ICAPS 2006

The English Lake District, Cumbria, UK

WS3

Workshop on Planning under Uncertainty and Execution Control for Autonomous Systems

Christophe Guettier

SAFRAN SA, France














WS3

ICAPS 2006

The English Lake District, Cumbria, UK

Workshop on Planning under Uncertainty and Execution Control for Autonomous Systems

Christophe Guettier

SAFRAN SA, France

ICAPS 2006
Workshop on Planning under Uncertainty and Execution Control for Autonomous Systems

Table of contents

ICAPS 2006
Workshop on Planning under Uncertainty and Execution Control for Autonomous Systems

Preface

The variety of autonomous systems is increasing both in industry and academia (AUV, UGV, UAV, Robots, Space Probes), which addresses a broad range of missions and duty (space exploration, search and rescue, defense and security). Such systems must operate with limited human intervention in a changing environment and behave efficiently in spite of mission updates, failure recovery and limited on-board resources. Recent experiences show that simple planning and execution control techniques may endanger the system when uncertain information or limited knowledge about the environment is available. In order to achieve a consistent or optimized behavior, planning and execution control problems, addressed separately in the past, have to be solved in a combined and coordinated way. The environment being partially known by an autonomous system, plan generation and execution must consider uncertain information, biased data, unpredicted or contingent events. Moreover, planning and execution control have to satisfy different time scales, depending on mission, system and environment tempos. Lastly, uncertainty may result from approximations at the planning level or unexpected events may occur due to a lack of control. Various theoretical and practical aspects of planning and execution control have been studied so far (Constraint solving, combinatorial optimization, Markov decision processes, controllable networks, Petri nets...). However, a broad range of hybrid algorithms, architecture designs and alternative approaches are explored during the workshop.

We would like to thank members of the Program Commitee who have provided excellent and very constructive reviews. We are also grateful to the ICAPS conference and workshop chairs for their help in organizing this workshop.

Organizers

- *Christophe Guettier (*Sagem Defense and Security, France*)*

Programme Committee

- *Mark Wallace (*Monash University - Australia*)*
- *Alessandro Saffiotti (*Orebro University - Sweden*)*
- *Russel Knight (*Jet Propulsion Lab - USA*)*
- *Nicolas Meuleau (*NASA Ames Research Center - USA*)*
- *Abdel-Illah Mouaddib (*Universite de Caen - France*)*
- *David Ferguson (*Canergie Mellon University - USA*)*
- *Felix Ingrand (*LAAS/CNRS, France*)*
- *Patrick Fabiani (*ONERA, France*)*
- *Mark Boddy (*Adventium Labs - USA*)*
- *Roy Turner (*University of Maine, USA*)*
- *Grard Verfaillie (*ONERA, France*)*
- *Wheeler Ruml (*PARC, USA*)*
- *Neil Yorke-Smith (*SRI, USA*)*
- *Christophe Guettier (*SAFRAN, France*)*

View Planning Algorithms for a Multi-Camera Surveillance System

Jun Miura Noriko Takemura Takuro Sakiyama
Department of Mechanical Engineering, Osaka University
Suita, Osaka 565-0871, Japan
{jun,takemura,saki}@cv.mech.eng.osaka-u.ac.jp

Abstract

This paper deals with a view planning of multiple active cameras for tracking multiple persons for surveillance purposes. We develop algorithms for dynamically planning viewing directions of cameras so that the expected number of tracked persons is maximized, based on a probabilistic model of person motion. Since a naive approach to this planning easily causes a combinatorial explosion, we adopt a meta-heuristic algorithm, namely, multi-start local search (MLS). We first develop an MLS-based algorithm that exhibits a comparable performance to an exhaustive search-based one but with a considerably low planning cost. We then modify the problem so that intermittent observations of a person are allowed for estimating the person's motion continuously. In this modified problem, cameras are encouraged to frequently change fixation points so that they can track as various persons as possible. For this problem, we develop another MLS-based planning method which searches the space of sequences of fixation points and uses an effective initial solution generation. Simulation results show the effectiveness of this planning method.

Introduction

Visual surveillance is one of the active research areas in computer vision. Most previous works are concerned with development of image processing algorithms for detecting persons or vehicles reliably and/or for analyzing their activities (Lee, Romano, & Stein 2000; Stauffer & Grimson 2000; Buxton 2003). This paper focuses another important problem in surveillance, namely, view planning of cameras.

One way to cover a wide area for surveillance is to use many fixed cameras whose fields of view collectively cover the area. This is, however, costly and sometimes difficult due to installation problems. We therefore take an approach of using a small number of active cameras; by appropriately controlling the fixation points of cameras, the whole area, although it cannot be covered at a time, will be covered within a certain period of time. A key to effective surveillance in this approach is view planning of cameras.

Ukita and Matsuyama (2003) developed a method of tracking multiple target by multiple active cameras. Multiple vision agents, each of which is responsible for controlling one camera, dynamically form several agencies (set of agents) according to the number of targets and their situations. Karuppiah et al. (2005) proposed a method of dynamically configuring multiple cameras so that a target can be tracked reliably, using a utility function evaluating the measurement accuracy and the predictability of possible events. These works deal with tracking of a few persons in a relatively small area.

Horling et al. (2001) dealt with a cooperative vehicle monitoring by a distributed sensor network. They formulate the problem as a resource allocation problem in which what area to be sensed by each sensor and what information should be communicated are determined with consideration of sensor and communication uncertainties. Isler et al. (2005) developed algorithms for assigning targets to multiple cameras so that the expected error in the target location estimation is minimized. These works treated the case where the number of cameras is relatively larger than that of targets.

Jung and Sukhatme (2004) dealt with a coordination of multiple mobile robots to track multiple targets. They calculate the urgency over the field and use it to distribute the robots. The evaluation of urgency is based on the current distribution of targets not on a prediction of future states.

Miura and Shirai (2002) dealt with a multi-camera multi-person tracking problem in the context of parallelization of planning and action. They used a heuristic planning algorithm which iteratively refines the assignment of persons to cameras, formulated as an anytime algorithm (Dean & Boddy 1988).

Krishna, Hexmoor, and Sogani (2005) developed a view planning algorithm for a multi-sensor surveillance system. To avoid a combinatorial explosion, they dynamically prioritize the sensors based on their predicted coverage of targets. Coverage prediction is performed using statistical knowledge of the target distribution; however, they do not predict the motion of each person independently.

This paper deals with a view planning of multiple active cameras for tracking many persons. We first define the multi-camera multi-person tracking problem (called *MCMP problem*). We then describe a model of person motion and a method of predicting the positional distributions of persons to be used for estimating the expected number of tracked persons. Concerning planning algorithms, we first compare

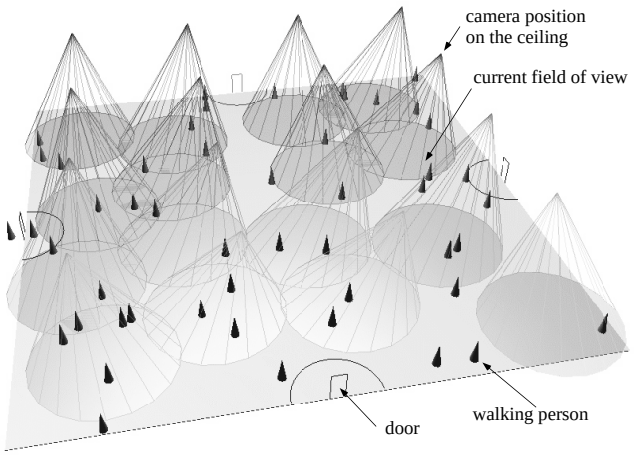


Figure 1: MCMP simulator.

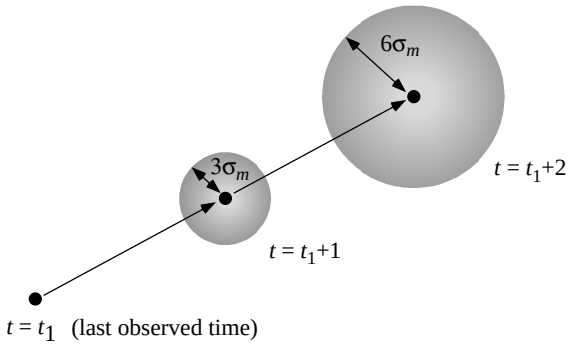


Figure 2: Motion uncertainty model of person.

two approaches, an exhaustive search-based one and a multi-start local search-based one, and show that the latter exhibits a comparable performance to the former with a considerably lower calculation cost. We then introduce a modified MCMP problem that allows a tracking with intermittent observations, and present an MLS-based planning method with an effective initial solution generation. We experimentally show that this method is better than other ones. We finally summarize the paper and discuss future works.

Multi-Camera Multi-Person Tracking Problem

This paper deals with the following MCMP problem. There are N_p persons arbitrarily waking in a room. There are N_c ($\ll N_p$) cameras fixed on the ceiling of the room so that no occlusions between persons occur. Each camera can change the viewing direction within a predetermined range. A single planning process controls the viewing directions of all cameras. The goal of the whole system is to track as many persons as possible during a certain period of time. Each camera is assumed to be able to recognize any person and measure his/her position/velocity, as long as the person is inside the field of view of the camera.

We made a simulator for the MCMP problem, as shown in Fig. 1. In addition to the general problem description mentioned above, we use the following detailed settings. The room is a $50[m] \times 50[m]$ square and four cameras are placed ($N_c = 4$) in a 2×2 array on the ceiling of $10[m]$ high. The field of view (FOV) of each camera is assumed to be always a circle of $10[m]$ radius; view planning of a camera is thus equivalent to selecting its fixation point (the center of FOV) on the floor. Each camera can move the fixation point within the circle of $10[m]$ radius centered at the home position right below the camera. The maximum speed of moving the fixation point is $2.5[m/s]$. The whole 2D space is discretized as a grid with $0.5[m]$ regular spacing and fixation points of cameras are limited to grid points; fixation point candidates thus form a 100×100 grid. The cameras observe and change fixation points at the cycle of $1[s]$.

The number of persons is 30 ($N_p = 30$) in the four camera case. Each person basically performs a linear and constant motion but the velocity and the moving direction change every step according to the normal distribution with the variances $1.5[m^2/s^2]$ and $25[deg^2]$, respectively. When a person touches a wall, he/she changes the velocity in a regular reflection manner.

We additionally use another setting in which the room is a $100[m] \times 100[m]$ square with 120 persons ($N_p = 120$) and sixteen cameras are placed in a 4×4 array ($N_c = 16$).

Prediction of Future States

Planning algorithms repeatedly determine the fixation points of all cameras at the *next* time step ($t = 1$) based on the prediction of states of tracked persons for future T time steps ($t = 1 \sim T$).

Motion Modeling of Person

We use a linear motion model for predicting positions of persons. Concerning the uncertainty in prediction, we use a simple probabilistic model that the positional uncertainty of a person is isotropic and represented by the so-called 3σ portion of the normal distribution with variance $\sigma_m^2 t$, where t is the time step from the last time at which the person is observed (see Fig. 2). σ_m^2 is determined so that the predicted uncertainty covers the actual uncertainty. We assume that the position of a person can be predicted if the period of not observing the person is less than three steps; otherwise, that person's positional uncertainty is too large to be used for planning.

Predicting the Number of Tracked Persons for a Fixation Point

The objective of planning is to repeatedly determine fixation points that can maximize the expectation of the number of tracked persons for a predetermined time duration. From the motion uncertainty model of person, we can calculate a set of positional distributions of the persons currently under consideration at a future time step. On the other hand, for each fixation point of a camera, its field of view (FOV) is calculated. The expected number of persons tracked by the

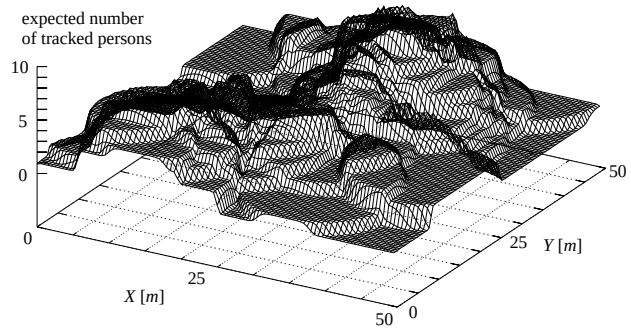


Figure 3: An example map of the expected number of tracked persons.

camera directed to a specific fixation point at a time step is thus calculated as the summation of the probabilities of the persons being within the corresponding FOV. Since the fixation points are on grid points in the room and all cameras have the same characteristics, we make a 2D grid map of the expected number of tracked persons and use it for every camera. This map is generated for each future time step to be considered. Fig. 3 shows an example map for $N_p = 30$.

The probability that a person is within an FOV is calculated by integrating the person's positional distribution within the FOV. Since the FOVs and the distributions are both circular, we can prepare a look-up table indexed by the variance of the distribution (which is equivalently the number of steps during which a person is not in any FOVs) and the distance between the mean position and the fixation point.

When FOVs of two or more cameras overlap with each other, the calculation of the expected number becomes a little more complex. The probability that a person is within any of FOVs is calculated as follows:

- If the positional distribution of the person is completely within the FOV of at least one camera, the probability is one.
- If the distribution of the person is completely out of all FOVs, the probability is zero.
- If only a part of the distribution is within some FOVs, we classify this case into the following three subcases:
  - If that part is included only in one FOV, the probability is calculated by the table look-up.
  - If that part is included in multiple FOVs but not in any intersection of the FOVs, the probability is the sum of the probabilities of being included these FOVs (i.e., the sum of the results of the table look-up).
  - If that part is included in the intersection of some of the FOVs, we need to integrate the probabilities inside the union of such FOVs; but this calculation is costly because the simple table look-up cannot be used.

Although the last subcase should be, in principle, treated differently from the others, we approximate the probability for the subcase with the one calculated in the same way as the other cases because we examined many data and found that the frequency that this subcase happens is about 1%.

Exhaustive Search vs. Multi-Start Local Search

This section compares two planning methods based on an exhaustive search with pruning and a multi-start local search (MLS). The performance of the former will be a benchmark for evaluating the latter.

Criteria for Evaluating Fixation Points

The primary criterion for selecting fixation points is the expected number of tracked persons for a certain period of time. Since several fixation points may have the same expected number of tracked persons, we use two more criteria for evaluation.

- The amount of movements of camera. Smaller values are better. This is for evaluating the smoothness of camera movements.
- The distance of the fixation point of a camera from its home position. Smaller values are better. This is for evaluating the distribution of camera fixation points. If persons are distributed widely in the room, then this criterion will be more important. In addition, more highly distributed fixation points are better for (fortunately) capturing currently-untracked persons.

These criteria used in the following order: the expected number of tracked persons, the amount of camera movements, and the distance from the home position. If two or more solutions are equivalent in terms of a preceding criterion, the next one is used for ordering the solutions. Ties under all criteria are broken randomly.

Exhaustive Search with Pruning

A planning method based on exhaustive search is used for obtaining optimal solutions. There are two parameters for controlling the search. T is the depth of look-ahead and V is the number of fixation point candidates to be kept at a depth. The order of the computation is thus $\mathcal{O}(V^T)$. The maximum value of V is given by C^{N_c}, where C is the number of all possible fixation point candidates for a camera and N_c is that of cameras. When this maximum value is used, the search is completely exhaustive. Due to a high computation time, we only tested the following two combinations of parameters: $(T, V) = (1, C^{N_c}), (2, 20)$.

We adopt two techniques for speeding up the planning. One is the pruning using an upper bound of the expected number of tracked persons, which is the one calculated under the assumption that any FOV does not overlap with the others. In examining a combination of fixation points, every time the fixation point of a camera is chosen, the upper bound of the combination is updated (using the upper bounds for the unchosen cameras) and compared with the current-best solution obtained so far. If the current combination is found to be unpromising, the computation for the combination is terminated. The upper bound for each camera is easily calculated by referring to the map described above.

Table 1: Comparison of exhaustive search- and sequential MLS-based methods.

method	look-ahead	tracking ratio (%)	std. dev. (%)	calculation time per step (sec.)	std. dev. (sec.)
exhaustive	$T = 1$	66.6	3.76	0.23	0.28
exhaustive	$T = 2$	67.5	2.47	143.62	94.0
sequential MLS	$T = 1$	65.3	3.78	0.019	0.0029
sequential MLS	$T = 2$	65.8	3.67	0.028	0.0039
sequential MLS	$T = 5$	67.3	3.53	0.073	0.0044

Another technique is to decompose the problem into a set of independent subproblems. A group of cameras can be planned independently with the other cameras as long as the FOVs of the cameras in the group do not overlap with those of the other cameras for the period of time under consideration. So we first segment cameras into such independent groups, then make a subplan for each group, and finally merge the subplans into the plan of all cameras.

Multi-Start Local Search

Multi-start local search (MLS) is a commonly-used algorithm for solving large-scale combinatorial problems (Yagiura & Ibaraki 2001). In MLS, local search (LS) is repeated from a number of initial solutions and the best solution found during the entire search is output.

In our tracking problem, the expected number of persons roughly continuously changes over the entire space and the number of local minima is expected to be relatively small (see Fig. 3). MLS is thus suitable for our problem.

Search Space, Neighborhood and Initial Solution We consider the space of all combinations of fixation points of the cameras. The search space at each time step is a subspace determined by the movable ranges of the cameras by that time step. We define the neighborhood of a solution (a point in the space) as the set of solutions in which the fixation point of only one camera is different from the solution by one step in the grid representation of 2D position (so-called 8 neighbors). The number of neighboring solutions is thus $8N_c$. We randomly generate N_{init} initial solutions within the search space.

Sequential MLS The algorithm for MLS-based planning (called *sequential MLS*) is as follows.

1. Choose N_{init} initial solution (i.e., set of fixation points for all cameras) for the next step ($t = 1$) randomly.
2. For each initial solution, repeat the following for $t = 1 \sim T$:
   (a) Perform local search for the locally-best solution. We use the *best admissible move strategy*, in which the best solution in the neighbor of the current solution is chosen as the next one.
   (b) Randomly generate one initial solution for the next time step from the locally-best solution, if $t < T$.
3. Select the first step of the best among N_{init} solutions, which maximizes the expected number of tracked persons for the duration $[1, T]$), as the movement for the next step.

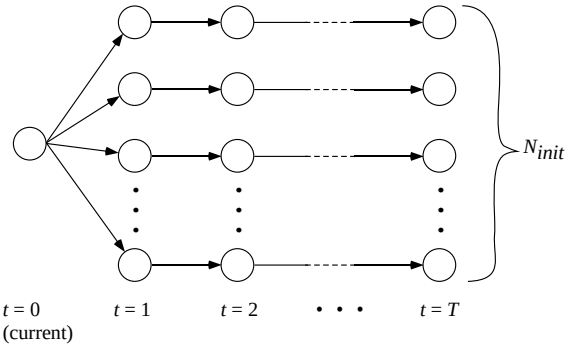


Figure 4: Search tree of *sequential MLS*. Each link indicates a pair of initial solution generation and local search.

Fig. 4 shows the search tree for *sequential MLS*. We currently use $N_{init} = 15$, which is empirically determined.

Results

We made 10 sets of simulation data, each of which is composed of 100 step movements of 30 persons. Using these sets, we compared the exhaustive search-based method with $T = 1, 2$ and the sequential MLS-based method with $T = 1, 2, 5$ for the four-camera setting. We evaluate the methods in terms of *tracking ratio*, which is the averaged number of tracked persons per time step divided by the total number of persons. Since the sequential MLS is a randomized method, for each data set, we ran the method 10 times and calculated the average of the resulting tracking ratios.

