We unify $p(X, a)$ and $p(b, Z)$ resulting in the substitution $\theta = \{X/b, Z/a\}$.

We can split parametric factor (1) on θ resulting in

$$\langle \{\}, \{p(b, a), q(Y, c), s(b, Y)\}, t_1 \rangle \quad (3)$$

$$\langle \{X \neq b\}, \{p(X, a), q(Y, c), s(X, Y)\}, t_1 \rangle \quad (4)$$

Parametric factor (4) is a **residual** parametric factor. No instance of parametric factor (4) ever needs to be multiplied by any instance of parametric factor (2) when eliminating p and doesn't participate further in the product.

Similarly, we can split (2) on θ resulting in:

$$\langle \{W \neq d\}, \{p(b, a), q(W, T), r(W, T)\}, t_2 \rangle \quad (5)$$

$$\langle \{Z \neq a, W \neq d\}, \{p(b, Z), q(W, T), r(W, T)\}, t_2 \rangle \quad (6)$$

Parametric factor (6) is a residual factor and doesn't participate further in the elimination of p . All instances of (3) and (5) would be multiplied together when eliminating $p(b, a)$ in variable elimination.

Determining the dimensionality of the product

In Example 6, all instances of parametric factors (3) and (5) would be multiplied if we were to ground parametric factors (1) and (2) and carry out VE. However, not all of the product factors have the same dimension; some have two different q

instances, and some have one. We need to do more splitting to ensure that all of the products have the form of parametric factors.

Suppose we have parafactors $\phi' = \langle C', V', t' \rangle$ and $\phi'' = \langle C'', V'', t'' \rangle$ that need to be multiplied. If for all $p' \in V'$ and for all $p'' \in V''$, either p' and p'' are identical or non-unifiable or if $mgu(p', p'')$ is incompatible with the constraints, we know that all instances of their product has the same dimension. If there is a $p' \in V'$ and $p'' \in V''$ that are not identical but unify with mgu consistent with C' and C'' , we can split ϕ' and ϕ'' on $mgu(p', p'')$. The resulting instances and residuals either have identical instances of p' and p'' or non-unifiable instances. We then multiply each instance created from ϕ' by each instance created from ϕ'' .

When we know all the instances $\phi' = \langle C', V', t' \rangle$ and $\phi'' = \langle C'', V'', t'' \rangle$ have the same dimension, we create the parafactor $\langle C' \cup C'', V' \cup V'', t' \otimes t'' \rangle$ where $t' \otimes t''$ is the product of the tables for t' and t'' where we maintain one dimension for each member of $V' \cup V''$. Thus, those members in common in V' and V'' are treated as the same variable in the product, but those members that don't unify, even with the same functor, in V' and V'' are treated as different variables in the product table.

Example 7 When we need to multiple parafactors (3) and (5), we notice that $q(Y, c)$ and $q(W, T)$ unify (with unifier $\{W/Y, T/c\}$). We thus split parametric factor (5) on $T = c$, and rename W to Y , replacing (5) with:

$$\langle \{Y \neq d\}, \{p(b, a), q(Y, c), r(Y, c)\}, t_2 \rangle \quad (7)$$

$$\langle \{W \neq d, T \neq c\}, \{p(b, a), q(W, T), r(W, T)\}, t_2 \rangle \quad (8)$$

We can now multiply parametric factors (3) and (7) producing:

$$\langle \{Y \neq d\}, \{p(b, a), q(Y, c), r(Y, c), s(X, Y)\}, t_1 \otimes t_2 \rangle \quad (9)$$

where $t_1 \otimes t_2$ is the product of factors. Parametric factor (9) represents all of the factors of dimension four created by multiplying factors that are instances of parametric factors (1) and (2).

We also multiply parametric factors (3) and (8) producing:

$$\langle \{W \neq d, T \neq c\}, \{p(b, a), q_1(Y, c), s(X, Y), q_2(W, T), r(W, T)\}, t'_1 \otimes t'_2 \rangle \quad (10)$$

where t'_1 is the factor t_1 , but with q labelled as q_1 and t'_2 is the factor t_2 , but with q labelled as q_2 . Thus $t'_1 \otimes t'_2$ is a factor on p, q_1, s, q_2, r . Parametric factor (10) represents all of the factors of dimension five created by multiplying factors that are instances of parametric factors (1) and (2).

While this may seem very complicated, remember that parametric factor (10) represents $m^2(m-1)^2$ factors (assuming that all populations have size m). Even if m is 10, this is 8100 factors.

When we have multiplied all of the appropriate parafactors we are ready to sum out the variables being eliminated. We must remember that when we are eliminating $p(X_1, \dots, X_k)$, we are not just eliminating one random variable, but we are eliminating a number of variables equal to the product of the population sizes of $X_1, \dots, X_k$. If some parameters only appear in the parametrized random variable that is being eliminated, in the grounding we are multiplying the number of factors equal to the effective population size of those parameters.

To eliminate instances of $p(X_1, \dots, X_k)$ from multiset Φ of parafactors:

```

while there are parafactors  $\phi_1 = \langle C_1, V_1, t_1 \rangle$  and
     $\phi_2 = \langle C_2, V_2, t_2 \rangle$  in  $\Phi$ 
    (with variables renamed to be different)
    such that  $V_1 = \{p_1\} \cup V'_1$  and  $V_2 = \{p_2\} \cup V'_2$  and
     $p_1$  and  $p_2$  are instances of  $p(X_1, \dots, X_k)$  that unify
    and  $\theta = mgu(p_1, p_2)$  is consistent with  $C_1$  and  $C_2$  {
    remove  $\phi_1$  and  $\phi_2$  from  $\Phi$ ;
    split  $\phi_1$  on  $\theta$  with residuals going into  $\Phi$ ;
    split  $\phi_2$  on  $\theta$  with residuals going into  $\Phi$ ;
    let  $\Phi_1 = \{\phi_1\}$ ,  $\Phi_2 = \{\phi_2\}$ ;
    while  $\exists \phi' = \langle C', \{p'\} \cup V', t' \rangle \in \Phi_1$  and
         $\phi'' = \langle C'', \{p''\} \cup V'', t'' \rangle \in \Phi_2$  and
         $\phi'$  and  $\phi''$  are not identical and
         $\theta' = mgu(\phi', \phi'')$  and
         $\theta'$  is consistent with  $C'$  and  $C''$  {
        split  $\phi'$  on  $\theta'$  putting result and
            residuals in  $\Phi_1$ ;
        split  $\phi''$  on  $\theta'$  putting result and
            residuals in  $\Phi_2$ ;
        }
    for every  $\langle C', V', t' \rangle \in \Phi_1$  and  $\langle C'', V'', t'' \rangle \in \Phi_2$  {
        add  $\langle C' \cup C'', V' \cup V'', t' \otimes t'' \rangle$  to  $\Phi$ 
    }
}

for every  $\langle C, \{p'\} \cup V, t \rangle \in \Phi$ 
    where  $p'$  is an instance of  $p(X_1, \dots, X_k)$  {
    let  $m$  be the effective population size of
        the parameters in  $p'$  not in  $V$ 
    replace  $\langle C, \{p'\} \cup V, t \rangle$  with  $\langle C', V, (\sum_{p'} t)^m \rangle$  in  $\Phi$ 
        where  $C'$  is  $C$  restricted to the parameters in  $V$ 
    }

```

Figure 3: Eliminating all instances of functor p

The effective population size is the product of the populations of the parameters less the number excluded by the constraints. Thus we have to take each element in the table and put it to the power of the effective population size.