Table 1 summarizes the comparison results. The table shows the average tracking ratio of all the data sets. The computation time for the exhaustive search-based method becomes very large even for $T = 2$ to be used in practical systems. In addition, the variance of computation time is larger. Concerning the sequential MLS, as the look-ahead becomes longer, the performance increases while the computation time increases only approximately linearly. The sequential MLS with $T = 5$ exhibits a comparable performance to the exhaustive with $T = 2$, and spends a very short computation time, which is short enough to be used for on-line planning.

These results show that MLS-based methods are suitable for the MCMP problem.

Tracking with Frequently Changing Fixation Points

When we visually track many arbitrarily walking persons, we do not continuously track the same group of persons but usually take a strategy of changing the fixation point frequently from person to person at various positions. Even if we do not look at a person for a short period of time, we can estimate (or interpolate) his/her movement from the intermittent observation data[1]. This strategy can thus increase the number of tracked persons while keeping a sufficient accuracy in motion estimation. This section applies this strategy to the MCMP problem using MLS.

Evaluation Criterion for Tracking with Intermittent Observations

We assume that a low-level tracking system is working beneath the view planner. Such a system is often developed based on statistical data integration methods such as Kalman filter (Koyasu, Miura, & Shirai 2001) or particle filters (Maskell *et al.* 2003). These methods use a probabilistic model of state evolution. Such a model usually indicates that the positional uncertainty of a target increases as time elapses if no observations are available, and that the target will eventually be lost if it is not observed for a long time.

This implies that as long as the time period during which a target is not observed is *sufficiently* short, the target's movement can reliably be estimated. In this paper, for simplicity, we set a threshold and if the non-observation time period for a target is less than or equal to the threshold, the target is considered being tracked even for that time period. Currently, we use two as the threshold. That is, when a person is observed at time t_1 and t_2 ($t_1 < t_2$) and not observed at times $\{t \,|\, t_1 < t < t_2\}$, the total number of tracking for the person given at time t_2 is $t_2 - t_1$ if $t_2 \leq t_1 + 3$ and one otherwise. We use the evaluation criterion based on this calculation of the number of tracked persons.

This change of evaluation criterion will alter the behavior of cameras. Fig.5 shows an illustrative example. There are two groups of persons on the upper and the lower side of the space, respectively, and the camera cannot capture both groups at times $t = 1, 2$. When we maximize the number of persons within the FOV of the camera (by the previous evaluation criterion), the camera moves like Fig. 5(a) and the total number of the tracked person is eleven. On the other hand, if we use the new evaluation criterion, the camera will move like Fig. 5(b) and the total number of the tracked persons now becomes twelve; the camera tends to move to the persons that have been out of FOVs for a while.

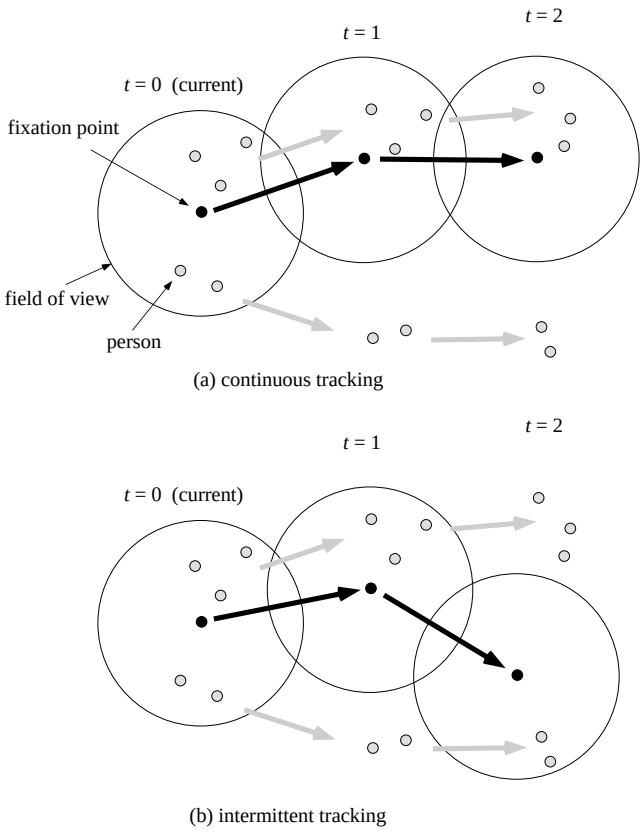


Figure 5: Different behaviors for different evaluation criteria.

Search Space and Neighborhood

In the previous MLS-based method (sequential MLS), the search space is composed of all combination of fixation points of the cameras *at one time step* and a set of fixation points is sequentially determined from the next step to the final step. In the new method, however, fixation positions cannot be evaluated at one time step but should be evaluated as a sequence of them. We therefore define the search space as all combinations of fixation points of the cameras during the whole time period under consideration.

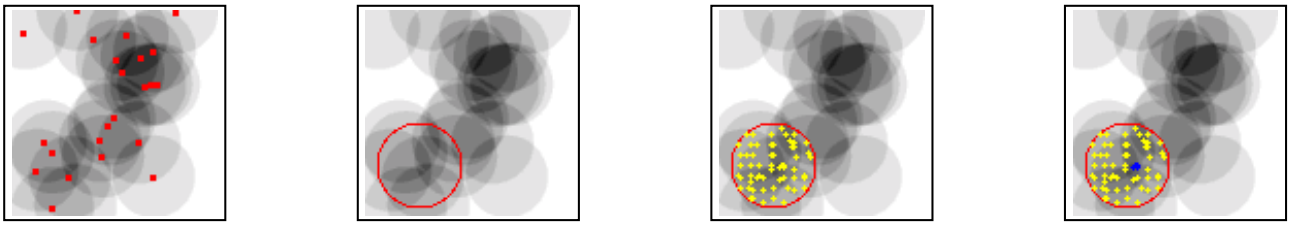
We define the neighborhood of a solution as the set of solutions in which the fixation point of only one camera *at only one time* is different from the solution by one step in the grid representation of 2D position (again, 8 neighbors). Letting T be the depth of look-ahead, the number of neighboring solutions is thus $8N_cT$. We randomly generate N_{init} initial solutions within the search space.

We also use the best admissible move strategy in this method.

Generating Initial Solutions

The new MLS-based algorithm searches the space of a sequence of sets of fixation points up to the depth limit. This means that the search space is considerably larger than that of the previous MLS-based method (sequential MLS) which determines a set of fixation points for one time step to another. A larger search space usually requires more initial solutions to get satisfactory results in MLS, thus increasing the computation time. It is possible to use the result of sequential MLS as an initial solution. Sequential MLS, however,

[1]Note that not observations themselves but those for a person are intermittent; that is, cameras obtain observations at every time step but targets of observation may be different from time to time.



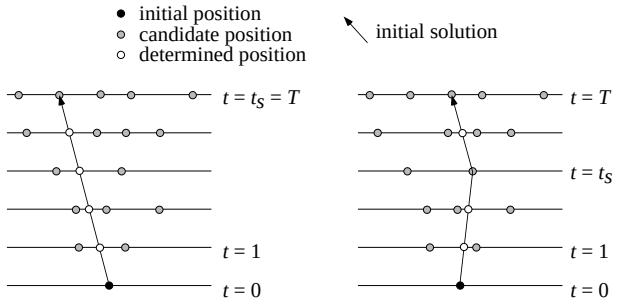
(a) persons (red squares) and the map of expected number of tracked.

(b) a range of fixation points at a time step.

(c) representative points for uniformly-divided regions.

(d) selected promising candidate points (indicated by blue).

Figure 6: Generating a map of promising fixation points at a time step.



(a) candidate position at time T is selected.

(b) candidate position at time $t_S < T$ is selected.

Figure 7: Generate initial solutions.

tries to generate solutions in which the same group of persons tends to be tracked *continuously* with relatively smooth camera movements (as shown in Fig. 5(a)), and may not be appropriate for generating solutions with frequent changes of fixation points (as shown in Fig. 5(b)). We thus take another approach in which promising fixation points in *space-time* are explicitly enumerated and used for generating initial solutions.

The steps for generating initial solutions are as follows. These steps are performed for each camera independently (i.e., we do not consider the overlap of FOVs at this stage).

1. Generate maps of the expected number of tracked persons, as described above, for the time steps under consideration ($t = 1 \sim T$) (see Fig. 6(a) for the map for a time step).
2. Divide the maps into a set of uniform-sized regions (composed of 5×5 grid points) within the movable range of each camera (see Fig. 6(b)) and select one representative point within each region which has the maximum expected number (see Fig.6(c)). The expected number becomes the score of the region.
3. Determine the maximum score and set a threshold for *promising* fixation points as the $\alpha\%$ of the maximum (currently, $\alpha = 90$). The representative points of the regions whose scores are higher than the threshold become a set of fixation point candidates (see Fig.6(d)).
4. Repeat the following for each camera to select N_{init} initial solutions:

   Select one fixation point among the candidates randomly. Let t_s be the time step at which the fixation point is. If $t_s = T$ then the fixation points at $t = 1 \sim T - 1$ are determined by the interpolation. Fig. 7(a) shows such a case. The horizontal lines in the figure represent a side view of 2D maps. If $t_s < T$, then the fixation points at $t = 1 \sim t_s - 1$ are determined by the interpolation, and those at $t > t_s$ are determined recursively (select one candidate point at $t > t_s$ randomly and so on) (see Fig. 7(b)).
5. Merge N_{init} sets of initial solutions for all cameras.

Planning Algorithm

The new planning algorithm performs MLS using the initial solutions mentioned above. We examined the performance of planning for several N_{init}'s and decided to use $N_{init} = 15$. Once the set of fixation point candidates is generated (Steps 1 to 3 in the above), the rest of the initial solution generation and the local search are completely parallelizable. We thus use a PC cluster system with 15 CPU's to speed up the planning. The average computation time for one time step is about $0.8\,[sec]$.

Experimental Results

This section describes experimental results using the same data sets as the one used in the previous comparison (i.e., 10 sets of simulation data, each of which is composed of 100 step movements of 30 persons).

Comparison of Methods for Generating Initial Solutions

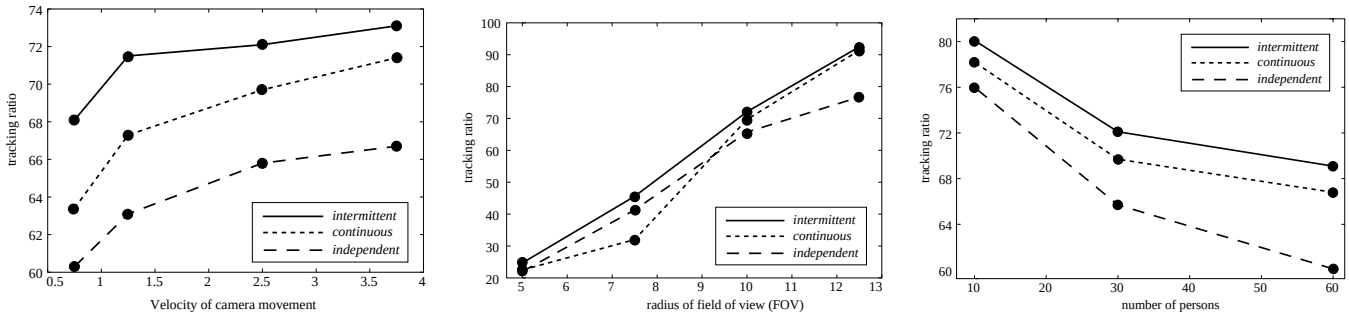
We compare the following three methods:

- Explicitly enumerates promising fixation points for generating initial solutions (proposed method).
- Use the results of the sequential MLS method as initial solutions.
- Randomly generate initial solutions.

Table 2 summarizes the comparison result. The proposed method outperforms the others.

Table 3: Comparison of five methods.

	Data ID										
	1	2	3	4	5	6	7	8	9	10	Average
intermittent	73.2	74.1	71.9	71.3	64.7	74.3	67.2	73.8	73.0	77.7	72.1
continuous	72.9	72.8	70.3	67.1	65.0	73.7	66.1	64.8	68.3	75.8	69.7
independent	67.7	69.6	65.2	66.5	56.8	67.5	60.7	63.1	64.0	60.2	64.1
random	46.1	52.0	49.9	47.4	48.5	47.8	48.1	48.2	47.0	46.8	48.2
no planning	45.6	53.2	50.2	41.3	41.9	51.5	47.2	46.8	44.7	46.1	46.9



(a) maximum velocity of camera movement vs. tracking ratio.

(b) radius of field of view of cameras vs. tracking ratio.

(c) maximum number of persons to be tracked vs. tracking ratio.

Figure 8: Comparison in various problem settings.

Table 2: Comparison of methods of generating initial solutions in terms of tracking ratio.

Method	Proposed	MLS solution	Random
Tracking ratio	72.1%	70.1%	69.1%

Comparison with Other Methods

We here compare the following five methods:

- New MLS-based method for tracking with intermittent observations (called *intermittent*).
- Sequential MLS method (called *continuous*).
- Select fixation point of each camera independently for tracking with intermittent observations (called *independent*).
- Select fixation points randomly at every time step (called *random*).
- Fixed cameras (called *no planning*).

Table 3 compiles the results. Note that the new evaluation criterion that allows tracking with intermittent observations is used for evaluating all methods. The table shows that *random* and *no planning* produce much worse results. Among the other three, the *intermittent* method exhibits the best performance.

We have also compared *intermittent* and *continuous* for another problem setting in which the room is a $100[m] \times 100[m]$ square with 120 persons ($N_p = 120$) and sixteen cameras are placed in a 4×4 array ($N_c = 16$). The averaged tracking ratios of *intermittent* and *continuous* for 3 data sets are 72.7% and 68.5%, respectively. The proposed *intermittent* method again has exhibited the best performance.

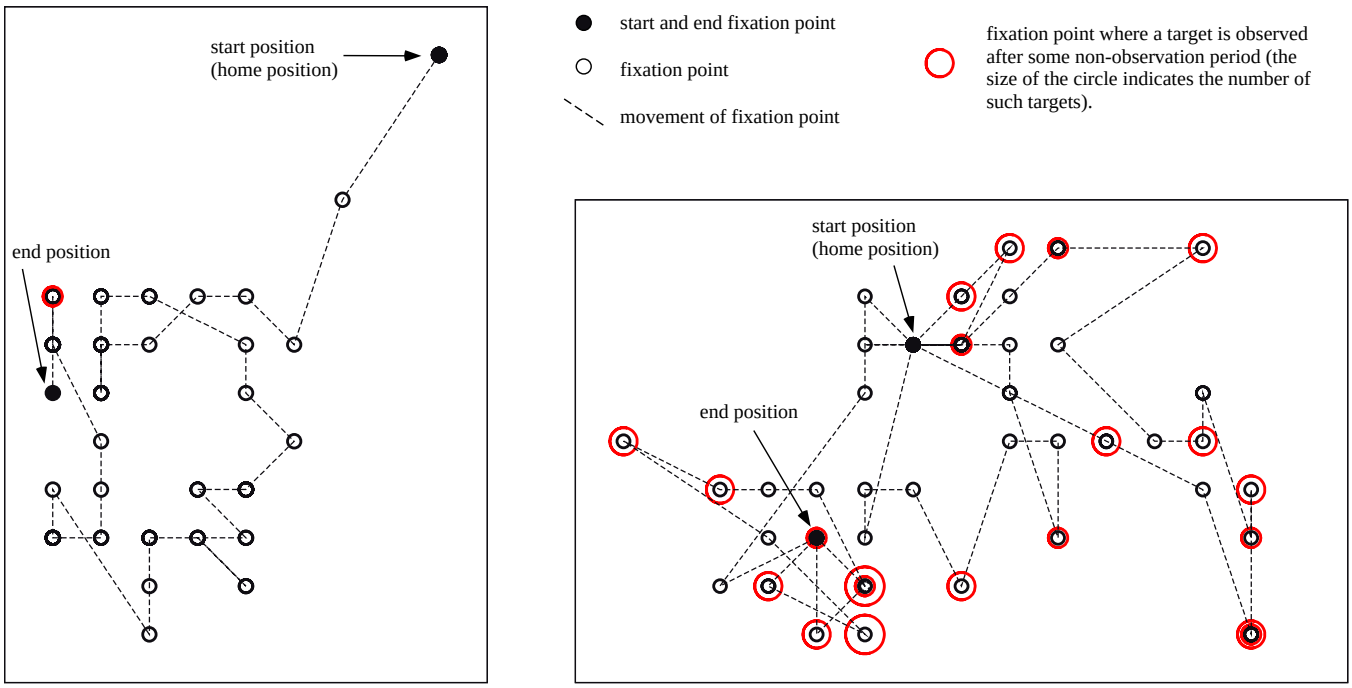
Comparison in Various Problem Settings

We then compare the three methods (*intermittent*, *continuous*, *independent*) in various problem settings. In general, the difference in performance between planning methods is smaller in easier problems. As the problem becomes harder, however, only *good* methods are expected to exhibit a satisfactory performance. We therefore change several parameters determining the *hardness* of the problem to examine if there exists such a tendency.

Fig. 8(a)-(c) show the comparison results for changing the maximum velocity of the camera fixation point, the radius of the field of view, and the number of persons, respectively. In all cases, the *intermittent* method outperforms the others and its performance degradation according to the problem being harder is smaller. These results show the effectiveness of the proposed *intermittent* method.

Comparison of Camera Behaviors for Continuous and Intermittent

Fig. 9 shows a comparison of the movements of a camera for a set of data. Fig. 9(a) shows the movement generated by the *continuous* planning method, while Fig. 9(b) shows the one generated by the *intermittent* planning method. Red circles indicate the fixation points where one or more targets



(a) *continuous* tracking

(b) *intermittent* tracking

Figure 9: Camera movements for *continuous* and *intermittent*. The start positions are the same actually.

are observed after some non-observation period. The figure shows that by *intermittent* planning method, the camera changes fixation points more frequently and widely in order to capture as various persons as possible. The total numbers of tracked persons for the whole time period (i.e., 100 time steps) for *continuous* and *intermittent* planning are 432 and 506, respectively. This also indicates the effectiveness of the proposed *intermittent* planning method.

Conclusions and Discussion

This paper has presented methods of view planning for multi-camera surveillance applications. We have defined a multi-camera multi-person tracking problem (MCMP problem), in which the objective of planning is to maximize the number of tracked persons. We first compared an exhaustive search-based method and a multi-start local search (MLS)-based method and have shown that the latter method exhibits a comparable performance to the former with much less computation time. We then introduced a new evaluation criterion that allows tracking with intermittent observations thus encouraging frequent changes of fixation points. For this criterion, we have developed another MLS-based method that searches the space of combinations of fixation points of all cameras during the look-ahead. We also developed a method of generating initial solutions from a set of promising fixation points in space-time. This MLS-based method outperforms other methods, especially when the problem is *hard*.

Currently, we make several assumptions: no occlusion, negligible target recognition time, perfect recognition ability. A future work is to remove these assumptions in order to consider more realistic situations such as occasional occlusion and recognition failure. Especially, when we remove the assumption of perfect recognition ability, we need to model the performance of recognition, which will decrease as the time for not observing a target increases. We then need to consider the tradeoff between increasing recognition performance by observing each target frequently and increasing the number of tracked persons by frequently changing fixation points.

Another future work is to apply the current method to the cases where the above assumptions almost hold. An example case is the one where cameras are set at high positions and persons with distinctive colors walk in a simple background. The proposed method can also be applied to the case where we analyze very large images from stationary cameras and need to select a portion of the images to analyze at each frame due to computation limitation.