3.5 Aggregation over populations

When we have parameters in the parents of a node and not in the node itself, the number of parents grows with the population size, and we need to specify how a node is a function of its parents. There are many possibilities, such as a node being the “logical or” of its parents, the “logical and”, the max of its parents, the average if its parents, true iff greater than k of its parents are true, or the vote of its parents (according to some voting scheme, for example when the majority wins), or some other function. Zhang and Poole [1996] give an analysis where there is an arbitrary associative and commutative operator between the parents. In order to make the presentation simpler, in this paper we assume that the operator is a logical “or” [Díez and Galán, 2002]. It is straightforward to use the techniques of Zhang and Poole [1996] to extend this to other operators.

We can transform the problem into one of the form $p(\bar{Z})$ with parents $q(\bar{Z}, \bar{X})$, where p is the “logical or”, over all of the $\bar{X}$ values.

To eliminate q , we multiply all of the compatible clauses that contain q , and as we eliminate each q we accumulate the probability over the q ’s (as in [Díez and Galán, 2002; Zhang and Poole, 1996]). The only difference is when there is a free (perhaps with inequality constraints) parameter in the $\bar{X}$ ’s. In this case we need to do the “or” over the effective population size. We can determine the effective population size by determining the effective population size of each free parameter by subtracting the number of excluded values from the population (e.g., if we have $X \neq a$ and $X \neq b$ with a population size of 10, we have an effective population size of $10 - 2 = 8$) and multiplying by the effective populations of all of the free parameters.

If we have an effective population size of m and each one contributes a probability of p then, assuming they are all independent, the probability that they are all false is $(1 - p)^m$. So the probability that at least one is true (i.e., the logical “or”) is $1 - (1 - p)^m$. If the effective population size is countably infinite, the probability that at least one is true is 1 if $p > 0$ and is 0 if $p = 0$.

However, we cannot assume the instances of the q ’s are independent. They are only dependent if they either have (1) a common ancestor in the grounding or (2) a common observed descendent. If we condition on the observation after eliminating the q ’s, we can get around the second condition. The only way that the q ’s can have a common ancestor is if there is an ancestor that doesn’t involve one of the parameters. If we make sure that we eliminate the q ’s before we eliminate the common ancestors that separate the other common ancestors, then we can just use the equation above that assumes independence. Effectively we are doing the logical “or” for each value of the ancestors, and we know they are independent for each value of the ancestors.

If we have an existential observation or query (e.g., conditioning on the fact that someone who fits a certain description is guilty), we need to construct the “or” and either condition on or query the resulting node.

Example 8 Let’s return to Example 1 (see Figure 1). Suppose that as well as observing the hair-colour, car-colour, and shoe size of Joe, we also observed that there exists a person who is guilty and fits a certain description of hair-colour, car-colour, height. This is depicted in Figure 4.

Suppose we want to compute whether Joe is guilty given Joe’s hair-colour, car-colour and shoe size and given the witness description. We can first instantiate all of the observed random variables except the witness observation. This splits off *joe* as a special case. We can now eliminate all of the random variables except for *town_conservativeness*, *guilty(joe)*, *descn(X)* (for $X = \text{joe}$ and $X \neq \text{joe}$) and *witness*. This results in two parametric factors:

$$\langle \{\}, \{\text{guilty(joe), descn(joe), conservativeness}\}, t_1 \rangle$$

$$\langle \{X \neq \text{joe}\}, \{\text{descn(X), conservativeness}\}, t_2 \rangle$$

We can now eliminate *descn(X)* for $X \neq \text{joe}$ and notice that the instances are independent given *conservativeness*. This

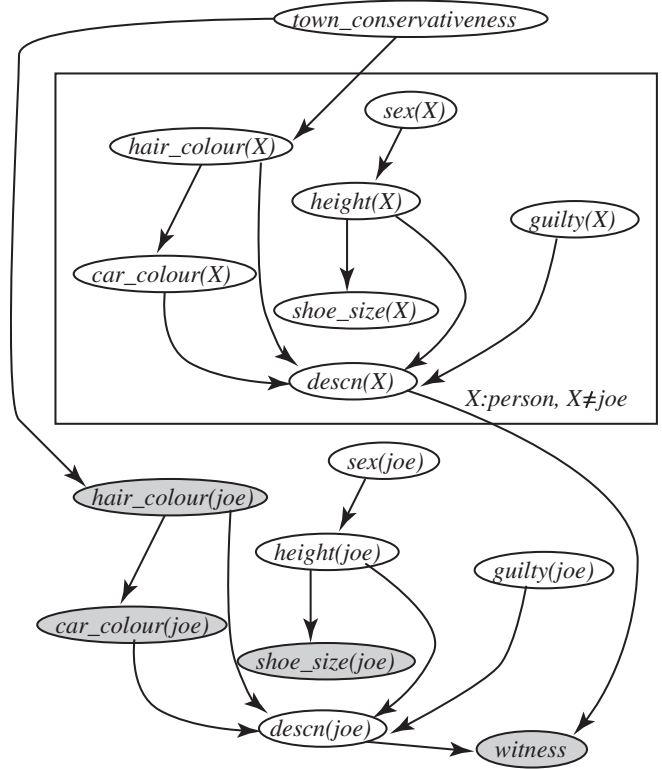


Figure 4: Robbing example with the witness observation.

results in the parametric factor:

$$\langle \{\}, \{\text{guilty(joe), witness, conservativeness}\}, t_3 \rangle$$

We can now sum out *conservativeness* and condition on *witness* and end up with a parametric factor

$$\langle \{\}, \{\text{guilty(joe)}\}, t_4 \rangle$$

We can now determine the probability that Joe is guilty from normalising t_4 . If we were to carry out this computation leaving the population as a parameter, the probability of *guilty(joe)* can be computed as a function of the population. We get a result that looks like Figure 5. The graph has this shape because it is the linear combination of exponential functions (for each value of *town_conservativeness* we have an exponential distribution).

4 Conclusion

This paper has made three main contributions:

- a way to do inference over populations without grounding out the theory and a way to use unification, where as well as the unifiers we also need to take into account the residuals;
- the idea that we need to take population size into account when we have aggregation over populations either as part of the model or as an observation; and
- a way to handle existential observations where we know someone exists but don’t know who it is.

This paper extends the inference in object-oriented Bayesian networks [Pfeffer et al., 1999] where the reasoning with

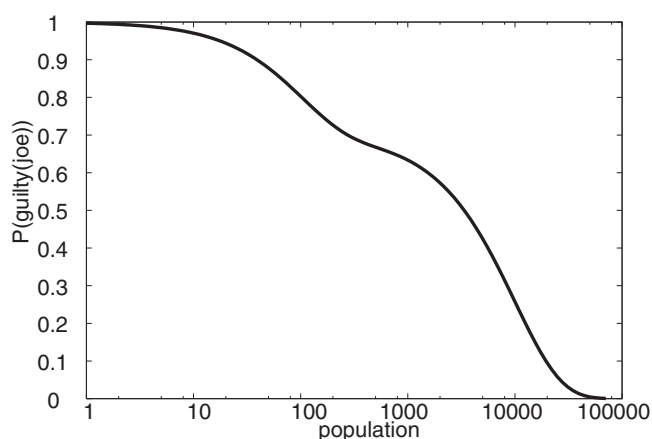


Figure 5: Probability of *guilty(joe)* for various populations.

generic class models corresponds to reasoning with free parameters in this paper. They do not use the power of unification as in this paper.

This paper contradicts the pessimistic conclusions of Jaeger [2000], but not the results. While in the worst case we may effectively ground the representation, in many cases we can do much better. How much we save in practice is still an open question.

There are still a number of open questions:

- What are good elimination orderings? We don't have to eliminate all instances of a functor at once.
- How to utilize other combination rules (apart from "or"). While the description here was in terms of noisy-or [Díez and Galán, 2002] or inter-causal independencies [Zhang and Poole, 1996], the actual use of splitting is much closer to the work on contextual independence [Poole and Zhang, 2003].
- We are interested in combining the work in this paper with richer languages (e.g., Poole [1993]), where much of the power comes from mixing contextual independence and free variables and allowing for function symbols and recursion.
- It is also interesting to think about combining this with MCMC [Pasula and Russell, 2001], however it may not be straightforward because we are representing a set of individuals as a unit (and so are completely dependent), which may cause problems when we separate one individual from the class and reason about that individual separately.

Acknowledgements

This work was supported by NSERC Research Grant OG-POO44121 and The Institute for Robotics and Intelligent Systems. Thanks to Valerie McRae and Mark Paskin for valuable comments.

References

Bacchus, F. [1990]. *Representing and Reasoning with Uncertain Knowledge*, MIT Press, Cambridge, Massachusetts.

- Breese, J. S. [1992]. Construction of belief and decision networks, *Computational Intelligence* **8**(4): 624–647.
- Buntine, W. L. [1994]. Operations for learning with graphical models, *Journal of Artificial Intelligence Research* **2**: 159–225.
- Chang, C. L. and Lee, R. C. T. [1973]. *Symbolic Logical and Mechanical Theorem Proving*, Academic Press, New York.
- Díez, F. J. and Galán, S. F. [2002]. Efficient computation for the noisy max, *International Journal of Intelligent Systems* p. to appear.
- Halpern, J. Y. [1990]. An analysis of first-order logics of probability, *Artificial Intelligence* **46**(3): 311–350.
- Horsch, M. and Poole, D. [1990]. A dynamic approach to probabilistic inference using Bayesian networks, *Proc. Sixth Conference on Uncertainty in AI*, Boston, pp. 155–161.
- Jaeger, M. [2000]. On the complexity of inference about probabilistic relational models, *Artificial Intelligence* **117**: 297–308.
- Kersting, K. and De Raedt, L. [2000]. Bayesian logic programs, *Linköping Electronic Articles in Computer and Information Science* **5**(34).
URL: <http://www.ep.liu.se/ea/cis/2000/034/>
- Koller, D. and Pfeffer, A. [1998]. Probabilistic frame-based systems, *AAAI-98*, AAAI Press, Madison, Wisconsin.
- Pasula, H. and Russell, S. [2001]. Approximate inference for first-order probabilistic languages, *IJCAI-01*, Seattle, WA, pp. 741–748.
- Pearl, J. [1988]. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, Morgan Kaufmann, San Mateo, CA.
- Pfeffer, A., Koller, D., Milch, B., and Takusagawa, K. [1999]. SPOOK: A system for probabilistic object-oriented knowledge representation, *UAI-99*, Stockholm, Sweden, pp. 541–550.
- Poole, D. [1993]. Probabilistic Horn abduction and Bayesian networks, *Artificial Intelligence* **64**(1): 81–129.
- Poole, D. and Zhang, N. L. [2003]. Exploiting contextual independence in probabilistic inference, *Journal of Artificial Intelligence Research* **18**: 263–313.
- Robinson, J. A. [1965]. A machine-oriented logic based on the resolution principle, *Journal ACM* **12**(1): 23–41.
- Wellman, M., Breese, J. and Goldman, P. [1992]. From knowledge bases to decision models, *Knowledge Engineering Review* **7**(1): 35–53.
- Zhang, N. and Poole, D. [1996]. Exploiting causal independence in Bayesian network inference, *Journal of Artificial Intelligence Research* **5**: 301–328.

Dynamic Probabilistic Relational Models

Sumit Sanghai Pedro Domingos Daniel Weld

Department of Computer Science and Engineering

University of Washington

Seattle, WA 98195-2350

{sanghai,pedrod,weld}@cs.washington.edu

Abstract

Intelligent agents must function in an uncertain world, containing multiple objects and relations that change over time. Unfortunately, no representation is currently available that can handle all these issues, while allowing for principled and efficient inference. This paper addresses this need by introducing dynamic probabilistic relational models (DPRMs). DPRMs are an extension of dynamic Bayesian networks (DBNs) where each time slice (and its dependences on previous slices) is represented by a probabilistic relational model (PRM). Particle filtering, the standard method for inference in DBNs, has severe limitations when applied to DPRMs, but we are able to greatly improve its performance through a form of relational Rao-Blackwellisation. Further gains in efficiency are obtained through the use of abstraction trees, a novel data structure. We successfully apply DPRMs to execution monitoring and fault diagnosis of an assembly plan, in which a complex product is gradually constructed from subparts.