References

Buxton, H. 2003. Learning and understanding dynamic scene activity: A review. *Image and Vision Computing* 21(1):125–136.

Dean, T., and Boddy, M. 1988. An analysis of time-dependent planning. In *Proceedings of AAAI-88*, 49–54.

Horling, B.; Vincent, R.; Mailler, R.; Shen, J.; Becker, R.; Rawlins, K.; and Lesser, V. 2001. Distributed sensor network for real time tracking. In *Proceedings of the 5th Int. Conf. on Autonomous Agents*, 417–424.

Isler, V.; Khanna, S.; Spletzer, J.; and Taylor, C. 2005. Target tracking with distributed sensors: The focus of attention problem. *Computer Vision and Image Understanding* 100(1–2):225–247.

Jung, B., and Sukhatme, G. 2004. A generalized region-based approach for multi-target tracking in outdoor environments. In *Proceedings of the 2004 IEEE Int. Conf. on Robotics and Automation*, 2189–2195.

Karuppiah, D.; Grupen, R.; Hanson, A.; and Riseman, E. 2005. Smart resource reconfiguration by exploiting dynamics in perceptual tasks. In *Proceedings of the 2005 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 1854–1860.

Koyasu, H.; Miura, J.; and Shirai, Y. 2001. Realtime omnidirectional stereo for obstacle detection and tracking in dynamic environments. In *Proceedings of the 2001 IEEE/RSJ Int. Conf. on Intelligent Robots and Sysetms*, 31–36.

Krishna, K.; Hexmoor, H.; and Sogani, S. 2005. A t-step ahead constrained optimal target direction algorithm for a multi sensor surveillance system. In *Proceedings of the 2005 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 1840–1845.

Lee, L.; Romano, R.; and Stein, G. 2000. Monitoring activities from multiple video streams: Establishing a common coordinate frame. *IEEE Trans. on Pattern Analaysis and Machine Intelligence* 22(8):758–767.

Maskell, S.; Rollason, M.; Gordon, N.; and Salmond, D. 2003. Efficient particle filtering for multiple target tracking with application to tracking in structured images. *Image and Vision Computing* 21(10):931–939.

Miura, J., and Shirai, Y. 2002. Parallel scheduling of planning and action for realizing an efficient and reactive robotic system. In *Proceedings of the 7th Int. Conf. on Control, Automation, Robotics and Vision*, 246–251.

Stauffer, C., and Grimson, W. 2000. Learning patterns of activity using real-time tracking. *IEEE Trans. on Pattern Analysis and Machine Intelligence* 22(8):747–757.

Ukita, N., and Matsuyama, T. 2003. Real-time cooperative multi-target tracking by communicating active vision agents. In *Proceedings of 6th Int. Conf. on Information Fusion*, 439–446.

Yagiura, M., and Ibaraki, T. 2001. On metaheuristic algorithms for combinatorial optimization problems. *Systems and Computers in Japan* 32(3):33–54.

Robust Execution of Contingent, Temporally Flexible Plans *

Stephen A. Block and **Brian C. Williams**
Computer Science and Artificial Intelligence Laboratory,
Massachusetts Institute of Technology,
Cambridge, MA, 02139, USA
sblock@mit.edu, williams@mit.edu

Abstract

Many applications of autonomous agents require groups to work in tight coordination. To be dependable, these groups must plan, carry out and adapt their activities in a way that is robust to failure and uncertainty. Previous work has produced contingent plan execution systems that provide robustness during their execution phase, by dispatching temporally flexible plans, and during their plan extraction phase, by choosing between functionally redundant methods. Previous contingent plan execution systems use a centralized architecture in which a single agent conducts planning for the entire group. This can result in a communication bottleneck at the time when plan activities are passed to the other agents for execution, and state information is returned.

This paper introduces a robust, *distributed* executive for temporally flexible plans. To execute a plan, the plan is first distributed over multiple agents, by creating a hierarchical ad-hoc network and by mapping the plan onto this hierarchy. Second, the plan is reformulated using a distributed, parallel algorithm into a form amenable to fast dispatching. Finally, the plan is dispatched in a distributed fashion.

We then extend the distributed executive to handle contingent plans. Contingent plans are encoded as Temporal Plan Networks (TPNs), which use a non-deterministic choice operator to compose temporally flexible plan fragments into a nested hierarchy of contingencies. A temporally consistent plan is extracted from the TPN using a distributed, parallel algorithm that exploits the structure of the TPN.

At all stages of the distributed executive, the communication load is spread over all agents, thus eliminating the communication bottleneck. In addition, the distributed algorithms reduce the computational load on each agent and provide opportunities for parallel processing, thus increasing efficiency.

Introduction

The ability to coordinate groups of autonomous agents is key to many real-world tasks, such as a search and rescue mission, or the construction of a Lunar habitat. Achieving this ability dependably requires techniques that are robust to four types of disturbances: temporal uncertainty, execution uncertainty, plan failure, and communication latency. We address the first three types of disturbances by leveraging prior work on robust plan execution.

Temporal Uncertainty. Temporally flexible plans (Dean & McDermott 1987) (Muscettola 1994) allow us to model activities of uncertain duration. Use of these plans allows us to provide robustness to temporal uncertainty.

Execution Uncertainty. To adapt to uncertainty in execution times, while executing the plan in a timely manner, the dispatcher dynamically schedules events. In particular, we use the methods of (Tsamardinos, Muscettola, & Morris 1998), which use a least commitment strategy. Here, scheduling is postponed until execution time, allowing temporal resources to be assigned as-needed. To minimize the computation that must be performed on-line at dispatch time, the plan is first reformulated off-line. This scheme provides robustness to execution uncertainty, including the uncertainty in the execution times of other agents.

Plan Failure. To address plan failure, (Kim, Williams, & Abramson 2001) introduced a system called *Kirk*, that performs dynamic execution of temporally flexible plans with contingencies. These contingent plans are encoded as alternative choices between functionally equivalent sub-plans. In Kirk, the contingent plans are represented by a Temporal Plan Network (TPN) (Kim, Williams, & Abramson 2001), which extends temporally flexible plans with a nested choice operator. Kirk first extracts a plan from the TPN that is temporally feasible, before executing the plan as above. Use of contingent plans adds robustness to plan failure.

As a centralized approach, however, Kirk can be brittle to failures caused by *communication latency*. Once a plan has been extracted and reformulated it must be executed and in the case of a multi-agent plan, this involves multiple agents. Therefore, we must communicate the primitive activities to all of the agents that will take part in the execution. This creates a communication bottleneck at the master agent.

We address this limitation through a distributed version of Kirk, called D-Kirk, which performs *distributed* execution of contingent temporally flexible plans. In the distributed framework, all agents send and receive messages. This evens out communication requirements and eliminates the brittleness to communication bottlenecks. In addition, we distribute computation between all agents to reduce compu-

*The plan extraction component of this work was made possible by the sponsorship of the DARPA NEST program under contract F33615-01-C-1896, and the plan reformulation and dispatching components by the sponsorship of Boeing contract MIT-BA-GTA-1

tational complexity and take advantage of parallel processing, thus improving performance relative to the centralized architecture

D-Kirk consists of the following four phases.

1. Distribute the TPN across the processor network,
2. Select a temporally consistent plan from the TPN.
3. Reformulate the selected plan for efficient dispatching.
4. Dispatch the selected plan, while scheduling dynamically.

This paper begins with a summary of dispatchable execution. We then present the steps required to execute a temporally flexible plan in a distributed manner. These are the distribution, reformulation and dispatching algorithms and correspond to steps 1, 3 and 4 above. Our key innovation is a set of distributed algorithm that handle limited inter-agent communication and are robust to all communication delays.

Finally, we extend the work to include contingent plans. We present a summary of TPNs and of previous work by (Wehowsky 2003) for dynamically selecting a feasible plan from a TPN, which corresponds to step 2 above.

Dispatchable Execution

The task of the executive is to robustly execute a temporally flexible plan. Temporally flexible plans make use of simple temporal constraints to describe uncertain durations. A simple temporal constraint $[l, u]$ places a bound $t^+ - t^- \in [l, u]$ on duration between the start time t^- and end time t^+ of the activity or sub-plan to which it is applied.

In all that follows, we discuss temporally flexible plans in terms of a graph representation, where nodes represent time events and directed edges represent simple temporal constraints. An example temporally flexible plan is shown in graph form in Fig. 1.

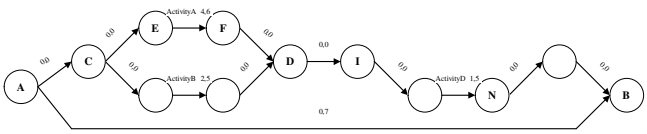


Figure 1: Example temporally flexible plan

Given a temporally flexible plan to be executed, traditional execution schemes fix the execution schedule at planning time, thus removing the temporal flexibility prior to execution. This approach leads to two problems. First, the fixed schedule lacks the flexibility to respond to temporal uncertainty at execution time, so the plan is prone to failure. Second, if we generate a very conservative schedule to increase the likelihood of successful execution, then the execution becomes sub-optimal in terms of total execution time.

We overcome this limitation with dispatchable execution (Muscettola, Morris, & Tsamardinos 1998), where scheduling is postponed until execution time. At execution time we have the most information available regarding execution history, so the dispatcher can schedule activities just-in-time. This provides robustness to uncertain durations that lie within the temporally flexible bounds of the plan.

Just-in-time scheduling introduces an added run-time computation that increases latency in the system's ability to respond to disturbances. To minimize the amount of computation that must be conducted in real-time, we use a two-stage execution strategy. Prior to execution, we use reformulation to compile the plan to a form that allows easy dispatching. In particular, we reformulate the plan to a Minimal Dispatchable Graph (MDG), which requires the minimum amount of processing at dispatch time. This is followed by dispatching, when the plan is executed. Dispatching of a plan in MDG form requires only local propagation of timing information.

To form the MDG, reformulation identifies the non-dominated edges in the plan. These are the edges along which execution information must be propagated at run time. The most simple reformulation algorithm begins by forming the All Pairs Shortest Path (APSP) graph. It then traverses the APSP graph and tests every edge to determine whether it is non-dominated. However, constructing the entire APSP graph is inefficient, requiring $O(N^2)$ time complexity and $O(N^2)$ space complexity, where N is the number of nodes in the plan. Furthermore, searching the APSP graph for non-dominated edges has $O(N^3)$ time complexity, giving an overall complexity for the simple reformulation algorithm of $O(N^3)$ in time and $O(N^2)$ in space.

We overcome this problem with fast reformulation (Tsamardinos, Muscettola, & Morris 1998). Fast reformulation extracts non-dominated edges by a series of traversals of the graph, without forming the complete APSP graph. In the worst case, we conduct a traversal from every node in the plan graph, to test whether the implicit edge from that node to the node being investigated is non-dominated. In order for the dominance tests to be conducted efficiently, we must conduct each traversal in Reverse Post Order (RPO) for the predecessor graph rooted at the relevant start node.

We can further reduce complexity by exploiting Rigid Components (RCs). A rigid component is a set of nodes whose execution times are fixed relative to each other. During reformulation we can represent each RC by a single node, thus reducing the effective value of N. The other nodes in each RC are then added back into the plan once reformulation is complete. Note that the treatment of RCs in this way is also required for the dominance tests to function correctly.

The node representing a given RC is known as the RC leader, and is the node with minimum Single Source Shortest Path (SSSP) distance from the start node. In order for it to represent the RC in the dominance tests, the edges to all other RC members are re-routed to the leader node. The member nodes themselves are connected with a doubly linked chain of edges, in increasing SSSP distance, as shown in Fig. 2. All other edges are deleted. We connect RC member nodes in this way because the doubly linked chain edges are guaranteed to be part of the MDG, so do not need to be processed in the dominance tests.

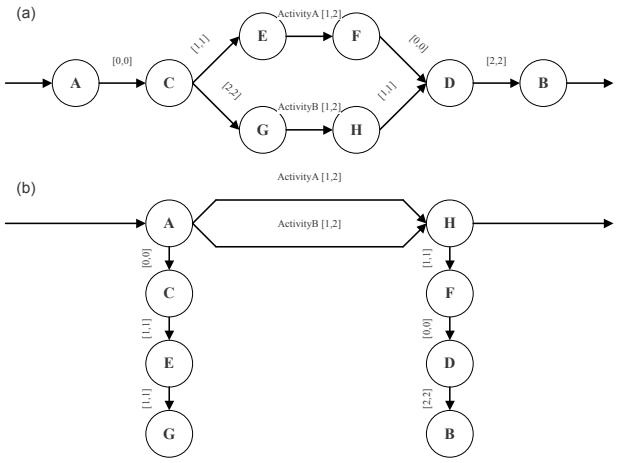


Figure 2: Rigid Components; (a) Before processing; (b) After processing

Plan Distribution

The first step of D-Kirk is to distribute the input plan over the available processors. A plan is distributed by assigning to each processor the responsibility for zero or more nodes of the plan graph, such that each node is assigned to exactly one processor. During the subsequent steps of D-Kirk, each processor maintains all data relevant to its assigned nodes.

D-Kirk begins by establishing a hierarchical, ad-hoc communication network, using the leader election algorithm in (Nagpal & Coore 1998). Given a set of processors with fixed but unknown communication availabilities, this algorithm forms a hierarchy of processors where communication is guaranteed between a processor and its *leader*, *neighbor leaders* and *followers*, as shown in Fig. 3.

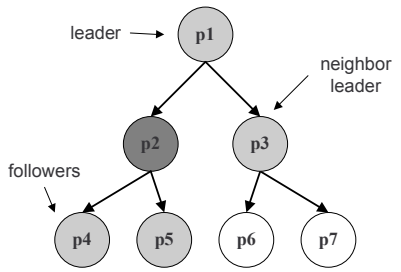


Figure 3: Example processor hierarchy, showing communication availability for processor p2

We then use a distribution algorithm (Wehowsky 2003) to assign nodes to processors in the hierarchy. The temporal consistency checking performed in the planning phase is centered around the temporal constraints between nodes of the plan. Therefore, this algorithm distributes the plan such that each pair of processors responsible for nodes linked by a temporal constraint are able to communicate.

The algorithm allows the plan to be distributed down to a level at which a processor handles only a single node; this permits D-Kirk to operate on heterogeneous systems that include computationally impoverished processors.

Reformulation

In D-Kirk we distribute the fast reformulation algorithm between all nodes, using a message passing scheme. We use a state machine approach to structure the reformulation algorithm, so each node can track its progress through the computation. This means that we do not require any existing synchronization between nodes: the state machine provides synchronization based solely on the messages received. Also, this allows a node to act only on received messages relevant to the phase of the computation with which it is currently concerned, while postponing processing of any messages for future phases that are received early.

As a result, the algorithm is robust to delays in message delivery and ensures correct operation even when messages arrive in an order different from that in which they were sent. This allows reliable operation when guarantees can not be placed upon the speed of communication channels. Also, the algorithm provides significant error checking with regard to the type of message that can be received at a given time.

The algorithm operates on the distance graph corresponding to the input plan. It requires only the communication channels guaranteed during distribution of the plan. Messages that must be sent between nodes that are not connected by known channels are redirected to follow the channels, with the re-routing handled internally by the algorithm.

Throughout the algorithm, we exploit parallel processing wherever possible. In the following subsections, we discuss the approaches used in each section of the distributed reformulation algorithm, and show how efficiency is maximized.

We also present the computational complexity of each phase of the algorithm, for comparison with the centralized case. Note that the algorithm is event driven, where an event is the receipt of a message. The computation to be performed on receipt of each message is simple, so the computational complexity is of the same order as the message complexity.

We illustrate the operation of the algorithm on the example temporally flexible plan shown in Fig. 1. Line numbers refer to the pseudo-code for the distributed reformulation algorithm shown in Fig. 4.

Rigid Component Processing (lines 1-16)

As mentioned above, for correct operation of the dominance tests, we must represent each RC by a single node. RC processing is therefore the first part of the distributed reformulation algorithm and proceeds as follows.

Form predecessor graph. The predecessor graph is the graph of shortest paths from the start node to all other nodes. To form the predecessor graph we use the distributed Bellman-Ford algorithm (Lynch 1997), algorithm to find the SSSP distance to each node (line 2). The distributed Bellman-Ford algorithm requires only local knowledge of the graph at each node, hence allowing the SSSP calculation to be performed locally. To ensure that the algorithm converges in time linear in the number of nodes, the Bellman-Ford algorithm is run synchronously. This algorithm is fully parallel. Once the SSSP distances have been computed, we use them to form the predecessor graph (line 3).

1: **Process Rigid Components**
2: Compute SSSP distances from phantom node using synchronous distributed Bellman-Ford
3: Form predecessor graph using SSSP distances
4: Perform DFS on predecessor graph and record RPO
5: **for** Each node in the graph, taken in RPO order **do**
6: Perform DFS on transposed predecessor graph to extract members of this RC and their edges
7: Determine member node with minimum SSSP distance and set as RC leader
8: **for** Each member node **do**
9: Form edges for RC doubly linked chain and record as members of MDG
10: Delete all other edges
11: **end for**
12: **for** Each edge **do**
13: Relocate to RC leader
14: Inform remote node of the relocation
15: **end for**
16: **end for**
17: **Perform Dominance Tests**
18: **for** Each RC leader **do**
19: Compute SSSP distances from this node for the graph of RC leaders using synchronous distributed Bellman-Ford
20: Form predecessor graph using SSSP distances
21: Perform DFS on predecessor graph and record RPO
22: Begin traversal in RPO
23: **for** Each node traversed **do**
24: Use *minimum* and *non-positive* data to apply dominance tests
25: **if** Implicit edge is non-dominated **then**
26: Record implicit edge as member of MDG
27: **end if**
28: Update *minimum* and *non-positive* data
29: Propagate *minimum* and *non-positive* data to successors
30: **end for**
31: Record non-dominated edges as members of MDG
32: **end for**
33: **Initialize Execution Windows**
34: Compute SSSP distances from start node using outgoing non-negative MDG edges using synchronous distributed Bellman-Ford
35: Record SSSP distances as upper bound of execution windows
36: Compute SSSP distances from start node using incoming non-positive MDG edges using synchronous distributed Bellman-Ford
37: Record SSSP distances as lower bound of execution windows

Figure 4: Distributed Reformulation Algorithm

Note that for the purpose of extracting RCs, we perform the SSSP calculation from a *phantom node*, a virtual node which has edges of zero length to every other node in the plan. The distance graph corresponding to the example plan in Fig. 1, with phantom node added, is shown in Fig. 5(a). The SSSP distances and predecessor graph for the example plan are shown in Fig. 5(b).

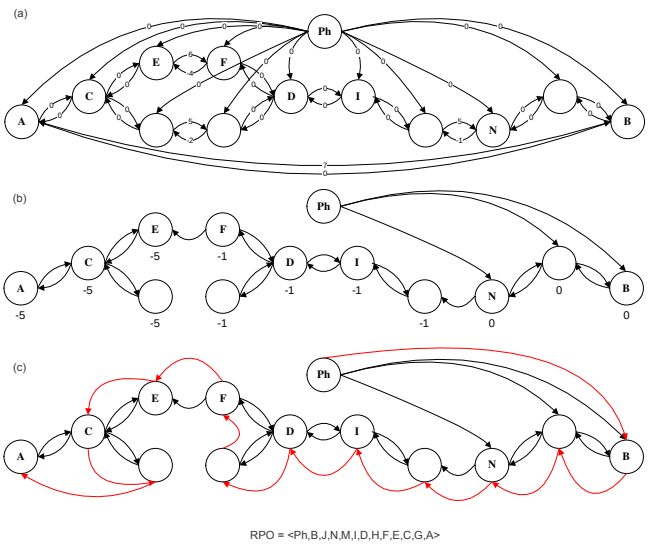


Figure 5: Reformulation example: (a) Distance graph with phantom node; (b) SSSP distances and predecessor graph; (c) RPO

The complexity of this stage is dominated by the SSSP calculation, which has computational complexity $O(Ne)$, where e is the number of edges at each node.

Extract reverse post order. The post order is the order in which nodes are removed from the search queue during search. We extract the Reverse Post Order (RPO) from a Depth First Search (DFS) on the predecessor graph (line 4). Since the RPO is inherently ordered, this part of the algorithm can not be conducted in parallel and is entirely serial. The computational complexity for each node is $O(e)$.

Note that in the presence of the phantom node, this search is more complicated than a simple DFS. We simulate a DFS search from the phantom node by starting a DFS from every node that has a zero SSSP distance. The RPO for the example plan is shown in Fig. 5(c).

A given node must complete its part in forming the predecessor graph before it can take part in RPO extraction, but we do not require all nodes to have completed the predecessor graph for the RPO extraction to begin. The RPO extraction process is begun as soon as the start node has completed its part in forming the predecessor graph, so it occurs concurrently with formation of the predecessor graph, waiting for a node to finish its part in forming the predecessor graph where necessary.

Process rigid components. We determine the members of each rigid component with a series of DFSs on the transposed predecessor graph. The transposed predecessor graph for the example plan is shown in Fig. 6(a). We start a DFS from every node in the graph, with the order determined by the predecessor graph RPO calculated above (line 5). The nodes visited in each DFS belong to a single RC.

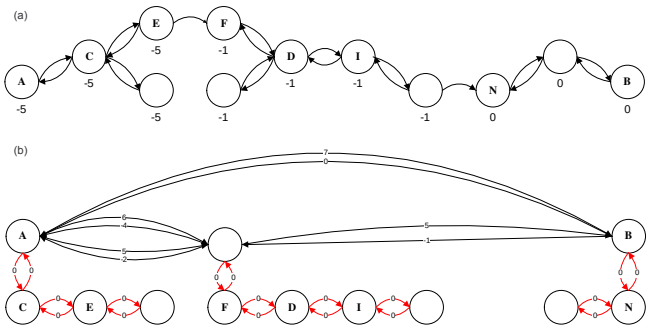


Figure 6: Reformulation example: (a) Transposed predecessor graph; (b) Plan with RC processing complete

During each DFS, we gather the list of member nodes and the list of edges belonging to them (line 6). After each DFS is complete, the relevant start node assigns as RC leader the member node with the minimum SSSP distance (line 7). It also sends messages to all members of the RC to instruct them to form the doubly linked chain of edges that connects the member nodes (line 9) and to delete all other edges (line 10). If the receiving node is the RC leader, it also adds the rearranged edges from the other members of the RC (line 13). Finally, the start node sends messages to inform the nodes at the far end of the RC members' edges that the edges have been moved to the RC leader (line 14). The example plan with RC processing complete is shown in Fig. 6(b).

Within each DFS, processing is performed in parallel: multiple branches of the tree are searched simultaneously. However, the DFSs must be strictly ordered relative to each other for the RC extraction to be successful, so this is done sequentially.

The complexity of this phase is dominated by the extraction of the members of the RC and by rearranging edges to the leader, giving a computational complexity of $O(e)$.

The RC extraction process begins at the first node in the RPO. By definition, this is the last node to complete the RPO extraction phase, so RC extraction does not begin until RPO extraction is complete.

Since in the presence of the phantom node the overall start node is not necessarily the first in the RPO, the RC processing phase is initiated by a message from the overall start node to the first node in the RPO.

Dominance Test Traversals (lines 17-32)

The dominance tests determine whether an edge is a member of the MDG and are applied in a series of graph traversals. The traversals are conducted on the subset of the plan graph comprised of RC leaders (line 18). In a traversal started at node A, the dominance test applied as we traverse node B determines whether or not the implicit edge AB is a member of the MDG.

A given node must complete its part in RC processing before it can begin the dominance test procedure. However, we do not need all nodes to have completed the RC phase before we start the dominance tests, so these two phases run concurrently, waiting for nodes to complete RC processing where required.

Each traversal must follow the RPO for the predecessor graph rooted at the start node of the traversal. The procedure for each traversal is as follows.

Form predecessor graph. We form the predecessor graph as described above, but we do not use a phantom node and we ignore any edges to RC non-leaders (lines 19-20). The computational complexity per traversal remains $O(Ne)$.

Extract reverse post order. We extract the RPO as described above, again not using a phantom node and ignoring any edges to RC non-leaders (line 21). The computational complexity per traversal remains $O(e)$.

Traverse and apply dominance tests. We conduct the traversal in the order given by the RPO (line 22). The traversal begins at the first node in the RPO and by definition, this is the last node to complete the RPO extraction phase, so the traversal does not begin until RPO extraction is complete.

At each node in the traversal, we send messages to all predecessor nodes with the values of the following two pieces of data (lines 28-29).

- **minimum** : the minimum SSSP distance encountered on this traversal.
- **non-positive** : whether a non-positive SSSP distance has been encountered on this traversal.

When the traversal reaches a node, it uses these pieces of data to determine whether or not the implicit edge is dominated (line 25). If the edge is not dominated, it is recorded locally (line 26). The edge is also recorded in a list which is passed back to the start node of the traversal for recording there (line 31).

The complexity is dominated by the need to propagate the data used in the dominance tests to all predecessors, giving a computational complexity per traversal of $O(e)$.

Once a traversal is complete, the start node of the traversal uses a message to initiate a traversal from the next node in the graph. In this way, successive traversals are conducted serially. The SSSP distances, predecessor graph, RPO and non-dominated edges for the example plan are shown in Figs. 7(a), (b) and (c) for the traversals from nodes B, H and A respectively.

Initialize Execution Windows (lines 33-37)

Once the dominance test traversals are complete, we must initialize the execution window for each node before dispatching. A node can not determine when it has completed the dominance test phase because we can not easily calculate how many traversals it will be involved in. This is because, in general, a node is only reachable from a subset of the nodes in the plan for the purposes of SSSP distance and RPO calculations. Therefore, when the traversal from a given node is complete, this node sends a message to inform the overall start node that this is the case. Once all such messages have been received, we know that the dominance test phase is complete and the execution windows can be calculated.

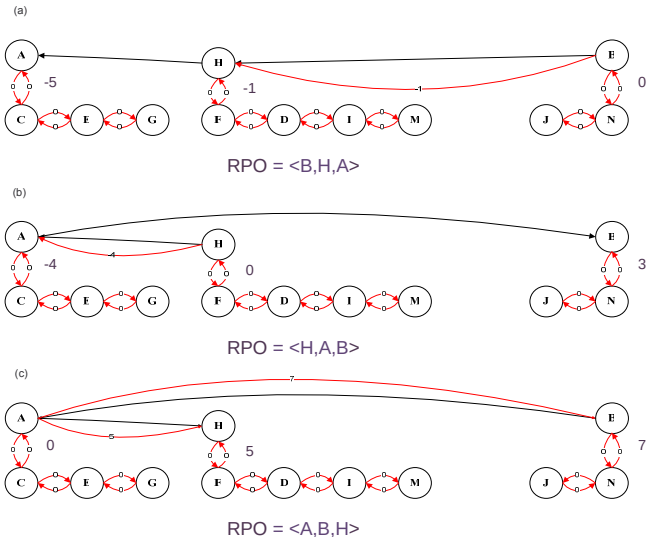


Figure 7: Reformulation example: SSSP distances, predecessor graph and non-dominated edges for MDG traversals; (a) From node B; (b) From node H; (c) From node A

Calculate upper bounds. We determine the upper execution bound on each node with a SSSP calculation from the start node. We use the distributed Bellman-Ford algorithm described above, but we do not use the phantom node and we consider only outgoing non-negative MDG edges (lines 34-35). In the worst case, the number of MDG edges per node is $O(e)$, so the computational complexity per traversal is $O(Ne)$.

Calculate lower bounds. We determine the lower execution bound on each node with a SSSP calculation from the start node. We use the distributed Bellman-Ford algorithm described above, but again we do not use the phantom node and we consider only incoming non-positive MDG edges (lines 36-37). Again, the number of MDG edges per node is $O(e)$, so the computational complexity per traversal is $O(Ne)$. Since we can not conduct multiple Bellman-Ford calculations simultaneously, we complete the upper bound calculations before we begin those for the lower bounds. The complete MDG and initial execution windows for the example plan are shown in Fig. 8.

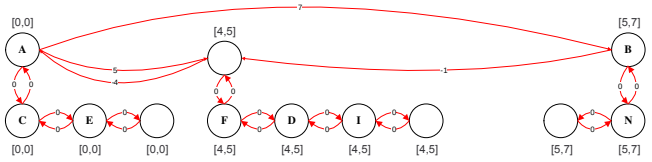


Figure 8: Reformulation example: Complete MDG and initial execution windows

Complexity

Using the complexities of each stage described above, we obtain a worst case overall computational complexity of $O(N^2e + Ne)$ for the distributed reformulation algorithm. Both of these terms are due to the Bellman-Ford SSSP calculations: approximately N calculations of complexity $O(Ne)$ used to obtain the predecessor graphs for RC processing and dominance test traversals, and 2 calculations of complexity $O(Ne)$ for the execution windows.

In the centralized case, the centralized Bellman-Ford algorithm has complexity $O(NE)$, where E is the total number of edges in the plan, giving an overall computational complexity of $O(N^2E)$. The improvement due to distributed processing, therefore, is approximately a factor of $\frac{E}{e} \approx N$.

Dispatching

Once the plan has been reformulated to an MDG, the dispatcher executes the activities in real time. We base our distributed algorithm on the dispatching algorithm of (Muscettola, Morris, & Tsamardinos 1998). We distribute the dispatching algorithm over all nodes, using a message passing scheme.

As with the reformulation algorithm, we use a state machine approach to provide robustness to delays in message delivery. This allows the algorithm to function correctly without requiring any synchronization between processors.

However, synchronization is required for the precisely timed execution of tasks. For this we assume a synchronous execution model. In particular, we require that each processor has a synchronized clock. The task of achieving this synchronization is not trivial, but is beyond the scope of this work. Note however, that approximate synchronization, to within the delivery time of a single message, is sufficient for many practical applications.

During dispatching, the processors operate independently, monitoring incoming messages and their clock to determine when their nodes' activities can be executed. The only messages used are those sent to inform neighbor nodes that a node has executed. The information carried in these messages is used to update the lower and upper bounds on a node's execution window and to propagate enablement conditions. Line numbers in the following description of the algorithm refer to the pseudo-code shown in Fig. 9.

First, a node must wait to be *enabled* (lines 1-4). A node is enabled when all nodes that must execute before it have been executed. These nodes are identified as those which are found at the end of outgoing non-positive MDG edges (line 2). While waiting, the node responds to incoming messages (line 3).

Once enabled, a node must wait for the current time to enter its execution window and for all uncontrollable activities that end at this node to complete (lines 6-14). While waiting, the node checks that the current time does not exceed the upper bound of the execution window (line 7), else the plan execution fails (line 8). Also, the node continues to respond to incoming messages (line 10).

Once the current time is in the execution window and all uncontrollable activities have completed, we can execute the node (lines 15-18). First, we stop any controllable activities that end at this node (line 16) and then start any activities

1: **Wait For Enablement**
2: **while** All nodes on outgoing non-positive edges have not executed **do**
3: Process received EXECUTED messages
4: **end while**
5: **Wait For Timing**
6: **while** No error **do**
7: **if** Current time has exceeded upper time bound **then**
8: Execution failure
9: **end if**
10: Process received EXECUTED messages
11: **if** current time is within execution window AND All uncontrollable end activities are complete **then**
12: Break
13: **end if**
14: **end while**
15: **Execute node**
16: Stop all controllable end activities
17: Start all start activities
18: Inform neighbor nodes that node has executed

Figure 9: Distributed Dispatching Algorithm

which begin at this node (line 17). Finally, we send messages to inform neighbor nodes that the node has executed (line 19).

The number of messages sent by each node is determined by the number of edges connected to it in the MDG. In the worst case, this number is $O(e)$.

In the centralized case, the lead node must send messages to every other node in the plan to instruct them to execute their activities, giving a peak message complexity of $O(E)$. Therefore, compared to the centralized case, D-Kirk reduces the number of messages at dispatch time, when we must operate in real time and are most susceptible to communication delays.

Furthermore, since the computational complexity is directly proportional to the number of messages received, D-Kirk improves this too.

Temporal Plan Networks

In order to encode contingencies, a Temporal Plan Network (TPN) augments the temporally flexible plan representation with a `choose` operator. The `choose` operator allows us to specify nested choices in the plan, where each choice is an alternative sub-plan that performs the same function.

The primitive element of a TPN is an `activity`$[l, u]$, which is an executable command whose duration is bounded by a simple temporal constraint. A simple temporal constraint $[l, u]$ places a bound $t^+ - t^- \in [l, u]$ on duration between the start time t^- and end time t^+ of the activity or contingent sub-plan to which it is applied. A TPN is built from a set of primitive activities and is defined recursively using the `choose`, `parallel` and `sequence` operators, taken from the Reactive Model-based Programming Language (RMPL) (Williams *et al.* 2003). A TPN encodes all executions of a non-deterministic concurrent, timed program, comprised of these operators.

- `choose`$(TPN_1, \ldots, TPN_N)$ introduces multiple subnetworks of which only one is to be chosen. A choice variable is used at the start node to encode the currently selected subnetwork. A choice variable is *active* if it falls within the currently selected portion of the TPN.
- `parallel`$(TPN_1, \ldots, TPN_N)\,[l, u]$ introduces multiple subnetworks to be executed concurrently. A simple temporal constraint is applied to the entire network.
- `sequence`$(TPN_1, \ldots, TPN_N)\,[l, u]$ introduces multiple subnetworks which are to be executed sequentially. A simple temporal constraint is applied to the entire network.

Graph representations of the `activity`, `choose`, `parallel` and `sequence` network types are shown in Fig. 10. Nodes represent time events and directed edges represent simple temporal constraints. A choice node is shown as an inscribed circle.

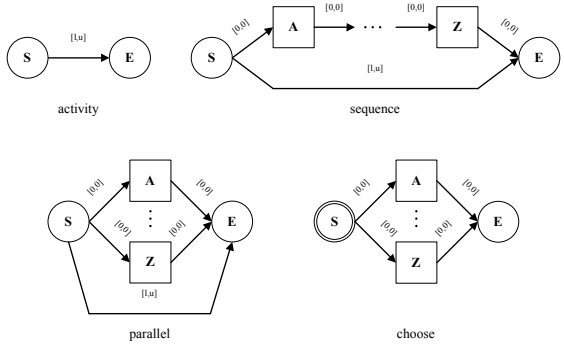


Figure 10: TPN Constructs

A *temporally consistent* plan is obtained from the TPN if and only if a *feasible choice assignment* is found. See (Wehowsky 2003) for a more precise definition.

Definition 1 *A temporally flexible plan is **temporally consistent** if there exists an assignment of times to each event such that all temporal constraints are satisfied.*

Definition 2 *A **feasible choice assignment** is an assignment to the choice variables of a TPN such that 1) all active choice variables are assigned, 2) all inactive choice variables are unassigned, and 3) the temporally flexible plan (program execution) corresponding to this assignment is **temporally consistent**.*

Plan Extraction

The plan extraction phase of D-Kirk consists of two interleaved processes: generation of candidate plans and testing them for temporal consistency.

The candidate plans correspond to different assignments to the choice variables at each choice node and are obtained by solving a conditional CSP (Mittal & Falkenhainer 1990). The D-Kirk planning algorithm uses parallel, recursive, depth first search to make these assignments.

Consistency checking is implemented using the distributed Bellman-Ford SSSP algorithm and is run on the distance graph corresponding to the portion of the TPN that

represents the current candidate. Temporal inconsistency is detected as a negative weight cycle (Dechter, Meiri, & Pearl 1991).

The planning algorithm exploits the hierarchical structure of the TPN to allow parallel processing. Consistency checking is interleaved with candidate generation, such that D-Kirk simultaneously runs multiple instances of the distributed Bellman-Ford algorithm on isolated subsets of the TPN. D-Kirk uses a distributed message-passing architecture that employs the following messages for candidate plan generation.

- `findfirst` instructs a network to make the initial search for a consistent set of choice variable assignments. If a node at level n in the hierarchy receives a `findfirst` message, it propagates it to all of its subnetworks at level $n + 1$ simultaneously. If each subnetwork finds a consistent assignment, we then check for consistency at level n.
- `findnext` is used when a network is consistent internally, but is inconsistent with other networks. In this case, D-Kirk uses `findnext` messages to conduct a systematic search for a new consistent assignment, in order to achieve global consistency. To achieve this, a node at level n in the hierarchy receiving a `findnext` message forwards the message to each subnetwork at level $n+1$ in turn. When a new consistent assignment to a subnetwork is found, we check for consistency at level n. Therefore, a successful `findnext` message will cause a change to the value assigned to a single choice variable, which may in turn cause other choice variables to become active or inactive.
- `fail` indicates that no consistent set of assignments was found and hence the current set of assignments within the network is inconsistent.
- `ack`, short for acknowledge, indicates that a consistent set of choice variable assignments has been found.

Whenever a node initiates search in its subnetworks, using `findfirst` or `findnext` messages, the relevant processors search the subnetworks simultaneously. This is the origin of the parallelism in the algorithm.

The planning phase of D-Kirk offers an improvement in computational complexity compared to a centralized architecture. The distributed Bellman-Ford algorithm has time complexity $O(Ne)$, compared to $O(NE)$ for the centralized version of the algorithm. Overall, the worst-case computational complexity of the planning algorithm remains exponential, due to the candidate generation phase.

An example TPN containing a single choice node I is shown in Fig. 11. The only feasible choice assignment is the pathway through nodes M and N, and this gives rise to the temporally consistent plan shown in Fig. 1.

Conclusion

To summarize, this paper introduced D-Kirk, a distributed executive that performs robust execution of contingent, temporally flexible plans. In particular, D-Kirk operates on Temporal Plan Networks (TPNs) and distributes both data and processing across available processors. D-Kirk employs

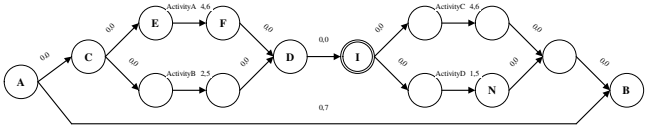


Figure 11: Reformulation example: Input TPN

a series of distributed algorithms that first, form a processor hierarchy and assign TPN subnetworks to each processor; second, search the TPN for a temporally consistent plan; third, reformulate the selected plan to a form amenable to execution and; finally, dispatch the plan. This distributed approach spreads communication evenly across the processors, thus eliminating the bottleneck in communication at dispatch time that is present in a centralized architecture. Furthermore, the distributed algorithms reduce the computational load on each processor at all four stages of execution and allow concurrent processing for increased performance.

References

Dean, T., and McDermott, D. 1987. Temporal database management. *Artificial Intelligence* 32:1–55.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence* 49:61–95.

Kim, P.; Williams, B.; and Abramson, M. 2001. Executing reactive, model-based programs through graph-based temporal planning. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-2001)*.

Lynch, N. 1997. *Distributed Algorithms*. Morgan Kaufmann.

Mittal, S., and Falkenhainer, B. 1990. Dynamic constraint satisfaction problems. In *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-1990)*.

Muscettola, N.; Morris, P.; and Tsamardinos, I. 1998. Reformulating temporal plans for efficient execution. In *Principles of Knowledge Representation and Reasoning*, 444–452.