1 Introduction

Sequential phenomena abound in the world, and uncertainty is a common feature of them. Currently the most powerful representation available for such phenomena is dynamic Bayesian networks, or DBNs [Dean and Kanazawa, 1989]. DBNs represent the state of the world as a set of variables, and model the probabilistic dependencies of the variables within and between time steps. While a major advance over previous approaches, DBNs are still unable to compactly represent many real-world domains. In particular, domains can contain multiple objects and classes of objects, as well as multiple kinds of relations among them; and objects and relations can appear and disappear over time. For example, manufacturing plants assemble complex artifacts (e.g., cars, computers, aircraft) from large numbers of component parts, using multiple kinds of machines and operations. Capturing such a domain in a DBN would require exhaustively representing all possible objects and relations among them. This raises two problems. The first one is that the computational cost of using such a DBN would likely be prohibitive. The second is that reducing the rich structure of the domain to a very large, “flat” DBN would render it essentially incomprehensible to

human beings. This paper addresses these two problems by introducing an extension of DBNs that exposes the domain’s relational structure, and by developing methods for efficient inference in this representation.

Formalisms that can represent objects and relations, as opposed to just variables, have a long history in AI. Recently, significant progress has been made in combining them with a principled treatment of uncertainty. In particular, probabilistic relational models or PRMs [Friedman *et al.*, 1999] are an extension of Bayesian networks that allows reasoning with classes, objects and relations. The representation we introduce in this paper extends PRMs to sequential problems in the same way that DBNs extend Bayesian networks. We thus call it *dynamic probabilistic relational models*, or DPRMs. We develop an efficient inference procedure for DPRMs by adapting Rao-Blackwellised particle filtering, a state-of-the-art inference method for DBNs [Murphy and Russell, 2001]. We introduce *abstraction trees* as a data structure to reduce the computational cost of inference in DPRMs.

Early fault detection in complex manufacturing processes can greatly reduce their cost. In this paper we apply DPRMs to monitoring the execution of assembly plans, and show that our inference methods scale to problems with over a thousand objects and thousands of steps. Other domains where we envisage DPRMs being useful include robot control, vision in motion, language processing, computational modeling of markets, battlefield management, cell biology, ecosystem modeling, and the Web.

The rest of the paper is structured as follows. The next two sections briefly review DBNs and PRMs. We then introduce DPRMs and methods for inference in them. The following section reports on our experimental study in assembly plan monitoring. The paper concludes with a discussion of related and future work.

2 Dynamic Bayesian Networks

A *Bayesian network* encodes the joint probability distribution of a set of variables, $\{Z_1, \dots, Z_d\}$, as a directed acyclic graph and a set of conditional probability models. Each node corresponds to a variable, and the model associated with it allows us to compute the probability of a state of the variable given the state of its parents. The set of parents of Z_i , denoted $Pa(Z_i)$, is the set of nodes with an arc to Z_i in the graph. The structure of the network encodes the as-

sertion that each node is conditionally independent of its non-descendants given its parents. The probability of an arbitrary event $Z = (Z_1, \dots, Z_d)$ can then be computed as $P(Z) = \prod_{i=1}^d P(Z_i | Pa(Z_i))$.

Dynamic Bayesian Networks (DBNs) are an extension of Bayesian networks for modeling dynamic systems. In a DBN, the state at time t is represented by a set of random variables $Z_t = \{Z_{1,t}, \dots, Z_{d,t}\}$. The state at time t is dependent on the states at previous time steps. Typically, we assume that each state only depends on the immediately preceding state (i.e., the system is first-order Markovian), and thus we need to represent the transition distribution $P(Z_{t+1} | Z_t)$. This can be done using a two-time-slice Bayesian network fragment (2TBN) B_{t+1} , which contains variables from Z_{t+1} whose parents are variables from Z_t and/or Z_{t+1} , and variables from Z_t without any parents. Typically, we also assume that the process is stationary, i.e., the transition models for all time slices are identical: $B_1 = B_2 = \dots = B_t = B_{\infty}$. Thus a DBN is defined to be a pair of Bayesian networks (B_0, B_{∞}) , where B_0 represents the initial distribution $P(Z_0)$, and B_{∞} is a two-time-slice Bayesian network, which as discussed above defines the transition distribution $P(Z_{t+1} | Z_t)$.

The set Z_t is commonly divided into two sets: the unobserved state variables X_t and the observed variables Y_t . The observed variables Y_t are assumed to depend only on the current state variables X_t . The joint distribution represented by a DBN can then be obtained by unrolling the 2TBN:

$$\begin{aligned} P(X_0, X_1, \dots, X_T, Y_0, Y_1, \dots, Y_T) \\ = P(X_0)P(Y_0|X_0) \prod_{t=1}^T P(X_t|X_{t-1})P(Y_t|X_t) \end{aligned}$$

Various types of inference in DBNs are possible. One of the most useful is state monitoring (also known as filtering or tracking), where the goal is to estimate the current state of the world given the observations made up to the present, i.e., to compute the distribution $P(X_T | Y_0, Y_1, \dots, Y_T)$. Proper state monitoring is a necessary precondition for rational decision-making in dynamic domains. Inference in DBNs is NP-complete, and thus we must resort to approximate methods, of which the most widely used one is *particle filtering* [Doucet et al., 2001]. Particle filtering is a stochastic algorithm which maintains a set of particles (samples) $x_t^1, x_t^2, \dots, x_t^N$ to approximately represent the distribution of possible states at time t given the observations. Each particle x_t^i contains a complete instance of the current state, i.e., a sampled value for each state variable. The current distribution is then approximated by

$$P(X_T = x | Y_0, Y_1, \dots, Y_T) = \frac{1}{N} \sum_{i=1}^N \delta(x_T^i = x)$$

where $\delta(x_T^i = x)$ is 1 if the state represented by x_T^i is same as x , and 0 otherwise. The particle filter starts by generating N particles according to the initial distribution $P(X_0)$. Then, at each step, it first generates the next state x_{t+1}^i for each particle i by sampling from $P(X_{t+1}^i | X_t^i)$. It then weights these samples according to the likelihood they assign to the observations, $P(Y_{t+1} | X_{t+1}^i)$, and resamples N particles from

this weighted distribution. The particles will thus tend to stay clustered in the more probable regions of the state space, according to the observations.