Muscettola, N. 1994. *HSTS: Integrating Planning and Scheduling*. Morgan Kaufmann.

Nagpal, R., and Coore, D. 1998. An algorithm for group formation in an amorphous computer. In *Proceedings of the Tenth International Conference on Parallel and Distributed Systems (PDCS-1988)*.

Tsamardinos, I.; Muscettola, N.; and Morris, P. 1998. Fast transformation of temporal plans for efficient execution. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-1998)*, 254–261.

Wehowsky, A. F. 2003. Safe distributed coordination of heterogeneous robots through dynamic simple temporal networks. Master's thesis, MIT, Cambridge, MA.

Williams, B. C.; Ingham, M.; Chung, S.; and Elliott, P. 2003. Model-based programming of intelligent embedded systems and robotic explorers. In *IEEE Proceedings, Special Issue on Embedded Software*.

Coordinated Plan Management Using Multiagent MDPs

David J. Musliner
Honeywell Laboratories
david.musliner@honeywell.com

Edmund H. Durfee, Jianhui Wu, Dmitri A. Dolgov
University of Michigan
durfee,jianhuiw,ddolgov@umich.edu

Robert P. Goldman
SIFT, LLC
rpgoldman@sift.info

Mark S. Boddy
Adventium Labs
mark.boddy@adventiumlabs.org

Introduction

For the past several years, we have been developing multi-agent technology to help humans coordinate their activities in complex, dynamic environments. In recent work on the DARPA COORDINATORS program, we have developed multi-agent Markov-decision process (MDP) techniques for distributed plan management. The COORDINATORS problems arrive in distributed form, with different agents getting local views of their portion of the problem and its relationship to others. Even so, the individual agents' MDPs that capture their local planning and scheduling problem can be too large to enumerate and solve. Furthermore, the COORDINATORS agents must build and execute their plans in real-time, interacting with a world simulation that makes their actions have uncertain outcome.

Accordingly, we have developed an embedded agent system that negotiates to try to find approximately-optimal distributed policies within tight time constraints. Our work draws together and extends ideas in multi-agent Markov decision processes, real-time computing, negotiation, meta-level control, and distributed constraint optimization. Contributions of our work include "unrolling" techniques for translating local hierarchical task networks to MDPs, "informed" heuristic search control of the unrolling process, and negotiation methods for allocating responsibilities across cooperating agents and using those allocations to influence local policy construction.

In the rest of this paper, we describe our approach in more detail. We begin by summarizing the challenges in distributed plan management embodied in the COORDINATORS problem, and the TÆMS representation used to model the actions and interactions requiring coordination. We then describe how we translate the problem represented in TÆMS into an MDP, and the strategies that we use for finding policies when the MDP state space exceeds the time and/or space bounds for our system. After that, we discuss the challenge of using inter-agent negotiation to coordinate the agents' policies. Finally, we point out the limitations of our initial implementation, and we outline our plan to improve the management of uncertain and unexpected events by more fully integrating ongoing deliberation and coordination.

The COORDINATORS Problem

COORDINATORS is a research program funded by DARPA IPTO to identify, prototype, and evaluate well-founded technical approaches to scheduling and managing distributed activity plans in dynamic environments. As a motivating example, consider the following scenario. A hostage has been taken and might be held in one of two possible locations. Rescuing the hostage requires that both possible locations are entered by special forces *simultaneously*. As the activities to move personnel and materiel into place are pursued, delays can occur, or actions to achieve precursor objectives might have unexpected results (e.g., failure). COORDINATOR agent systems will be associated with the various human participants. COORDINATOR agents should monitor the distributed plans and manage them as the situation evolves, to increase their effectiveness and make them more likely to succeed.

In general, a set of COORDINATOR agents is meant to work together to maximize the reward gained by the group as a whole. In other words, the problem is to compute an effective *joint* policy for the agent society, in which the actions taken by one agent can depend on the state of the group as a whole, not just the local state of that agent. The agents

are time-pressured: each agent must make timely action decisions during execution. Furthermore, the problem must be solved in a distributed fashion.

Each agent's partial model includes the actions that the agent can execute, which are stochastic, rather than deterministic, and some of the actions its peers can perform. The model also provides *partial* information about the rewards that the society as a whole will receive for reaching various states. This model is not static: the agent can receive model updates during execution. Therefore, agents must be able to manage and reformulate policies reactively.

The problems are formulated as complex hierarchical task networks, which we translate to MDPs. It is our hypothesis that MDPs provide the appropriate modeling tool for representing the COORDINATORS problem, and that each agent should generate, store, and follow the best policy that its knowledge and available resources (time, etc.) allow. To explore this approach, we must develop distributed negotiation techniques for coordinating the policy-finding activities of the various agents, and we must provide a *local* utility model that will cause the individual agents to (approximately) maximize group expected utility by (approximately) maximizing their local expected utility.

The intractable space of joint multiagent policies means that our negotiation protocols for collaboratively arriving at joint policies can explore only a small portion of the joint policy space. Our approach reduces this full space by projecting down to an alternative space of joint "commitments" which bias the agents' local policy formulation processes. However, because the MDPs are generally too large to create and solve within the environment's time limits, the agents can only formulate partial and approximate policies.We have developed single-agent policy-finding techniques that enable an agent to flexibly trade off the quality of a policy for time. At runtime, the agents monitor their changing local states, and model the changes to each other's states, tracking their passage through the MDP state space and taking the appropriate actions based on their policies.

C-TÆMS

COORDINATORS researchers have jointly defined a common problem domain representation based on the original TÆMS language (Horling *et al.* 1999). The new language, C-TÆMS, provides a semantically sound subset of the original language, representing multi-agent hierarchical tasks with probabilistic expectations on their outcomes (characterized by quality, cost, and duration) and complex hard and soft interactions (Boddy *et al.* 2005). Unlike other hierarchical task representations, C-TÆMS emphasizes complex reasoning about the utility of tasks, rather than emphasizing interactions between agents and the state of their environment.

C-TÆMS permits a modeler to describe hierarchically-structured tasks executed by multiple agents. A C-TÆMS task network has agents and nodes representing *tasks* (complex actions) and *methods* (primitives). Nodes are temporally extended: they have durations (which may vary probabilistically), and may be constrained by release times (earliest possible starts) and deadlines. At any time, each C-TÆMS agent can be executing at most one of its methods.

A C-TÆMS model is a discrete stochastic model: methods have multiple possible outcomes. Outcomes dictate the *duration* of the method, its *quality*, and its *cost*. Quality and cost are unit-less, and there is no fixed scheme for combining them into utilities. For the initial COORDINATORS experiments, we dispense with cost, so that quality may be thought of as utility. Methods that violate their temporal constraints yield zero quality (and are said to have *failed*).

Every task in the hierarchy has associated with it a "quality accumulation function" (QAF) that describes how the quality of its children are aggregated up the hierarchy. The QAFs combine both logical constraints on subtask execution and how quality accumulates. For example, a :MIN QAF specifies that all subtasks must be executed and must achieve some non-zero quality in order for the task itself to achieve quality, and the quality it achieves is equal to the minimum achieved by its subtasks. The :SYNCSUM QAF is an even more interesting case. Designed to capture one form of synchronization across agents, a :SYNCSUM task achieves quality that is the sum of all of its subtasks that start at the same time the earliest subtask starts. Any subtasks that start later cannot contribute quality to the parent task.

Traditional planning languages model interactions between agents and the state of their environment through preconditions and postconditions. In contrast, C-TÆMS does not model environmental state change at all: the only thing that changes state is the task network. Without a notion of environment state, in C-TÆMS task interactions are modeled by "non-local effect" (NLE) links indicating inter-node relationships such as enablement, disablement, facilitation, and hindrance. Complicating matters significantly is the fact

that these NLEs may have associated delays. We will discuss the implications of all of these in terms of developing Markov models of the COORDINATORS problem shortly.

Figure 1 illustrates a simple version of the two-agent hostage-rescue problem described earlier. The whole diagram shows a global "objective" view of the problem, capturing primitive methods that can be executed by different agents (A and B). The agents in a COORDINATORS problem are *not* given this view. Instead, each is given a (typically) incomplete "subjective" view corresponding to what that individual agent would be aware of in the overall problem. The subjective view specifies a subset of the overall C-TÆMS problem, corresponding to the parts of the problem that the local agent can directly contribute to (e.g., a method the agent can execute or can enable for another agent) or that the local agent is directly affected by (e.g., a task that another agent can execute to enable one of the local agent's tasks). In Figure 1, the unshaded boxes indicate the subjective view of agent-A, who can perform the primitive methods Move-into-Position-A and Engage-A. The "enable" link indicates a non-local effect dictating that the Move-into-Position-A method must be completed successfully before the agent can begin the Engage-A method. The diagram also illustrates that methods may have stochastic expected outcomes; for example, agent-B's Move-into-Position-B method has a 40% chance of taking 25 time units and a 60% chance of taking 35 time units. The :SYNCSUM QAF on the Engage task encourages the agents to perform their subtasks starting at the same time (to retain the element of surprise).

Execution environment

COORDINATORS agent designs are evaluated in a real-time simulation environment. When agents first wake up in the simulation they are given their local, subjective view problem description and an initial schedule of activities. The initial schedule captures the notion that the agents already have some nominal plan of what they should do. One of the prime objectives of COORDINATORS is to manage responses to uncertain outcomes and task model changes.

During evaluation, the agents send commands to the simulator specifying what methods they want to initiate, and the simulator randomly determines duration and outcome quality according to the distributions in the objective C-TÆMS model. In order to achieve the best results, agents will need to adapt their courses of action to the simulated events. Agents can use a combination of precomputation (policy generation) and reaction to adapt their plans. However, the simulated environment won't wait for the agents to replan; each simulated time tick is tied firmly to wall-clock time.

In addition to the stochastic outcomes that are modeled in C-TÆMS, the simulator can inject unmodeled events. Such unmodeled events can include adding nodes to the C-TÆMS network, removing nodes, or modifying duration/quality distributions for existing nodes. Agents must manage their plans at runtime to account for these changes, and they must also contend with the fact that only a subset of the agents may be aware of the unexpected change.

Multiagent MDPs for distributed plan management

Given a (fixed) C-TÆMS task network and the fact that method outcomes are stochastic, we frame the problem as building a *policy* that dictates how each agent in the network chooses methods at every point in time. In earlier work on TÆMS, the objective was to find a satisfactory balance among some combination of quality, cost, and duration (Wagner, Garvey, & Lesser 1998). In C-TÆMS, by contrast, the problem is simply to find a policy that maximizes the network's expected quality (utility).

Markov Decision Processes

We assume that the reader has a grasp of the basic definitions of Markov Decision Processes; we recommend Puterman's text (Puterman 1994) for more specifics. Briefly, an MDP is akin to a finite state machine, except that transitions are probabilistic, rather than deterministic or nondeterministic. Agents may also receive reward (which may be either positive or negative) for entering some states. Typically, this reward is additive over any trajectory through the state space (some adjustments are needed in the case of MDPs of infinite duration). The solution to an MDP is a *policy* — an assignment of action choice to every state in the MDP — that maximizes *expected utility*. MDPs' advantages are that they offer a sound theoretical basis for decision-making and action under uncertainty, and that there are relatively simple, polynomial algorithms for finding optimal policies.[1]

An agent's C-TÆMS task model may be thought of as specifying a *finite-horizon* MDP. The problems are finite-horizon because C-TÆMS problems have finite duration, with no looping or method retries. However, the MDP tends

[1] But polynomial in the (often large) size of the state space!

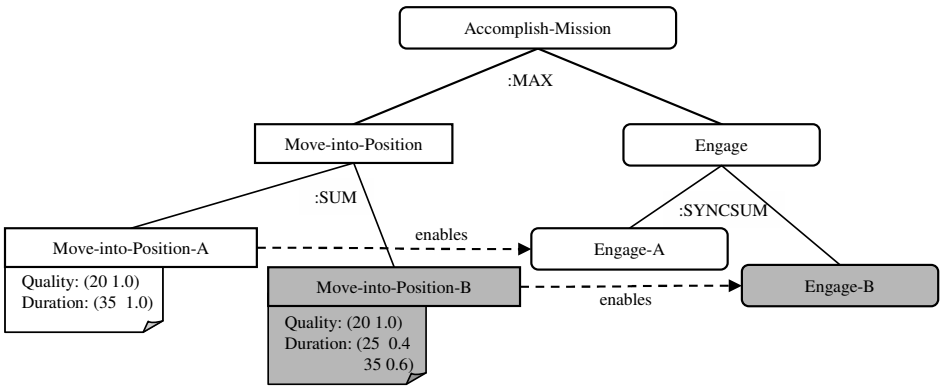


Figure 1: A simple C-TÆMS task network for two agents, illustrating some of the representation features. Some details have been omitted for brevity.

to be quite large for even modest-sized C-TÆMS problems because of the branching factor associated with uncertain outcomes, and because of the temporal component of the problem. Multi-agent C-TÆMS MDPs are even worse.

The COORDINATORS problem differs from most problems treated as MDPs in two important ways. First, the problem is inherently distributed and multi-agent, so that in the objective view, multiple actions can be executed simultaneously. For this reason, if one were to formulate a centralized COORDINATORS problem directly as an MDP, the action space would have to be a tuple of assignments of actions to each agent. As one would expect, this causes an explosion in the state space of the problem. A second difference is that the actions in the COORDINATORS domain are temporally extended, rather than atomic. Such "durative" actions can be accommodated, but only at the cost of a further explosion in the state space. Other aspects of the COORDINATORS problem make its MDP formulation tricky and increase the state space even more. For example, the delays associated with NLEs such as "enables" links require the MDP state to hold some execution outcome history.

Unrolling TÆMS task nets

We translate C-TÆMS task networks by "unrolling" them into MDPs that make explicit the state space implicit in the task net. For any agent, the C-TÆMS task network defines a possible state space and transition function. A C-TÆMS agent's state may be defined as a tuple: $\langle t, M \rangle$, where t is the current time, and M is a vector of method outcomes. If $\mathcal{M}$ is the set of methods a TÆMS agent can execute, we can assign to $\mathcal{M}$ an arbitrary numbering, $1...n$ for $n = |\mathcal{M}|$. Then M is a set of tuples, $\langle i, \sigma(i), \delta(i), q(i) \rangle$: the index of the method, its start time, duration, and quality. This information is sufficient (but not always all necessary) to give a state space that has the Markov property. The C-TÆMS task network, with its duration and quality distributions, defines a transition function. For example, if an agent executes a method i starting at time t, yielding a duration $\delta(i)$ and a quality $q(i)$, that is a state transition as follows:

$$\langle t, M \rangle \rightarrow \langle t + \delta(i), M \cup \{\langle i, t, \delta(i), q(i) \rangle\} \rangle$$

Our techniques "unroll" the state space for the MDP from its initial state ($\langle 0, \emptyset \rangle$) forward.[2] From the initial state, the algorithms identify every possible method that could be executed, and for each method every possible combination of duration-and-quality outcomes, generating a new state for each of these possible method-duration-quality outcomes. Each state is then further expanded into each possible successor state, and so on. For states where no methods can apply, a "wait-method" is generated that leads to a later state where some non-wait method has been enabled (or the scenario has ended). The unrolling process ends at leaf states whose time index is the end of scenario. The code for performing this unrolling was adapted from previous state-space unrollers developed at SIFT for applications in optimal cockpit task allocation (Miller, Goldman, & Funk 2003).

There are a number of complexities in performing this unrolling. For example, the set of enabled methods is a complex function of the current state, influenced by temporal constraints, NLEs, etc. Our unroller tries to eliminate

[2]Note that we can just as easily start unrolling *in medias res*, by starting from a state in which the agent has already executed some actions.

dominated action choices without enumerating the resulting states. For example, once one task under a :MIN QAF has failed (gotten zero quality) then none of the :MIN node's remaining children should ever be tried, because they cannot increase the :MIN node's quality.[3]

Furthermore, we can exploit the structure of the C-TÆMS task model, and its QAFs, to collapse together states that are equivalent with respect to future action choice and final quality. These techniques, and especially the equivalent-state folding, give exponential reductions in state space size.

Informed Unrolling

Since full enumeration of even single-agent C-TÆMS MDPs is impractical (the state space is too large), we have developed a technique for heuristic enumeration of a subspace of the full MDP. Our *informed unroller* (IU) algorithm prioritizes the queue of states waiting to be unrolled based on an estimate of the likelihood that the state would be encountered when executing the optimal policy from the initial state. The intent is to guide the unrolling algorithm to explore the most-probable states first. Because the probability of reaching a state is dependent on the policy, the IU intersperses policy-formulation (using the Bellman backup algorithm) with unrolling. However, the quality of a state at the edge of the partially-unrolled state space is generally difficult to assess, since quality often only accrues at the end of the entire execution trace (i.e., the domains include delayed reward). Thus, we have developed a suite of alternative heuristics for estimating intermediate state quality.

We have experimented with several different heuristic functions to evaluate edge states and guide the informed unroller. Currently the most consistent performance is achieved by a heuristic that emphasizes unrolling the highest (estimated) probability states first. However, computing the heuristic function and sorting the openlist is an expensive operation. Therefore we constrain the openlist sorting activity in two ways. First, the sorting only occurs when the size of the openlist doubles,[4] and second, once the sorting function takes more than a certain maximum allocated time (e.g., one second), it is never performed again. This has the net effect of sorting the openlist more often early in the search, when focus is particularly important, and less often or never as the space is unrolled farther and probability information becomes both less discriminatory (because the probability mass is distributed over a very large set of reachable edge nodes) and focus becomes less critical (because the agent will refine its model on the fly).

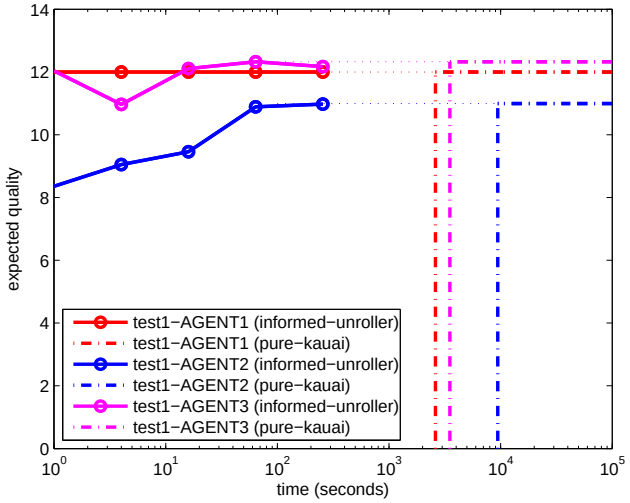


Figure 2: The Informed Unroller can find near-optimal policies much faster than building the complete MDP.

The informed unroller work is at an early stage, but early results from the evaluation against a complete solution are promising. For example, in Figure 2 we show a comparison of the performance of the informed unroller against the complete unrolling process. In these small test problems, the informed unroller is able to find a high-quality policy quickly and to return increasingly effective policies given more time. This allows the IU-agent to flexibly trade off the quality and timeliness of its policies. The current version of the IU does not support repeated, incremental unrolling of the state space during execution. However, we are actively working to build a new version, and integrate it into our CoOrdinators agent.

Related Techniques

The IU approach is related to the "approximate dynamic programming" algorithms discussed in the control theory and operations research literature (Bertsekas 2005). These approaches derive approximate solutions to MDP-type problems by estimating, in various ways, the "cost to go" in leaf nodes of a limited-horizon portion of the full state space. While our exploration of the literature is not yet complete,

[3] Actually, if a child node can enable some other local or non-local method it may still have utility. This sort of effect makes it quite challenging to accurately assess dominance.

[4] This threshold could, of course, be tailored.

initially we believe that a key difference in our IU approach is the notion of time-dependent horizon control and unrolling-guidance (vs. just estimation of leaf-node reward for policy derivation).

The IU method is also somewhat similar to LAO* (Hansen & Zilberstein 2001), which uses knowledge of the initial state(s) and heuristics to generate a state subspace from which the optimal policy can be provably derived. Our technique differs in substantial ways. The IU executes online, and might lack enough time to enumerate such a state subspace even if it knew exactly which states to include. The IU is an anytime algorithm, unlike LAO*, which runs offline. For this reason, the IU makes no claims about policy optimality; indeed, it is not even guaranteed to generate a closed policy. LAO* trims the state space by using an admissible heuristic. No such heuristic is available to the IU because estimating the final quality of a C-TÆMS network, given the various QAFs, is so difficult. However, because the IU is an online algorithm, it can be re-employed periodically during execution, so the policy created can be iteratively tailored to better fit the circumstances.

Coordination

When we consider multiple COORDINATOR agents, the problem expands to finding an optimal *joint policy*. This problem is challenging because:

- The number of possible local policies for agents is in general very large, so the product space of joint policies to search through can be astronomical.
- The size and distribution of the problem makes reasoning about the global behavior of the system impossible.

To address these practical limitations on solving the problem of finding optimal joint policies, our coordination approach is designed to take advantage of three assumptions:

- Inter-agent effects are well-defined and visible to individual agents through their subjective views.
- The agents are given reasonable *initial* schedules.
- The sum of the local expected qualities of the agents is a sufficient approximation of the global expected quality for the problem with respect to guiding search toward improved joint policies.

Our coordination method exploits the first assumption by mining an agent's local subjective view to detect the possible "coordination opportunities." For example, if one of agent-A's methods enables a method for agent-B, that represents a coordination opportunity. Given these discrete opportunities, we can reduce the overall problem from an enormous search over the space of joint policies to a merely huge search over the space of alternative commitments on coordination opportunities.

We exploit the second assumption by extracting default initial coordination decisions from the nominal initial schedule of activities. After ensuring that the agents have coherent and consistent expectations about those commitments, they can then search for approximately-optimal local policies where the commitments for their coordination opportunities are enforced, as described later.

The third assumption will enable negotiating agents to compare different sets of commitments. For an individual agent, a commitment set is preferred to another if it enables the creation of a local policy that has a higher expected reward. A group of agents will prefer a particular combination of local commitment sets if the sum of the expected qualities of their resulting local policies is higher. Note that some agents in the group might have lower expected qualities, but if these are more than compensated by the expected quality gains of other agents then the combination of local commitment sets is considered superior.

There are several ways in which these assumptions and our solution approach (coordination over commitments) may result in sub-optimal behavior. For example, the actual optimal policy set may not adhere to a static set of commitments: e.g., to behave optimally, agents may have to adjust which enablements they will accomplish depending on how prior methods execute. To mitigate this weakness, we plan to have our agents deliberating and negotiating continually, so that they can manage and adapt their commitment set and policies on the fly as methods execute.

Perhaps worse, the third assumption may be violated: an agent's subjective view may not give an accurate estimate of global quality. This problem arises because of the non-monotonic and non-additive effects of different QAFs. For example, suppose three agents have methods under a common :SYNCSUM parent, and in the initial schedule, two of them have agreed to execute their methods at time 10, while the third agent has decided not to execute his method. Now, suppose the third agent realizes it can execute its method at time 7. If it chooses to do so, its local C-TÆMS model may indicate that this change can produce increased local

quality and therefore looks like a good idea. But if the other agents do not adjust their plans, then their methods running at time 10 will be rendered useless (failed, due to the :SYNCSUM semantics emulating the lack of surprise). Thus the third agent may inadvertently reduce the overall team quality by pursuing its local goals without proper consideration of global effects in the task model. This example is fairly easy to recognize, but in general such effects can propagate across multiple NLE links, making them very hard to recognize and manage. Further study is required to determine how badly this assumption affects the performance of our COORDINATOR agent societies in widely varied problems.

Initial Commitments

The agents begin their coordination by identifying their potential interactions (coordination opportunities) and initializing their planned responses (commitments) to those interactions. Each agent does this based on its subjective view of the problem and on its corresponding initial schedule.

The subjective view includes information on the agent's own tasks, as well as some limited information on the tasks of other agents with which the agent may need to coordinate. Specifically, the agent will know about the existence, though not the details, of other agents' tasks that are connected to local tasks through *non-local effects*, for example tasks that another agent must execute before some local task can start. Several types of *coordination opportunities* are extracted from the agent's subjective view:

- *NLE* coordination opportunities are indicated by NLEs across multiple agents' tasks.
- *Synchronization* coordination opportunities are indicated by :SYNCSUM tasks in the subjective view.
- *Redundant Task* coordination opportunities are indicated by tasks that are visible to other agents, indicating that they too have children under that node. Depending on the QAF, such shared tasks may indicate that only one agent should execute methods below the task.

Initial Commitments

The initial schedule is used to infer a set of provisional commitments to the coordination opportunities. For example, if agent-A has a method M_A that enables agent-B's method M_B, then both agents recognize an NLE coordination opportunity. Each agent inspects its initial schedule to see which methods are initially scheduled. For example, agent-A may find that its initial schedule suggests executing M_A at time 3, with an anticipated finish at time 9, and agent-B's initial schedule may suggest executing M_B at time 11. Agent-A would form a tentative commitment to finish M_A before 9, and agent-B would form a tentative commitment expecting its enablement before time 11. Note that the agents' initial schedules may have flaws or poorly-aligned methods, so these initial commitments may not be entirely consistent.

Distributed Constraint Satisfaction

In our problem formulation, negotiation can be viewed as a search over the space of compatible commitments that agents make to their coordination opportunities, seeking a set of commitments that satisfactorily achieves high global expected quality (as estimated by the sum of local expected qualities). The search begins with the initial set of commitments, and then tries to improve on this initial set (or some previous improvement of that set) to increase the compatibility and/or quality of the commitments.

Our agents begin by ensuring compatibility among the commitments, essentially treating the problem as a distributed constraint satisfaction problem (DCSP). For example, an agent acting as the source for an enablement NLE should be committed to completing the enabling task before the target agent expects to begin the enabled task. If their locally-generated commitments do not satisfy this constraint, then the agents need to resolve the inconsistency.

As a simple first step, our agents exchange their initial commitments with the relevant partner agents and, in a one-shot computation, modify the commitments to ensure consistency. Since each agent applies the same process to the same information to establish consistency, each agent involved in a coordination opportunity will arrive at the same resolution to any inconsistency. Note that, even within this simple protocol, there are several possible approaches. For example, if agents disagree on the time an enablement will occur, the agreed-upon time could be the earlier time, later time, or some intermediate value.

Enforcing Commitments

Given a consistent set of commitments, we need a method to *enforce* those commitments in the agents' policy-finding process. One approach would be to re-write the task model so that the MDP unroller does not even consider the possibility of violating the commitment. This approach is too

restrictive because it would not permit the agents to reason flexibly about off-nominal method outcomes. In some situations it may be a rational decision to *not* satisfy a given commitment. Therefore, rather than rigidly enforce commitments, we strongly bias the policies generated by the MDP solver towards the committed behaviors. This can be done by providing an extra reward for states in which a given commitment is satisfied and an extra penalty for states in which it can be determined that the commitment will not be satisfied. We also add local "proxy methods" modeling the execution of other agents' tasks.

We simplify the negotiation process to reduce the complexity of enforcement. For example, the execution of tasks by other agents is currently modeled deterministically: the commitment agreement includes a single time by which the task is to be executed. A more expressive representation would accommodate completion time distributions.

Conclusion and Future Directions

Our agents are in the early stages of their design evolution, having "played the game" in simulation for only a short while, during which time the simulation environment itself has also evolved. In their current form, the agents recognize coordination opportunities, derive initial commitments from the initial schedule, communicate to ensure consistency among commitments, and then unroll the resulting local MDP until their first method's release time arrives. They then execute their MDP policy whenever possible. If the MDP unrolling expands the entire reachable state space, they should remain "on-policy" and perform according to the MDP's predictions unless there is an unmodeled failure or new task arrival. More often, the unrolling is incomplete and the agents can "fall off-policy" when the simulated world state reaches an unexpanded part of the MDP state space. At that point, the agents begin executing one of two simple reactive scheduling algorithms (one driven off the initial schedule, one based on a one-step lookahead).

Initial performance results indicate that, given a modest amount of time to unroll their MDPs (e.g., twenty seconds), the agents can explore tens of thousands of MDP states and sometimes dramatically outperform the simple reactive schemes alone. However, there is clearly room for major improvement, and the simple reactive approaches likewise sometimes dramatically outperform our initial IU system. As our system evolves to fulfill its full design, we will conduct more detailed experimentation and analysis.

We believe that we now have a strong foundation for principled future work on coordinated plan and schedule management in uncertain domains. We have many additional features to develop, including incremental and continuous MDP unrolling (so the agents continue to think about the problem as they are executing their existing policy) and improved negotiation that will lead to more dramatic changes in the initial schedule and inter-agent commitments.

Acknowledgments

This material is based upon work supported by the DARPA/IPTO COORDINATORS program and the Air Force Research Laboratory under Contract No. FA8750–05–C–0030. Any opinions, findings and conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA, the U.S. Government, or the Air Force Research Laboratory.

References

Bertsekas, D. P. 2005. Dynamic programming and suboptimal control: A survey from ADP to MPC. In *Proc. Conf. on Decision and Control*.

Boddy, M.; Horling, B.; Phelps, J.; Goldman, R. P.; and Vincent, R. 2005. C-TÆMS language specification. Unpublished; available from this paper's authors.

Hansen, E. A., and Zilberstein, S. 2001. LAO: a heuristic search algorithm that finds solutions with loops. *Artificial Intelligence* 129(1-2):35–62.

Horling, B.; Lesser, V.; Vincent, R.; Wagner, T.; Raja, A.; Zhang, S.; Decker, K.; and Garvey, A. 1999. The TAEMS white paper. Technical report, University of Massachussetts, Amherst, Computer Science Department.

Miller, C. A.; Goldman, R. P.; and Funk, H. B. 2003. A Markov decision process approach to human/machine function allocation in optionally piloted vehicles. In *Proceedings of FORUM 59, the Annual Meeting of the American Helicopter Society*.

Puterman, M. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons.

Wagner, T. A.; Garvey, A. J.; and Lesser, V. R. 1998. Criteria Directed Task Scheduling. *Journal for Approximate Reasoning* 19:91–118.

Compiling Uncertainty Away: Solving Conformant Planning Problems Using a Classical Planner (Sometimes)

Héctor Palacios
Departamento de Tecnología
Universitat Pompeu Fabra
08003 Barcelona, SPAIN
hector.palacios@upf.edu

Héctor Geffner
Departamento de Tecnología
ICREA & Universitat Pompeu Fabra
08003 Barcelona, SPAIN
hector.geffner@upf.edu

Abstract

Even under polynomial restrictions on plan length, conformant planning remains a very hard computational problem as plan verification itself can take exponential time. This heavy price cannot be avoided in general although in many cases conformant plans are verifiable efficiently by means of simple forms of disjunctive inference. This raises the question of whether it is possible to identify and use such forms of inference for developing an efficient but incomplete planner capable of solving non-trivial problems quickly. In this work, we show that this is possible by mapping conformant into classical problems that are then solved by an off-the-shelf classical planner. The formulation is sound as the classical plans obtained are all conformant, but it is incomplete as the inverse relation does not always hold. The translation accommodates 'reasoning by cases' by means of an 'split-protect-and-merge' strategy; namely, atoms L/X_i that represent conditional beliefs 'if X_i then L' are introduced in the classical encoding that are combined by suitable actions when certain invariants are verified. Empirical results over a wide variety of problems illustrate the power of the approach.

Introduction

Conformant planning is a form of planning where a goal is to be achieved when the initial situation is not fully known and actions may have non-deterministic effects (Goldman & Boddy 1996). Conformant planning is computationally harder than classical planning, as even under polynomial restrictions on plan length, plan verification remains hard (Haslum & Jonsson 1999; Baral, Kreinovich, & Trejo 2000; Turner 2002; Rintanten 2004). This additional complexity cannot be avoided in general, although often conformant plan verification can be done efficiently by means of simple forms of disjunctive inference.

For example, simple rules suffice to show that a robot that systematically scans a grid, collecting the objects in each of the cells, will pick up all the objects in the grid, regardless of their original locations. Or similarly, that a robot that moves n times to the right in an empty grid of size n, will necessarily end up in the rightmost column.

This raises the question of whether it is possible to identify and use such forms of inference for developing an efficient but incomplete conformant planner capable of solving non-trivial problems and scaling up as well as the best classical planners. In this work, we show that this is possible by formulating a suitable translation of conformant problems into classical problems which are then solved by an off-the-shelf classical planner. The translation is sound as the classical plans are all conformant, but it is incomplete as the converse does not always hold. The translation scheme accommodates 'reasoning by cases' by means of a 'split-protect-and-merge' strategy; namely, atoms L/X_i that represent conditional beliefs 'if X_i then L' are introduced in the classical encoding that are then combined by suitable actions when certain invariants in the plan are verified.

As an illustration, assuming that there is a $pickup(l)$ action with precondition $at(l)$ and effect 'if $at(o,l)$ then $hold(o)$' with $at(o,l)$ unknown, the translation introduces an effect of the form 'if **true** then $hold(o)/at(o,l)$' whose consequent is an atom $hold(o)/at(o,l)$ that stands for the conditional belief 'if $at(o,l)$ is true, then $hold(o)$ is true'. Then any classical plan that achieves the atoms $hold(o)/at(o,l)$ for each one of the possible locations l of o, and which preserves certain invariants (like that the 'hidden' locations do not change), can be shown to be a valid conformant plan for achieving $hold(o)$.

Actually, by formulating the underlying principles for this type of reasoning and generalizing it in accordance with a *clear semantics*, we will see that many other patterns of inference fall into place. For example, if in this same example we add 'push' actions that move objects from a cell to a neighboring cell (for each one of the possible directions), and at the same time, restrict the pick up actions to particular cells only (like corners or centers), then the classical encoding would produce valid conformant plans where enough pushes are done so that all objects are forced into such cells *regardless of their original location*, from which they are then collected. While several effective but incomplete formulations of conformant planning have been formulated before (some of which handle sensing as well; see (Baral & Son 1997; Petrick & Bacchus 2002)), none, as far as we know, can represent these types of plans.

The explicit representation of conditional beliefs in the form of 'tagged' atoms like $hold(o)/at(o,l)$ has some resemblance to the use of observation and contingency labels in (Peot & Smith 1992; Pryor & Collins 1996), although the use of such tags is different.

In this paper, we will look first at some proposals that tradeoff expressivity for efficiency, present then the proposed translation scheme, look at some empirical results, and discuss finally current limitations.

Taming Complexity

The problem of conformant planning can be formulated as a deterministic search problem in belief space, where a sequence of actions that map a given initial belief state bel_0 into a target set of beliefs is sought. A belief state bel represents the set of states s that are deemed possible, and actions a, whether deterministic or not, deterministically map one belief state bel into another, denoted as bel_a (Bonet & Geffner 2000). Since the number of belief states is exponential in the number of states, it is clear that the search for conformant plans takes place in a space that is exponentially larger than the search for classical plans. Indeed, while under polynomial length restrictions, classical plan existence is NP-Complete (Bylander 1994), under the same conditions conformant plan existence is harder, Σ_2^P (Turner 2002). This is because plan verification is 'easy' in the classical setting but 'hard' in the conformant one, as the verification requires to evaluate the plan for every initial state and transition.

This difference in complexity explains why it is still very easy to come up with simple and small conformant problems that no general domain-independent planner can solve, while the same is no longer true for classical planners. The main motivation of this work is to narrow this gap by developing an approach that targets 'simple' conformant problems effectively. The approach will not be complete but it will provide solutions to non-trivial problems, where it will scale up as well as classical planners do.

One way to trade off completeness for efficiency in conformant planning results from approximating belief states or transitions. For example, the 0-approximation introduced in (Baral & Son 1997) represents belief states bel by means of two sets: the set of literals that are true in bel, and the set of literals that are false in bel. Variables which do not appear in either set are unknown. In this representation, checking whether an action a is applicable in bel, computing the next belief state bel_a, and verifying polynomial length plans are all polynomial time operations. Roughly, a fluent literal L makes it into bel_a iff a) action a has some conditional effect $C \rightarrow L$ such that all literals in C are in bel, or b) L is in bel and for all conditional effects $C' \rightarrow \neg L$ of action a, the complement of some literal $L' \in C'$ is in bel.

Conformant planning under the 0-approximation is thus no more complex, theoretically, than classical planning. The problem however is that the 0-approximation is strongly incomplete, as it does not capture any non-trivial form of disjunctive inference. For example, given a disjunction $p \vee q$ and an action a that maps either p or q into r, the semantics will not validate a as a conformant plan for r. Indeed, disjunctions that are not tautologies are thrown away. The 0-semantics does capture, on the other hand, situations in which the information that is missing is not relevant. For example, if there are actions that can make a variable p true or false, then uncertainty in the initial state of p would not hurt. Classical planners, on the other hand, cannot handle such situations.

Another sound but incomplete approach to planning with incomplete information is advanced in (Petrick & Bacchus 2002) where belief states bel are represented by more complex formulas which may include disjunctions. Yet in order to make belief updates efficient several approximations are introduced, and in particular, while existing disjunctions can be carried from one belief to the next and can be simplified, no new disjunctions are added. This too imposes a serious limitation in the type of problems that can be handled.

Expressivity, however, is not the only problem; efficiency or control is the other. Indeed, it is not enough to introduce restrictions that under polynomial length constraints bring the complexity of conformant planning to that of classical planning or SAT; the control knowledge needed for solving the resulting problem must be made available as well. The approach in (Petrick & Bacchus 2002) leave this problem largely unaddressed relying on a blind search over compact belief representations and efficient update rules. Recent elaborations of the 0-approximation in (Son, Tu, & M. Gelfond 2005) rely in turn on a fixed heuristic function that counts the number of goals achieved, which applies well to some problems but not to others. In this work, we aim to address both problems, expressivity and control, by accounting for certain forms of disjunctive reasoning in a translation scheme that maps conformant problems into classical problems that can be handled efficiently.

Basic Translation

The translation scheme maps a conformant planning problems P into a classical planning problems $K(P)$. We describe the contents of $K(P)$ in three parts, starting with the basic core $K_0(P)$.

We assume that P is given by tuples of the form $\langle F, O, I, G\rangle$ where F stands for the fluent symbols in the problem, O stands for a set of actions a, I is a set of clauses over F defining the initial situation, and G is a set of literals over F defining the goal. In addition, every action a has a precondition given by a set of fluent literals, and a set of conditional effects $C \rightarrow L$ where C is a set of fluent literals and L is a literal. We assume that actions are all *deterministic* and hence that all uncertainty lies in the initial situation.

We will usually refer to the conditional effects $C \rightarrow L$ of an action a as the *rules* associated with a, and sometimes write them as $a : C \rightarrow L$. Also, we use the expression $C \wedge X \rightarrow L$ to refer to rules with literal X in their bodies. In both cases, C may be empty. Last, when L is a literal, we take $\neg L$ to denote the complement of L.