Although particle filtering has scored impressive successes in many practical applications, it also has some significant limitations. One that is of particular concern to us here is that it tends to perform poorly in high-dimensional state spaces. This is because the number of particles required to maintain a good approximation to the state distribution grows very rapidly with the dimensionality. This problem can be greatly attenuated by analytically marginalizing out some of the variables, a technique known as *Rao-Blackwellisation* [Murphy and Russell, 2001]. Suppose the state space X_t can be divided into two subspaces U_t and V_t such that $P(V_t | U_t, Y_1, \dots, Y_t)$ can be computed analytically and efficiently. Then we only need to sample from the smaller space U_t , requiring far fewer particles to obtain the same degree of approximation. Each particle is now composed of a sample from $P(U_t | Y_1, \dots, Y_t)$ plus a parametric representation of $P(V_t | U_t, Y_1, \dots, Y_t)$. For example, if the variables in V_t are discrete and independent of each other given U_t , we can store for each variable the vector of parameters of the corresponding multinomial distribution (i.e., the probability of each value).

3 Probabilistic Relational Models

A *relational schema* is a set of classes $\mathcal{C} = \{C_1, C_2, \dots, C_k\}$, where each class C is associated with a set of *propositional attributes* $\mathcal{A}(C)$ and a set of *relational attributes* or *reference slots* $\mathcal{R}(C)$. The propositional attribute A of class C is denoted $C.A$, and its domain (assumed finite) is denoted $V(C.A)$. The relational attribute R of C is denoted $C.R$, and its domain is the power set $2^{C'}$ of a target class $C' \in \mathcal{C}$. In other words, $C.R$ is a set of objects belonging to some class C' .¹ For example, the *Aircraft* schema might be used to represent partially or completely assembled aircraft, with classes corresponding to different types of parts like metal sheets, nuts and bolts. The propositional attributes of a bolt might include its color, weight, and dimensions, and its relational attributes might include the nut it is attached to and the two metal sheets it is bolting. An *instantiation* of a schema is a set of objects, each object belonging to some class $C \in \mathcal{C}$, with all propositional and relational attributes of each object specified. For example, an instantiation of the aircraft schema might be a particular airplane, with all parts, their properties and their arrangement specified.

A *probabilistic relational model (PRM)* encodes a probability distribution over the set of all possible instantiations I of a schema [Friedman et al., 1999]. The *object skeleton* of an instantiation is the set of objects in it, with all attributes unspecified. The *relational skeleton* of an instantiation is the set of objects in it, with all relational attributes specified, and all propositional attributes unspecified. In the simplest case, the relational skeleton is assumed known, and the PRM specifies a probability distribution for each attribute A of each class C . The parents of each attribute (i.e., the variables it depends on) can be other attributes of C , or attributes of

¹ $C.R$ can also be defined as a function from C to $2^{C'}$, but we choose the simpler convention here.

classes that are related to C by some slot chain. A slot chain is a composition of relational attributes. In general, it must be used together with an *aggregation function* that reduces a variable number of values to a single value. For example, a parent of an attribute of a bolt in the aircraft schema might be $\text{avg}(\text{bolt.plate.nut.weight})$, the average weight of all the nuts on the metal plates that the bolt is attached to.

Definition 1 A *probabilistic relational model (PRM)* Π for a relational schema $\mathcal{S}$ is defined as follows. For each class C and each propositional attribute $A \in \mathcal{A}(C)$, we have:

- A set of parents $Pa(C.A) = \{Pa_1, Pa_2, \dots, Pa_l\}$, where each Pa_i has the form $C.B$ or $\gamma(C.\tau.B)$, where τ is a slot chain and $\gamma()$ is an aggregation function.
- A *conditional probability model* for $P(C.A|Pa(C.A))$. $\square$

Let $\mathcal{O}$ be the set of objects in the relational skeleton. The probability distribution over instantiations I of $\mathcal{S}$ represented by the PRM is then

$$P(I) = \prod_{obj \in \mathcal{O}} \prod_{A \in \mathcal{A}(obj)} P(obj.A|Pa(obj.A))$$

A PRM and relational skeleton can thus be unrolled into a large Bayesian network with one variable for each attribute of each object in the skeleton.² Only PRMs that correspond to Bayesian networks without cycles are valid.

More generally, only the object skeleton might be known, in which case the PRM also needs to specify a distribution over the relational attributes [Getoor *et al.*, 2001]. In the aircraft domain, a PRM might specify a distribution over the state of assembly of an airplane, with probabilities for different faults (e.g., a bolt is loose, the wrong plates have been bolted, etc.).

4 Dynamic Probabilistic Relational Models

In this section we extend PRMs to modeling dynamic systems, the same way that DBNs extend Bayesian networks. We begin with the observation that a DBN can be viewed as a special case of a PRM, whose schema contains only one class Z with propositional attributes $Z_1, \dots, Z_d$ and a single relational attribute *previous*. There is one object Z_t for each time slice, and the *previous* attribute connects it to the object in the previous time slice. Given a relational schema $\mathcal{S}$, we first extend each class C with the relational attribute $C.\text{previous}$, with domain C . As before, we initially assume that the relational skeleton at each time slice is known. We can then define two-time-slice PRMs and dynamic PRMs as follows.

Definition 2 A *two-time-slice PRM (2TPRM)* for a relational schema $\mathcal{S}$ is defined as follows. For each class C and each propositional attribute $A \in \mathcal{A}(C)$, we have:

- A set of parents $Pa(C.A) = \{Pa_1, Pa_2, \dots, Pa_l\}$, where each Pa_i has the form $C.B$ or $f(C.\tau.B)$, where τ is a slot chain containing the attribute *previous* at most once, and $f()$ is an aggregation function.
- A *conditional probability model* for $P(C.A|Pa(C.A))$. $\square$

²Plus auxiliary (deterministic) variables for the required aggregations, which we omit from the formula for simplicity.