Definition 1 (Core Translation) *The* core translation *maps the conformant problem P into the classical problem $K_0(P) = \langle F', O', I', G'\rangle$ where*

- $F' = \{KL, K\neg L \mid L \in F\}$
- $I' = \{KL, \neg K\neg L \mid L \in I\} \cup \{\neg KL', \neg K\neg L' \mid L' \notin I\}$
- $G' = \{KL \mid L \in G\}$
- $O' = O$ *but with each literal precondition L for $a \in O$ replaced by KL, and each conditional effect $a : C \rightarrow L$ replaced by $a : KC \rightarrow KL$ and $a : \neg K\neg C \rightarrow \neg K\neg L$.*

For any literal L in P, KL denotes its 'epistemic' counterpart in $K_0(P)$ whose meaning is that L is known. We write KC for $C = L_1 \wedge L_2 \ldots$ as an abbreviation for $KL_1 \wedge KL_2 \ldots$, and $\neg K\neg C$ for $\neg K\neg L_1 \wedge \neg K\neg L_2 \ldots$.[1] $L \in I$ ($L \notin I$) means that literal L is (not) a unit clause in I.

The intuition behind the translation is simple: first, complementary literals L and $\neg L$ whose status is not known in the initial situation in P are 'negated', by mapping them into the negated K-literals $\neg KL$ and $\neg K\neg L$ that are jointly consistent and can only appear in the body of conditions leading to other negated K-literals only. This mapping removes all uncertainty from $K_0(P)$. In addition, to ensure soundness, each conditional effect $a : C \rightarrow L$ in P maps, not only into the 'supporting' rule $a : KC \rightarrow KL$ but also into the 'cancellation' rule $a : \neg K\neg C \rightarrow \neg K\neg L$ that guarantees that literal $K\neg L$ is deleted (prevented to persist) when action a is applied except when C is known to be false. The soundness of the translation can be then expressed as:

Theorem 2 (Soundness $K_0(P)$) *If π is a plan that solves the classical planning problem $K_0(P)$, then π is a plan that solves the conformant planning problem P.*

This can be proved through the following lemma that formally captures the meaning of K-literals:

Lemma 3 (Meaning K-literals) *If π is a plan that yields the literal KL in $K_0(P)$, then π is a plan that yields the literal L with certainty in P.*

A plan π here is an applicable action sequence, and π yields a formula if the formula is necessarily true upon completion of the plan. The reader can verify that this translation is in agreement with the 0-approximation semantics (Baral & Son 1997):

Theorem 4 (Equivalence $K_0(P)$ and 0-Approximation) *π solves the classical planning problem $K_0(P)$ iff it solves the conformant planning problem P according to the 0-Approximation.*

This correspondence is not surprising as both formulations throw away the disjunctive information, and restrict the valid plans to those that render the missing information irrelevant. Also, the states s_0^k, s_1^k, ... generated by the action sequence $\pi = a_0, a_1, \ldots$ over the classical encoding $K_0(P)$ encode precisely the literals that are known according to the 0-approximation; namely, L is known at time t according to the 0-approximation iff the literal KL is true in s_t^k.

As an illustration, given a conformant problem P with $I = \{p, r\}$ (i.e., nothing else is known; there is no CWA), and actions a and b with effects $a : p \rightarrow q$, $a : r \rightarrow \neg s$, and $b : q \rightarrow s$, the plan $\{a, b\}$ is valid for achieving q and s according to both $K_0(P)$ and the 0-approximation, while the singleton sequence $\{a\}$ is not valid according to either. At the same time, if the initial situation is changed to $I = \{p \vee q\}$, neither approach would sanction that plan $\{a\}$ for q, even if it is a valid conformant plan. For this, some ability to reason with disjunctions is needed.

[1] Taking $\neg K\neg C$ as an abbreviation for $\neg K\neg L_1 \wedge \neg K\neg L_2$ when $C = L_1 \wedge L_2$ means that we take C to be known as false only when one of the literals in C is known to be false. In modal logics, this is correct but not required; C may be known to be false even when no literal in C is; see (Fagin *et al.* 1995).

Case Analysis over Single Actions

We will make the formulation stronger by accounting for certain disjunctive inferences in the translation. This will result into more actions and conditional effects added to $K(P)$ which is initially set to $K_0(P)$.

Consider an action a that in a given context C' can force a literal L to make the transition from false to true, while preventing the opposite transition. In such a context C', even if L is unknown, a can be used to make L true. This type of inference is captured in the translation as follows:

Rule 2 (Action Compilation) *If P contains a rule $a : C \wedge \neg L \rightarrow L$, and the rules for the same action a with $\neg L$ in the head are $C_i \rightarrow \neg L$, $i = 1, \ldots, n$ for $n \geq 0$, then add to $K(P)$ the rules $KC \wedge K\neg L_1 \wedge \cdots \wedge K\neg L_n \rightarrow KL$ where L_i is a literal in C_i.*

This is a modular translation rule in which the context C' above is the formula $C \wedge \neg L_1 \wedge \cdots \wedge \neg L_n$, for any combination of literals L_i chosen as to preempt the rules $C_i \rightarrow \neg L$ associated with the same action a that can clobber L. All the literals in C' are preceded by K's as they refer to literals in $K(P)$ that ensure that the condition holds with certainty. This translation remains polynomial as long as the number of rules $a : C_i \rightarrow L$ associated with the same action a and literal L and the number of literals in the conditions C_i remain both bounded, which is normally the case (in the existing benchmarks indeed both numbers are pretty small, one or two at most).

It is not difficult to show that this translation rule preserves soundness. A key characteristic of the rule and others to be introduced below is that they make use of the conditional effects $a : C \wedge X \rightarrow L$ in the problem P for deriving L with certainty when the body $C \wedge X$ is not fully known.

In an example like 'empty room', where a robot moves in an empty square grid and literals X_i are used to represent the column location of the robot, this translation ensures that literal $K\neg X_1$ is obtained right after a single 'move right' action (namely, that the robot cannot be in the leftmost column then), and similarly, that $K\neg X_2$ is obtained after two consecutive right moves, etc. If the grid is nxn, the resulting classical theory yields $K\neg X_i$ for $i < n$ after $n - 1$ steps, although it does not yield KX_n (being in the rightmost column). For this, the disjunction expressing the possible column positions, namely $X_1 \vee X_2 \vee \cdots \vee X_n$, needs to be taken into account as well. We address this next.

Case Analysis over Action Sequences

We extend the translation further so that the disjunctions in P are taken into account in a form that is similar to the Disjunction Elimination inference rule used in Logic (Barwise & Etchemendy 1991):

$$\text{If } X_1 \vee \cdots \vee X_n,\ X_1 \supset L, \ldots, \text{and } X_n \supset L \text{ then } L \quad (1)$$

For this, we create new atoms in $K(P)$, written L/X_i, that aim to capture the conditional beliefs $X_i \supset L$. Then, the resulting classical encoding will be such that once these atoms

are 'achieved' for each $i = 1, \ldots, n$, and when they are suitably 'protected', the literal L will be made 'achievable' by an extra 'dummy' action with conditional effect similar to (1).

As already mentioned, the atoms L/X_i will stand for the conditional belief 'if X_i then L'. In principle, any rule $a : C \wedge X_i \rightarrow L$ in P with X_i uncertain can be used to produce a rule $a : KC \rightarrow L/X_i$ in $K(P)$, meaning that if KC is known and a is applied, then if X_i was true, L will become true. However, we want L/X_i to mean exactly that '*right after the action* a, if X_i is true, then L is true', and for this, some additional care is needed. Indeed, if a contains also rules $a : C_k \rightarrow X_i$ that can make X_i true, it may be possible that L and X_i are false at time t when a is applied, and that L remains false but X_i becomes true, and then that 'if X_i at t, then L at $t + 1$' is true, but 'if X_i at $t + 1$, then L at $t + 1$' is false. In order to rule out this situation we define the corresponding translation rule as follows:

Rule 3 (Split) *For each rule $a : C \wedge X_i \rightarrow L$ in P where X_i is a literal that appears in a disjunction $X : X_1 \vee X_2 \vee \cdots \vee X_n$, if $a : C_k \rightarrow X_i$, $k = 1, \ldots, m$ for $m \geq 0$ are the rules in P for the same action a with X_i in the head, then add to $K(P)$ the atoms L/X_j, $j = 1, \ldots, n$, all initialized to **false**, and the rules $a : KC \wedge K\neg L_1 \wedge \cdots \wedge K\neg L_m \rightarrow L/X_i$ where L_k is a literal in C_k.*

The combinations of the conditional beliefs represented by the atoms L/X_i is achieved by means of extra actions added to the classical encoding $K(P)$ that generalize (1) slightly, allowing some of the cases X_i to be disproved:[2]

Rule 4 (Merge) *For each disjunction $X : X_1 \vee \cdots X_n$ and atom L in P such that L/X_i is an atom in $K(P)$, add to $K(P)$ a new action $a_{X,L}$ with conditional effect*

$$(L/X_1 \vee K\neg X_1) \wedge \cdots \wedge (L/X_n \vee K\neg X_n) \wedge FLAG_{X,L} \rightarrow L$$

*where $FLAG_{X,L}$ is a boolean initialized to **true**. If $L = X_i$ for some $i \in [1, n]$, remove the conjunct $(L/X_i \vee K\neg X_i)$ from the rule body.*

A key distinction from Logic is that the disjunction $X_1 \vee \cdots \vee X_n$ and the conditional beliefs 'if X_i then L' represented by the atoms L/X_i need all be **preserved** until they are combined together to yield L. This is the purpose of the boolean $FLAG_{X,L}$ that is initially set to true, but which is deleted when an action is done in a context where it is not possible to prove that 1) L is preserved (if true), 2) the disjunction $X \vee L$ is preserved (the disjunction X is initially true but it is actually sufficient to preserve the weaker disjunction $X \vee L$), and 3) the conditional beliefs represented by the atoms L/X_i achieved are preserved. This is accomplished by extending $K(P)$ with the following cancellation rules:

Rule 5 (Protect) *If there is a boolean flag $FLAG_{X,L}$ in $K(P)$ for $X : X_1 \vee \cdots \vee X_n$, then for each action a: 1) if $a : C \rightarrow \neg L$ in P, add to $K(P)$ the rule $a : \neg K\neg C \rightarrow \neg FLAG_{X,L}$, 2) if $a : C \rightarrow \neg X_i$ in P and neither $a : C \rightarrow X_k$ nor $a : C \rightarrow L$ in P for X_i and X_k in X, add to $K(P)$ the rule $a : \neg K\neg C \rightarrow \neg FLAG_{X,L}$, and 3) if $a : C \rightarrow X_k$ for X_k in X, then add to $K(P)$ the rule $a : \neg K\neg C \wedge L/X_k \rightarrow \neg FLAG_{X,L}$.*

These rules, as we will see, yield expressivity without sacrificing efficiency, as they manage to *accommodate nontrivial forms of disjunctive inference in a classical theory without having to carry disjunctive information explicitly in the belief state:* disjunctive information is represented in terms of the conditional atoms L/X_i and the *invariants* that are enforced in the encoding.

Theorem 5 (Soundness $K(P)$) *Any plan that achieves the literal KL in $K(P)$ is a plan that achieves L in the conformant problem P.*

The key element in the proof is the following lemma that captures the meaning of the L/X_i atoms:

Lemma 6 (L/X_i Atoms) *Any plan that yields L/X_i while preserving $FLAG_{X,L}$ in $K(P)$ is a plan that achieves the conditional $X_i \supset L$ in P.*

A proof sketch goes as follows. Let us assume that L/X_i, which is initially false, is made true at time t by an action a in the plan. We need to prove that if $FLAG_{X,L}$ remains true in $K(P)$ until time $t' \geq t$, then the conditional $X_i \supset L$ remains true until t' in P, which we write as $X_i(t') \supset L(t')$. From the argument above, if L/X_i became true in $K(P)$ at time t, so does the conditional $X_i(t) \supset L(t)$ in P. From this, $X_i(t') \supset L(t')$ follows if we can show both $X_i(t') \supset X_i(t)$ and $L(t) \supset L(t')$. The latter is true because the rules in $K(P)$ ensure that if a rule $a' : C' \rightarrow \neg L$ gets triggered by the plan in P, the rule $a' : \neg K\neg C' \rightarrow \neg FLAG_{X,L}$ will be triggered by the plan in $K(P)$. Similarly, the former is true because the rules in $K(P)$ ensure that if a rule $a' : C' \rightarrow X_i$ is triggered by the plan in P when L/X_i is true in $K(P)$, then the rule $a' : \neg K\neg C' \wedge L/X_i \rightarrow FLAG_{X,L}$ will be triggered in $K(P)$. In either case, $FLAG_{X,L}$ would be deleted, so if it is not, $X_i(t') \supset X_i(t)$ and $L(t) \supset L(t')$ must hold, and since $X_i(t) \supset L(t)$ holds, so must $X_i(t') \supset L(t')$.

As an illustration, given an object to be collected from an unknown location in a grid with two cells A and B using the actions $pick(X)$, $push(X, Y)$, and $go(X, Y)$, where X and Y are cells, and the three actions have as a precondition that the agent is at X, and the first two have the expected effects conditional on the location of the object, it follows that if the agent is initially at A, the plan $\pi_1 = \{pick(A), go(A, B), pick(B)\}$ is valid in $K(P)$ and so is $\pi_2 = \{push(A, B), go(A, B), pick(B)\}$, but $\pi_3 = \{pick(A), go(A, B), push(B, A)\}$ is invalid. If $at(Obj, A) \vee at(Obj, B)$ is the disjunction X, and L is $hold(Obj)$, then π_1 results from achieving the atoms $hold(Obj)/at(Obj, A)$ and $hold(B)/at(Obj, B)$, π_2 from achieving $K\neg at(Obj, A)$ and $hold(B)/at(Obj, B)$, while π_3 achieves $hold(Obj)/at(Obj, A)$ and $K\neg at(Obj, B)$ but clobbers the protection flag $FLAG_{X,L}$ as the rule $push(B, A) : at(Obj, B) \rightarrow at(Obj, A)$ in P, yields the rule $push(B, A) : \neg K\neg at(Obj, B) \wedge L/at(Obj, A) \rightarrow$

[2] When using the classical plans obtained from $K(P)$ as conformant plans in P, such 'dummy' actions must are removed.

	P				$K(P)$		
Problem	#Actions	#Atoms	#Effects	Translation time	#Actions	#Atoms	#Effects
Bomb-100-60	6060	320	24120	1.35	6260	1041	79560
Cube-11-Ctr	6	33	120	0.036	72	226	1152
Sqr-64-Ctr	4	128	504	0.31	260	893	4796
Grid-4-5	174	155	444	5.65	183	351	1244
Safe-100	100	101	100	0.11	101	304	804
Logistics-4-10-10	3320	610	6640	3.52	3370	1321	13880

Table 1: Data concerning the translation of some conformant problems P into classical encodings $K(P)$. The sizes refer to the grounded versions, and all times are in seconds and they include grounding time.

$\neg FLAG_{X,L}$ in $K(P)$ which gets triggered by the action sequence π_3.

Experimental Results

We have implemented the translation scheme into a program `cf2cs` that takes a conformant planning problem P as input an outputs a classical problem $K(P)$. In the experiments below, this problem is fed into the FF v2.3 classical planner (Hoffmann & Nebel 2001). We refer to the resulting conformant planner as `cf2cs(ff)`. The experiments were tested on a Intel/Linux machine running at 2.80GHz with 2Gb, and were cutoff when they took more than 30 minutes or 800Mb of memory.

We report results on two classes of instances: existing benchmarks and some domains of our own. In both cases, we compare the results (times and plan lengths) with those obtained by running Conformant FF, an state-of-the-art conformant planner (Brafman & Hoffmann 2004).[3] We could have used other recent classical and conformant planners, but as a reference, this should do. We want to show that our approach solves a wide variety of non-trivial problems without any 'help' in the encoding or control, scaling up as well as the best classical planners.

Encodings are important. For example, it is common to see a problem like Bomb-in-the-Toilet solved by having the action $dump(pkg)$ with conditional effect $dump(pkg) \rightarrow disarmed(pkg)$ and goals $disarmed(pkg_1)$, ..., $disarmed(pkg_n)$. Such encodings however are not natural (the bomb should be disarmed not the packages) and change the problem significantly. This is indeed the whole idea of our approach that automatically re-codes conformant into classical problems. For the existing benchmarks we thus use the actual encodings from the Conformant FF repository, the other encodings will be made available from us. These encodings cannot be solved by either the basic $K_0(P)$ translation or the 0-approximation.

Table 1 shows data concerning the translation of a number of problems from various sources, used and explained in (Brafman & Hoffmann 2004). Bomb-x-y refers to the Bomb-in-the-toilet problem with x packages, y toilets, and clogging. Cube-n-Ctr refers to the problem of reaching the center of a cube of size n^3 from a completely unknown location. Square-n-Ctr is similar but involves only n^2 possible locations. Logistics-i-j-k, Grid-n and Safe-n are from (Brafman & Hoffmann 2004).

The table provides information about the size of the original (ground) conformant problems P, the resulting classical problems $K(P)$, as well as the time taken in the translation. This last figure is less than a second in most problems, but grows up to a few seconds in some. The mapping is linear in $|P|$ and can be further optimized (grounding the problem P is currently taking most of the time).

Table 2 shows the plan times and lengths obtained by `cf2cs(ff)` vs. Conformant FF over various benchmarks, where it can be seen that `cf2cs(ff)` scales up much better, solving problems like Sqr-240-Ctr and Cube-75-Ctr that are well beyond the reach of current complete or incomplete conformant planners (with the exception of (Cimatti, Roveri, & Bertoli 2004) that reports very good number on many of these domains).

Among the existing benchmarks, not included in the table, there are three domains, Sorting-Nets, (Incomplete) Blocks, and Ring, which in their standard encodings, cannot be handled by the proposed translation scheme; namely, in none of these encodings P the planner finds a classical plan in $K(P)$ even though P has conformant solutions. In Blocks, for example, this is because of the presence of conditional effects $a : C \wedge X_i \wedge Y_j \rightarrow L$ whose bodies involve disjuncts X_i and Y_j from different disjunctions. The translation scheme can be adapted to deal with such cases, even though such extensions would not render the scheme complete either (see the discussion below).

Finally, Table 3 shows plan times and lengths for a family of grid problems that we devised: Retrieve is about retrieving objects whose locations are unknown; Dispose is about retrieving such objects and placing them in a trash can at a given, known location; Push is a variation of Retrieve when there is also a push action that can move objects; and Push-to is a further variation where the retrieve action is applicable only at two known corners of the grid, and therefore, objects need to be pushed into those locations. Problem P-n-m stands for problem P over grid of size n and m objects. Once again the Table shows a different scaling behavior between `cf2cs(ff)` and Conformant FF except in Push-To, the reason being that the resulting classical encoding $K(P)$ of that domain has many dead-ends that are not detected by the heuristic used in FF. This problem could be solved in principle by refining the FF heuristic or the translation, and we would like to explore both alternatives in the future.

[3] Conformant FF uses the overall architecture of FF but not the actual planner; see (Brafman & Hoffmann 2004) for details.