Definition 3 A *dynamic probabilistic relational model (DPRM)* for a relational schema $\mathcal{S}$ is a pair (M_0, M_∞) , where M_0 is a PRM over I_0 , representing the distribution P_0 over the initial instantiation of $\mathcal{S}$, and M_∞ is a 2TPRM representing the transition distribution $P(I_t|I_{t-1})$ connecting successive instantiations of $\mathcal{S}$. $\square$

For any T , the distribution over $I_0, \dots, I_T$ is then given by

$$P(I_0, \dots, I_T) = P_0(I_0) \prod_{t=1}^T P(I_t|I_{t-1})$$

DPRMs are extended to the case where only the object skeleton for each time slice is known in the same way that PRMs are, by adding to Definition 2 a set of parents and conditional probability model for each relational attribute, where the parents can be in the same or the previous time slice. When the object skeleton is not known (e.g., if objects can appear and disappear over time), the 2TPRM includes in addition a Boolean existence variable for each possible object, again with parents from the same or the previous time slice.³ As with DBNs, we may wish to distinguish between observed and unobserved attributes of objects. In addition, we can consider an *Action* class with a single attribute whose domain is the set of actions that can be performed by some agent (e.g., painting a metal plate, or bolting two plates together). The distribution over instantiations in a time slice can then depend on the action performed in that time slice. For example, the action $\text{Bolt}(\text{Part1}, \text{Part2})$ may with high probability produce $\text{Part1.mate} = \{\text{Part2}\}$, and with lower probability set Part1.mate to some other object of Part2 's class (i.e., be improperly performed, resulting in a fault).

Just as a PRM can be unrolled into a Bayesian network, so can a DPRM be unrolled into a DBN. (Note, however, that this DBN may in general contain different variables in different time slices.) In principle, we can perform inference on this DBN using particle filtering. However, the filter is likely to perform poorly, because for non-trivial DPRMs its state space will be huge. Not only will it contain one variable for each attribute of each object of each class, but relational attributes will in general have very large domains. We overcome this by adapting Rao-Blackwellisation to the relational setting. We make the following (strong) assumptions:

1. Relational attributes with unknown values do not appear anywhere in the DPRM as parents of unobserved attributes, or in their slot chains.
2. Each reference slot can be occupied by at most one object.

Proposition 1 *Assumptions 1 and 2 imply that, given the propositional attributes and known relational attributes at times t and $t - 1$, the joint distribution of the unobserved relational attributes at time t is a product of multinomials, one for each attribute.*

Notice also that, by Assumption 1, unobserved propositional attributes can be sampled without regard to unobserved relational ones. Rao-Blackwellisation can now be applied

³Notice that the attributes of nonexistent objects need not be specified, because by definition no attributes of any other objects can depend on them [Getoor *et al.*, 2001].

with U_t as the propositional attributes of all objects and V_t as their relational attributes. A Rao-Blackwellised particle is composed of sampled values for all propositional attributes of all objects, plus a probability vector for each relational attribute of each object. The vector element corresponding to $obj.R[i]$ is the probability that relation R holds between obj and the i th object of the target class, conditioned on the values of the propositional attributes in the particle, etc.

Rao-Blackwellising the relational attributes can vastly reduce the size of the state space which particle filtering needs to sample. However, if the relational skeleton contains a large number of objects and relations, storing and updating all the requisite probabilities can still become quite expensive. This can be ameliorated if context-specific independencies exist, i.e., if a relational attribute is independent of some propositional attributes given assignments of values to others [Boutilier *et al.*, 1996]. We can then replace the vector of probabilities with a tree structure whose leaves represent probabilities for entire sets of objects. More precisely, we define the *abstraction tree* data structure for a relational attribute $obj.R$ with target class C' as follows. A node ν of the tree is composed of a probability p and a logical expression ϕ over the propositional attributes of the schema. Let $O_\nu(C')$ be the set of objects in C' that satisfy the ϕ 's of ν and all of ν 's ancestors. Then $p \stackrel{\text{def}}{=} \sum_{obj' \in O_\nu(C')} P(obj' \in (obj.R)_t \mid U_t)$. The root of an abstraction tree contains $\phi = \text{true}$. The children ν_i of a node ν contain expressions ϕ_i such that the $O_{\nu_i}(C')$ form a partition of $O_\nu(C')$. Each leaf of the tree stores a parametric distribution giving the probability that each object in the leaf is a member of $obj.R$, as a function of the object's propositional attributes. The probability that an arbitrary object $obj' \in C'$ is a member of $obj.R$ is found by starting at the root of the abstraction tree for $obj.R$, going to the child whose condition is satisfied by obj' , and so on recursively until a leaf is reached and the object's probability is read from the leaf distribution.

Initially, the abstraction tree consists only of the root, and as inference progresses it is gradually refined as dictated by the attributes that $C.R$ depends on. For example, suppose the first action to be performed is $Bolt(Part1, Part2)$, and with probability p_f the action is performed incorrectly. The faulty action consists of attaching $Part1$ to some other object of $Part2$'s class C' , with uniform probability over C' . Then two children ν_1 and ν_2 of the root of $Part1.mate$'s abstraction tree are created, with ϕ_1 specifying the singleton set $\{Part2\}$ and ϕ_2 its complement in C' , and with $p_1 = 1 - p_f$ and $p_2 = p_f$. The uniform distribution in leaf ν_2 has a single parameter, the probability $p = p_f / (|C'| - 1)$ that a given object in it is attached to $Part1$. This takes $O(1)$ space to store and $O(1)$ time to update, as opposed to $O(|C'|)$. If objects with different attributes have different probabilities of being bolted to $Part1$, a node for each relevant combination of attributes is created. Thus, if n_c is the number of such combinations, the storage and update time required for $Part1.mate$ are $O(n_c)$ instead of $O(|C'|)$. By design, $n_c \leq (|C'|)$; in the worst case, the tree will have one leaf per element of C' . As we will see in the next section, the use of abstraction trees can greatly reduce the computational cost of Rao-Blackwellised particle filtering in DPRMs.

5 Experiments

In this section we study the application of DPRMs to fault detection in complex assembly plans. We use a modified version of the *Schedule World* domain from the AIPS-2000 Planning Competition.⁴ The problem consists of generating a plan for assembly of objects with operations such as painting, polishing, etc. Each object has attributes such as surface type, color, hole size, etc. We add two relational operations to the domain: bolting and welding. We assume that actions may be faulty, with fault model described below. In our experiments, we first generate a plan using the FF planner [Hoffmann and Nebel, 2001]. We then monitor the plan's execution using particle filtering (PF), Rao-Blackwellised particle filtering (RBPF) and RBPF with abstraction trees.

We consider three classes of objects: *Plate*, *Bracket* and *Bolt*. *Plate* and *Bracket* have propositional attributes such as weight, shape, color, surface type, hole size and hole type, and relational attributes for the parts they are welded to and the bolts bolting them to other parts (e.g., *Plate73.bolt4* corresponds to the fourth bolt hole on plate 73). The *Bolt* class has propositional attributes such as size, type and weight. Propositional actions include painting, drilling and polishing, and change the propositional attributes of an object. The relational action *Bolt* sets a *bolt* attribute of a *Plate* or *Bracket* object to a *Bolt* object. The *Weld* action sets a *welded-to* attribute of a *Plate* or *Bracket* object to another *Plate* or *Bracket* object.

The fault model has a global parameter, the *fault probability* p_f . With probability $1 - p_f$, an action produces the intended effect. With probability p_f , one of several possible faults occurs. Propositional faults include a painting operation not being completed, the wrong color being used, the polish of an object being ruined, etc. The probability of different propositional faults depends on the properties of the object being acted on. Relational faults include bolting the wrong objects and welding the wrong objects. The probability of choosing a particular wrong object depends on its similarity to the intended object. Similarity depends on different propositional attributes for different actions and different classes of objects. Thus the probability of a particular wrong object being chosen is uniform across all objects with the same relevant attribute values.

The DPRM also includes the following observation model. There are two instances of each attribute: the true one, which is never observed, and the observed one, which is observed at selected time steps. Specifically, when an action is performed, all attributes of the objects involved in it are observed, and no others. Observations are noisy: with probability $1 - p_o$ the true value of the attribute is observed, and with probability p_o an incorrect value is observed. Incorrect values for propositional observations are chosen uniformly. Incorrect values for relational observations are chosen with a probability that depends on the similarity of the incorrect object to the intended one.

Notice that, if the domain consisted exclusively of the propositional attributes and actions on them, exact inference might be possible; however, the dependence of relational attributes and their observations on the propositional attributes

⁴URL: <http://www.cs.toronto.edu/aips2000>

creates complex dependencies between these, making approximate inference necessary.

A natural measure of the accuracy of an approximate inference procedure is the K-L divergence between the distribution it predicts and the actual one [Cover and Thomas, 2001]. However, computing it requires performing exact inference, which for non-trivial DPRMs is infeasible. Thus we estimate the K-L divergence by sampling, as follows. Let $D(p||\hat{p})$ be the K-L divergence between the true distribution p and its approximation $\hat{p}$, and let $\mathcal{X}$ be the domain over which the distribution is defined. Then

$$\begin{aligned} D(p||\hat{p}) &\stackrel{\text{def}}{=} \sum_{x \in \mathcal{X}} p(x) \log \frac{p(x)}{\hat{p}(x)} \\ &= \sum_{x \in \mathcal{X}} p(x) \log p(x) - \sum_{x \in \mathcal{X}} p(x) \log \hat{p}(x) \end{aligned}$$

The first term is simply the entropy of X , $H(X)$, and is a constant independent of the approximation method. Since we are mainly interested in measuring differences in performance between approximation methods, this term can be neglected. The K-L divergence can now be approximated in the usual way by taking S samples from the true distribution:

$$\hat{D}_H(p||\hat{p}) = -\frac{1}{S} \sum_{i=1}^S \log \hat{p}(x_i)$$

where $\hat{p}(x_i)$ is the probability of the i th sample according to the approximation procedure, and the H subscript indicates that the estimate of $D(p||\hat{p})$ is offset by $H(X)$. We thus evaluate the accuracy of PF and RBPF on a DPRM by generating $S = 10,000$ sequences of states and observations from the DPRM, passing the observations to the particle filter, inferring the marginal probability of the sampled value of each state variable at each step, plugging these values into the above formula, and averaging over all variables. Notice that $\hat{D}_H(p||\hat{p}) = \infty$ whenever a sampled value is not represented in any particle. The empirical estimates of the K-L divergence we obtain will be optimistic in the sense that the true K-L divergence may be infinity, but the estimated one will still be finite unless one of the values with zero predicted probability is sampled. This does not preclude a meaningful comparison between approximation methods, however, since on average the worse method should produce $\hat{D}_H(p||\hat{p}) = \infty$ earlier in the time sequence. We thus report both the average K-L divergence before it becomes infinity and the time step at which it becomes infinity, if any.

Figures 1 and 2 show the results of the experiments performed. The observation noise parameter p_o was set to the same value as the fault probability p_f throughout. One action is performed in each time step; thus the number of time steps is the length of the plan. The graphs show the K-L divergence of PF and RBPF at every 100th step (it is the same for RBPF with and without abstraction trees). Graphs are interrupted at the first point where the K-L divergence became infinite in any of the runs (once infinite, the K-L divergence never went back to being finite in any of the runs), and that point is labeled with the average time step at which the blow-up occurred. As can be seen, PF tends to diverge rapidly, while the

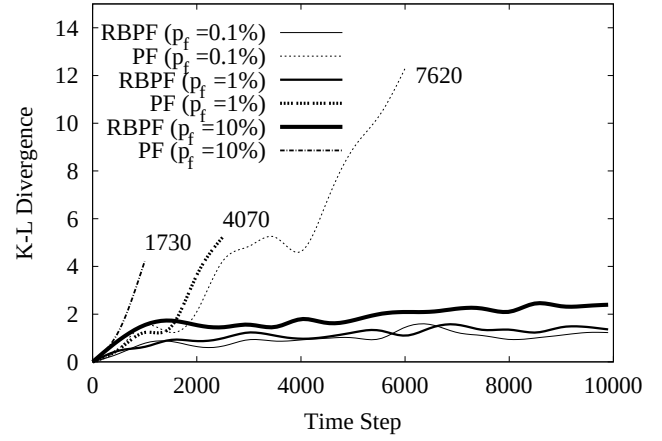


Figure 1: Comparison of RBPF (5000 particles) and PF (200,000 particles) for 1000 objects and varying fault probability.

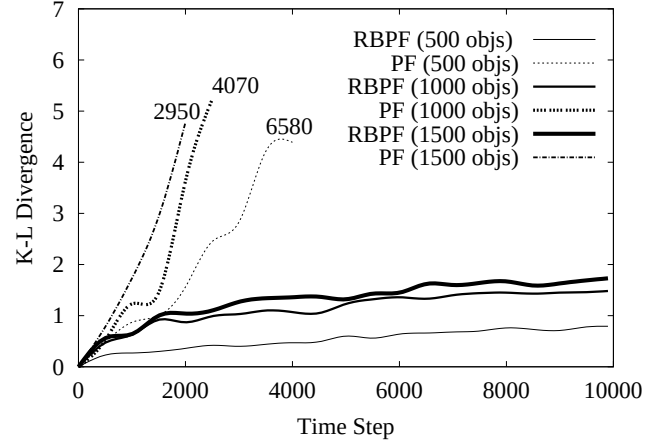


Figure 2: Comparison of RBPF (5000 particles) and PF (200,000 particles) for fault probability of 1% and varying number of objects.

K-L divergence of RBPF increases only very slowly, for all combinations of parameters tried. Abstraction trees reduced RBPF's time and memory by a factor of 30 to 70, and took on average six times longer and 11 times the memory of PF, per particle. However, note that we ran PF with 40 times more particles than RBPF. Thus, RBPF is using less time and memory than PF, and performing far better in accuracy.

We also ran all the experiments while measuring the K-L divergence of the full joint distribution of the state (as opposed to just the marginals). RBPF performed even better compared to PF in this case; the latter tends to blow up much sooner (e.g., from around step 4000 to less than 1000 for $p_f = 1\%$ and 1000 objects), while RBPF continues to degrade only very slowly.

6 Related Work

Dynamic object-oriented Bayesian networks (DOOBNS) [Friedman *et al.*, 1998] combine DBNs with OOBNS, a predecessor of PRMs. Unfortunately, no efficient inference

methods were proposed for DOOBs, and they have not been evaluated experimentally. DPRMs can also be viewed as extending relational Markov models (RMMs) [Anderson *et al.*, 2002] and logical hidden Markov models (LOHMMs) [Kersting *et al.*, 2003] in the same way that DBNs extend HMMs. Downstream, DPRMs should be relevant to research on relational Markov decision processes (e.g., [Boutilier *et al.*, 2001]).

Particle filtering is currently a very active area of research [Doucet *et al.*, 2001]. In particular, the FastSLAM algorithm uses a tree structure to speed up RBPf with Gaussian variables [Montemerlo *et al.*, 2002]. Abstraction trees are also related to the abstraction hierarchies in RMMs [Anderson *et al.*, 2002] and to AD-trees [Moore and Lee, 1997]. An alternate method for efficient inference in DBNs that may also be useful in DPRMs was proposed by Boyen and Koller [1998] and combined with particle filtering by Ng *et al.* [2002]. Efficient inference in relational probabilistic models has been studied by Pasula and Russell [2001].

7 Conclusions and Future Work

This paper introduces dynamic probabilistic relational models (DPRMs), a representation that handles time-changing phenomena, relational structure and uncertainty in a principled manner. We develop efficient approximate inference methods for DPRMs, based on Rao-Blackwellisation of relational attributes and abstraction trees. The power of DPRMs and the scalability of these inference methods are illustrated by their application to monitoring assembly processes for fault detection.

Directions for future work include relaxing the assumptions made, further scaling up inference, formally studying the properties of abstraction trees, handling continuous variables, learning DPRMs, using them as a basis for relational MDPs, and applying them to increasingly complex real-world problems.

8 Acknowledgements

This work was partly supported by an NSF CAREER Award to the second author, by ONR grant N00014-02-1-0932, and by NASA grant NAG 2-1538. We are grateful to Mausam for helpful discussions.

References

- [Anderson *et al.*, 2002] C. Anderson, P. Domingos, and D. Weld. Relational Markov models and their application to adaptive Web navigation. In *Proc. 8th ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining*, pages 143–152, Edmonton, Canada, 2002.
- [Boutilier *et al.*, 1996] C. Boutilier, N. Friedman, M. Goldszmidt, and D. Koller. Context-specific independence in Bayesian networks. In *Proc. 12th Conf. on Uncertainty in Artificial Intelligence*, pages 115–123, Portland, OR, 1996.
- [Boutilier *et al.*, 2001] C. Boutilier, R. Reiter, and B. Price. Symbolic dynamic programming for first-order MDPs. In *Proc. 17th Intl. Joint Conf. on Artificial Intelligence*, pages 690–697, Seattle, WA, 2001.
- [Boyen and Koller, 1998] X. Boyen and D. Koller. Tractable inference for complex stochastic processes. In *Proc. 14th Conf. on Uncertainty in Artificial Intelligence*, pages 33–42, Madison, WI, 1998.
- [Cover and Thomas, 2001] T. Cover and J. Thomas. *Elements of Information Theory*. Wiley, New York, 2001.
- [Dean and Kanazawa, 1989] T. Dean and K. Kanazawa. A model for reasoning about persistence and causation. *Computational Intelligence*, 5(3):142–150, 1989.
- [Doucet *et al.*, 2001] A. Doucet, N. de Freitas, and N. Gordon, editors. *Sequential Monte Carlo Methods in Practice*. Springer, New York, 2001.
- [Friedman *et al.*, 1998] N. Friedman, D. Koller, and A. Pfeffer. Structured representation of complex stochastic systems. In *Proc. 15th National Conf. on Artificial Intelligence*, pages 157–164, Madison, WI, 1998.
- [Friedman *et al.*, 1999] N. Friedman, L. Getoor, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In *Proc. 16th Intl. Joint Conf. on Artificial Intelligence*, pages 1300–1309, Stockholm, Sweden, 1999.
- [Getoor *et al.*, 2001] L. Getoor, N. Friedman, D. Koller, and B. Taskar. Learning probabilistic models of relational structure. In *Proc. 18th Intl. Conf. on Machine Learning*, pages 170–177, Williamstown, MA, 2001.
- [Hoffmann and Nebel, 2001] J. Hoffmann and B. Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.
- [Kersting *et al.*, 2003] K. Kersting, T. Raiko, S. Kramer, and L. De Raedt. Towards discovering structural signatures of protein folds based on logical hidden Markov models. In *Proc. 8th Pacific Symposium on Biocomputing*, Kauai, HI, 2003.
- [Montemerlo *et al.*, 2002] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM: A factored solution to the simultaneous localization and mapping problem. In *Proc. 18th National Conf. on Artificial Intelligence*, pages 593–598, Edmonton, Canada, 2002.
- [Moore and Lee, 1997] A. W. Moore and M. S. Lee. Cached sufficient statistics for efficient machine learning with large datasets. *Journal of Artificial Intelligence Research*, 8:67–91, 1997.
- [Murphy and Russell, 2001] K. Murphy and S. Russell. Rao-Blackwellised particle filtering for dynamic Bayesian networks. In A. Doucet, N. de Freitas, and N. Gordon, editors, *Sequential Monte Carlo Methods in Practice*, pages 499–516. Springer, New York, 2001.
- [Ng *et al.*, 2002] B. Ng, L. Peshkin, and A. Pfeffer. Factored particles for scalable monitoring. In *Proc. 18th Conf. on Uncertainty in Artificial Intelligence*, Edmonton, Canada, 2002. Morgan Kaufmann.
- [Pasula and Russell, 2001] H. Pasula and S. J. Russell. Approximate inference for first-order probabilistic languages. In *Proc. 17th Intl. Joint Conf. on Artificial Intelligence*, pages 741–748, Seattle, WA, 2001.