Problem	cf2cs(ff) Time	cf2cs(ff) Length	CFF Time	CFF Length
Bomb-50-50	2.13	50	0.2	50
Bomb-100-1	0.84	199	96.2	199
Bomb-100-60	9.64	140	23.53	140
Cube-7-Ctr	0.02	24	38.2	39
Cube-9-Ctr	0.05	33	—-	—-
Cube-11-Ctr	0.09	42	—-	—-
Cube-75-Ctr	484	330	—-	—-
Sqr-8-Ctr	0.03	22	140.5	50
Sqr-12-Ctr	0.04	32	—-	—-
Sqr-64-Ctr	9.66	188	—-	—-
Sqr-120-Ctr	59.44	356	—-	—-
Sqr-240-Ctr	858	716	—-	—-
Grid-4-4	0.06	25	0.11	25
Grid-4-5	0.05	30	0.14	30
Safe-30	0.01	30	6.6	30
Safe-70	0.08	70	561.8	70
Safe-100	0.28	100	—-	—-
Logistics-3-10-10	4.42	109	11.15	108
Logistics-4-10-10	5.91	125	11.74	121

Table 2: Plan times and lengths obtained by a classical planner (FF) over $K(P)$ translation (`cf2cs(ff)`) in relation to Conformant FF for various conformant problems P. Times in seconds. The symbol '—-' means cutoff exceeded (30 mins or 800Mb)

Problem	cf2cs(ff) Time	cf2cs(ff) Length	CFF Time	CFF Length
Retrieve-4-1	0.02	41	0.04	33
Retrieve-4-2	0.08	75	0.23	49
Retrieve-4-3	0.17	91	0.8	65
Retrieve-8-1	1.18	220	204.68	210
Retrieve-8-2	3.13	291	—-	—-
Retrieve-8-3	132.49	415	—-	—-
Retrieve-12-1	—-	—-	—-	—-
Dispose-4-1	0.02	37	0.12	39
Dispose-4-2	0.05	54	0.47	56
Dispose-4-3	0.09	71	1.49	73
Dispose-8-1	1.83	265	361	227
Dispose-8-2	2.87	280	—-	—-
Dispose-8-3	6.87	367	—-	—-
Dispose-12-1	—-	—-	—-	—-
Push-4-1	0.07	41	0.09	33
Push-4-2	0.24	75	0.41	49
Push-4-3	0.53	91	1.23	65
Push-8-1	3.29	220	—-	—-
Push-8-2	12.89	291	—-	—-
Push-8-3	—-	—-	—-	—-
Push-to-3-1	0.32	21	0.03	29
Push-to-4-1	—-	—-	0.48	46
Push-to-4-2	—-	—-	5.01	77
Push-to-4-3	—-	—-	228.84	114

Table 3: Plan times and lengths obtained by a classical planner (FF) over $K(P)$ translation (`cf2cs(ff)`) in relation to Conformant FF for various problems P. Times in seconds. The symbol '—-' means cutoff exceeded (30 mins or 800Mb).

Discussion

We have introduced a translation scheme that enables a wide class of conformant planning problems to be solved by an off-the-shelf classical planner. The translation accounts for a limited form of 'reasoning by cases' by means of an 'split-protect-and-merge' strategy; namely, atoms L/X_i that represent conditional beliefs 'if X_i then L' are introduced, and when certain invariants are verified, they are combined. Empirical results over a variety of problems illustrate the power of the approach.

The simplicity of the translation and the semantics captured by the theorems not only enable us to prove the soundness of the approach, but as importantly, to delimit its scope. In relation to natural deduction systems in the style of Ficht (Barwise & Etchemendy 1991), the type of disjunctive reasoning accounted for in the translation is limited in two ways. First, while disjunctions $X_1 \vee \cdots \vee X_n$ in P are used to create sub-derivations by making assumptions of the form X_i, these sub-derivations are not nested, and therefore, disjunctions are not combined. Second, the sub-derivations that arise when making the assumptions X_i are very limited; in particular, the atoms L/X_i can only be used for proving L and nothing else. The translation captures plans that can be verified by reasoning with 'one disjunction' at a time. Verification that involve reasoning that combines all disjunctions is intractable, yet verifications that use a bounded number of disjunctions N at a time are tractable, and could be accommodated in a polynomial translation scheme as the one proposed (although it may not be effective for large N).

Acknowledgements

We thank Joerg Hoffmann for making CFF and the source of FF available, and for the encoding of the problems. Thanks also to Malte Helmert for providing the Fast-Downward planner and useful discussion. We also thank A. Frangi and A. Sanz for the use of the Hermes Computing Resource at the Aragon Inst. of Engr. Research (I3A), U. of Zaragoza. H. Geffner is partially supported by grant TIN2005-09312-C03-03 from MEC/Spain.

References

Baral, C., and Son, T. C. 1997. Approximate reasoning about actions in presence of sensing and incomplete information. In *Proc. ILPS 1997*, 387–401.

Baral, C.; Kreinovich, V.; and Trejo, R. 2000. Computational complexity of planning and approximate planning in the presence of incompleteness. *Artificial Intelligence* 122(1-2):241–267.

Barwise, J., and Etchemendy, J. 1991. *The Language of First-Order Logic*. CSLI, Stanford.

Bonet, B., and Geffner, H. 2000. Planning with incomplete information as heuristic search in belief space. In *Proc. of AIPS-2000*, 52–61. AAAI Press.

Brafman, R., and Hoffmann, J. 2004. Conformant planning via heuristic forward search: A new approach. In *Proc. ICAPS-04*.

Bylander, T. 1994. The computational complexity of STRIPS planning. *Artificial Intelligence* 69:165–204.

Cimatti, A.; Roveri, M.; and Bertoli, P. 2004. Conformant planning via symbolic model checking and heuristic search. *Artificial Intelligence* 159:127–206.

Fagin, R.; Halpern, J.; Moses, Y.; and Vardi, M. 1995. *Reasoning about Knowledge*. MIT Press.

Goldman, R. P., and Boddy, M. S. 1996. Expressive planning and explicit knowledge. In *Proc. AIPS-1996*.

Haslum, P., and Jonsson, P. 1999. Some results on the complexity of planning with incomplete information. In *Proc. ECP-99, Lect. Notes in AI Vol 1809*, 308–318. Springer.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14:253–302.

Peot, M., and Smith, D. E. 1992. Conditional nonlinear planning. In Hendler, J., ed., *Proc. 1st Int. Conf. on AI Planning Systems*, 189–197.

Petrick, R., and Bacchus, F. 2002. A knowledge-based approach to planning with incomplete information and sensing. In *Proc. AIPS'02*, 212–221.

Pryor, L., and Collins, G. 1996. Planning for contingencies: A decision-based approach. *Journal of AI Research* 4:287–339.

Rintanten, J. 2004. Complexity of planning with partial observability. In *Proc. ICAPS-2004*, 345–354.

Son, T. C.; Tu, P. H.; and M. Gelfond, A. M. 2005. Conformant planning for domains with constraints-a new approach. In *Proc. AAAI-05*, 1211–1216.

Turner, H. 2002. Polynomial-length planning spans the polynomial hierarchy. In *JELIA '02: Proc. of the European Conference on Logics in AI*, 111–124. Springer-Verlag.

Using Correlation to Compute Better Probability Estimates in Plan Graphs

Daniel Bryce
Dept. of Computer Science & Eng.
Arizona State University
Tempe Arizona 85287–5406
dan.bryce@asu.edu

David E. Smith
Intelligent Systems Division
NASA Ames Research Center
Moffet Field, CA 94035–1000
de2smith@email.arc.nasa.gov

Abstract

Plan graphs are commonly used in planning to help compute heuristic "distance" estimates between states and goals. A few authors have also attempted to use plan graphs in probabilistic planning to compute estimates of the probability that propositions can be achieved and actions can be performed. This is done by propagating probability information forward through the plan graph from the initial conditions through each possible action to the action effects, and hence to the propositions at the next layer of the plan graph. The problem with these calculations is that they make very strong independence assumptions - in particular, they usually assume that the preconditions for each action are independent of each other. This can lead to gross overestimates in probability when the plans for those preconditions interfere with each other. It can also lead to gross underestimates of probability when there is synergy between the plans for two or more preconditions.

In this paper we introduce a notion of the binary correlation between two propositions and actions within a plan graph, show how to propagate this information within a plan graph, and show how this improves probability estimates for planning. This notion of correlation can be thought of as a continuous generalization of the notion of mutual exclusion (mutex) often used in plan graphs. At one extreme (correlation $= 0$) two propositions or actions are completely mutex. With correlation $= 1$, two propositions or actions are independent, and with correlation > 1, two propositions or actions are synergistic. Intermediate values can and do occur indicating different degrees to which propositions and action interfere or are synergistic. We compare this approach with another recent approach by Bryce that computes probability estimates using Monte Carlo simulation of possible worlds in plan graphs.

Introduction

Plan graphs are commonly used in planning to help compute heuristic "distance" estimates between states and goals. A few authors have also attempted to use plan graphs in probabilistic planning to compute estimates of the probability that propositions can be achieved and actions can be performed. This information can then be used to help guide a probabilistic planner towards the most effective actions for maximizing probability or for achieving the goals with a given probability threshold.

Typically, probability information is given for the propositions in the initial state and is propagated forward through the plan graph, in a manner similar to the propagation of cost and resource estimates in classical planning. The probability of being able to perform an action is taken to be the probability that its preconditions can be achieved, which is usually approximated as the product of the probabilities of the preconditions. The probability of a particular action effect is taken as the product of the action probability and probability of the effect given the action. Finally, the probability of achieving a proposition at the next layer is then taken to be either the sum or maximum of the probabilities for the different effects matching that proposition. As an example, consider the plan graph layer shown in Figure 1 where we have two actions a and b each with two preconditions and two unconditional effects. Suppose that the probabilities for the propositions p, q, and r are .8, .5, and .4 as shown in the diagram. The probability that action a is possible would then be the probability of the conjunction $p \wedge q$ which would be $.8(.5) = .4$. Similarly, the probability for action b would be $.5(.4) = .2$. Action a produces effect e with certainty (probability 1), so e simply inherits the probability of .4 from a. Similarly, action b produces effect g with probability .5, so the probability of g can be calculated as $.2(.5) = .1$. The calculation for the effect f is a bit harder because both actions a and b can produce f, and we could in fact develop a plan that uses them both to increase the chances of f. Using a alone, the probability of f is $.4(.5) = .2$, and using b alone the probability is $.2(1) = .2$, so the probability of f using both actions is $.2 + .2 - .2(.2) = .36$.

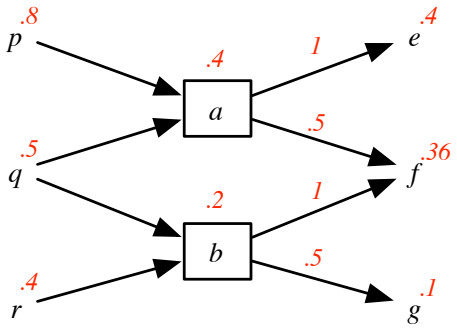


Figure 1: A plan graph layer with simple probability calculations made using the independence assumption.

The problem with these simple estimates is that they assume independence between all pairs of propositions and all pairs of actions in the plan graph. This is frequently a very bad assumption. If two propositions are produced by the same action (e.g. e and f), they are not independent of each other, and computing the probability of the conjunction by taking the product of the individual probabilities can result in a significant underestimate. Conversely, if two propositions are mutually exclusive, then the probability of achieving them both is zero, and the product of their probabilities will be a significant overestimate. In our example, we first assumed that the propositions p, q and r were independent of each other when computing the probabilities of actions a and b. Even if this is so, we then proceeded to assume that actions a and b were independent, when computing the probability of effect f. Clearly this is wrong, since a and b share a precondition.

One obvious way to improve the estimation process would be to propagate and use mutual exclusion information, and assign a probability of zero to actions with mutex preconditions at a given level. However, this only helps with the extreme case where propositions or actions are mutex. It does not help with cases of synergy, or with cases where propositions are not strictly mutex, but it is much "harder" (less probable) to achieve them both.

To attempt to address this problem, we introduce a more general notion which we call "correlation"[1] to capture both positive and negative interactions between pairs of propositions, pairs of actions, and pairs of action effects. In the section that follows, we first give a formal definition of our notion of correlation. We then show how to compute and use correlation information within a plan graph to get better probability estimates. Finally we show some preliminary results, and compare this technique with another recent technique developed by Bryce, Kambhampati, & Smith (2006b).

Definitions and Representation

Action Representation

Similar to the representation used in (Bryce, Kambhampati, & Smith 2006b) an action a is taken to have:

- an enabling precondition, $\text{Pre}(a)$
- a set of probabilistically weighted outcomes, $\Phi_i(a)$

The enabling precondition $\text{Pre}(a)$ is a conjunction of literals, just as for an action in probabilistic PDDL (PPDDL) (Younes *et al.* 2005; Younes & Littman 2004) or an ordinary classical action in PDDL (McDermott 1998). Each outcome $\Phi_i(a)$ has a weight $w_i(a)$ giving the probability that the outcome is realized, and $\Phi_i(a)$ consists of a conjunction of conditional effects $\phi_{ij}(a)$ of the form:

$$\rho_{ij} \rightarrow \varepsilon_{ij}$$

where both ρ_{ij} and ε_{ij} are conjunctions of literals. Of course, ρ_{ij} may be empty, in which case ε_{ij} is an unconditional effect. This representation of effects follows the 1ND normal form presented in (Rintanen 2003).[2]

Correlation

Formally, we define the correlation between two propositions, two actions, or two effects x and y as:

$$C(x,y) \equiv \frac{\Pr(x \wedge y)}{\Pr(x)\Pr(y)} \quad (1)$$

which by Bayes Rule can also be seen as:

$$= \frac{\Pr(x|y)}{\Pr(x)}$$
$$= \frac{\Pr(y|x)}{\Pr(y)}$$

Correlation is a continuous quantity that can range from zero to plus infinity. Essentially, it measures how much more or less probable it is that we can establish x and y together as opposed to if we could establish them independently. It has the following characteristics:

$$\begin{aligned} C(x,y) &= 0 && \text{if } x \text{ and } y \text{ are mutex} \\ &= 1 && \text{if } x \text{ and } y \text{ are independent} \\ &= \tfrac{1}{\Pr(x)} = \tfrac{1}{\Pr(y)} && \text{if } x \text{ and } y \text{ are completely correlated}[3] \end{aligned}$$

More generally, $0 < C(x,y) < 1$ means that there is some interference between the best plans for achieving x and y so it is harder (less probable) to achieve them both than to achieve them independently. Similarly, $1 < C(x,y) < 1/\Pr(x)$ means that there is some amount of synergy between plans for achieving x and y, so it is easier (more probable) to achieve them both than to achieve them independently.

Instead of computing and keeping mutex information in the plan graph, we will compute correlation information between all pairs of propositions and all pairs of actions at each level. It is worthwhile noting that for a pair of propositions or actions x and y we could instead choose to directly store the probability $\Pr(x \wedge y)$, or either of the two conditional probabilities $\Pr(x|y)$ or $\Pr(y|x)$ instead of the correlation $C(x,y)$. This is because these quantities are essentially equivalent - from our definition of correlation and Bayes Rule any of these quantities can be computed from any other. We have chosen to introduce the notion of correlation and store this quantity because:

1. it is symmetric, unlike the conditional values.
2. we only need to store it for cases where it is not one - i.e. the propositions/actions are not independent.
3. it can be easily interpreted and understood in terms of the intuitive concepts of mutex, independence, and synergy.

[1] Not to be confused with the traditional statistical notion of correlation.

[2] The representation in PPDDL (Younes *et al.* 2005; Younes & Littman 2004) is a bit more general since it allows arbitrary nesting of conditional effects and probabilistic outcomes. We have chosen to use the 1ND normal form here because it is a bit easier to work with, and PPDDL can be expanded into this form.

[3] x cannot occur without y, and vice versa, which means that their probabilities must be the same.

Computing Probability and Correlation

To compute probability and correlation information in a plan graph, we begin at the initial state (level 0) and propagate information forward through the plan graph to subsequent levels (just as with construction and propagation in ordinary classical plan graphs). In the subsections that follow, we give the details of how to do this beginning with the initial proposition layer and working forward to actions, then effects, and finally to the next proposition layer.[4]

Computing Action Probabilities

Suppose that we have the probabilities and correlation information for propositions at a given level of the plan graph. How do we use this information to compute probabilities and correlation information for the subsequent action layer? First consider an individual action a with preconditions $\{x_1, \ldots, x_n\}$. The probability that the action can be executed is the probability that all the preconditions can be achieved:

$$\begin{aligned} \Pr(a) &= \Pr(x_1 \wedge \ldots \wedge x_n) \\ &= \Pr(x_1) \Pr(x_2|x_1) \ldots \Pr(x_n|x_1 \ldots x_{n-1}) \end{aligned} \quad (2)$$

If the propositions x_i are all independent this is just the usual product of the individual probabilities of the preconditions. However, if they are not independent then we need the conditional probabilities, $\Pr(x_i|x_1 \ldots x_{i-1})$. Since we have pairwise correlation information we can readily compute the first of these terms:

$$\Pr(x_2|x_1) = C(x_1, x_2) \Pr(x_2)$$

However, to compute the higher order terms (i.e. $i > 2$) we must make an approximation. Applying Bayes Rule we get:

$$\Pr(x_i|x_1 \ldots x_{i-1}) = \frac{\Pr(x_1 \wedge \ldots \wedge x_{i-1}|x_i) \Pr(x_i)}{\Pr(x_1 \wedge \ldots \wedge x_{i-1})}$$

If we make the assumption that $x_1 \ldots x_{i-1}$ are independent for purposes of this computation we get:

$$\Pr(x_i|x_1 \ldots x_{i-1}) = \frac{\Pr(x_1|x_i) \ldots \Pr(x_{i-1}|x_i) \Pr(x_i)}{\Pr(x_1) \ldots \Pr(x_{i-1})}$$

Applying our analogue of Bayes Rule again $i-1$ times, we get:

$$\begin{aligned} \Pr(x_i|x_1 \ldots x_{i-1}) &= \frac{\Pr(x_i|x_1)}{\Pr(x_i)} \cdots \frac{\Pr(x_i|x_{i-1})}{\Pr(x_i)} \Pr(x_i) \\ &= \Pr(x_i) C(x_i, x_1) \ldots C(x_i|x_{i-1}) \\ &= \Pr(x_i) \prod_{j=1 \ldots i-1} C(x_i, x_j) \end{aligned} \quad (3)$$

Returning to the calculation of:

$$\begin{aligned} \Pr(a) &= \Pr(x_1 \wedge \ldots \wedge x_n) \\ &= \Pr(x_1) \Pr(x_2|x_1) \ldots \Pr(x_n|x_1 \ldots x_n - 1) \end{aligned}$$

if we plug in the above expression for the $\Pr(x_i|x_1 \ldots x_{i-1})$ we get

$$\begin{aligned} \Pr(a) &= \Pr(x_1 \wedge \ldots \wedge x_n) \\ &= \prod_{i=1 \ldots n} \left[\Pr(x_i) \prod_{j=1 \ldots i-1} C(x_i, x_j) \right] \end{aligned} \quad (4)$$

Several properties of this approximation are worth noting:

1. the above expression is easy to compute and does not depend on the order of the propositions.
2. If the x_i are independent, the $C(x_i, x_j)$ are 1 and the above simplifies to the product of the individual probabilities.
3. If any x_i and x_j are mutex then $C(x_i, x_j) = 0$ and the above expression becomes zero. If the $C(x_i, x_j)$ are positive but less than one then the probability of the conjunction is less than the product of the probabilities of the individual elements.
4. If the $C(x_i, x_j)$ are greater than one, there is synergy between the conjuncts. The probability of the conjunction is greater than the product of the probabilities of the individual conjuncts, but less than or equal to the minimum of those probabilities.

While these properties are certainly desirable, and match our intuitions, it is reasonable to ask how good the approximation in Equation 4 is in other cases. As it turns out, for a conjunction with n terms, Equation 4 turns out to be exact if only about n of the possible n^2 $C(x_i, x_j)$ are not equal to 1. More precisely:

Theorem 1 *Consider the undirected graph consisting of a node for each conjunct x_i, and an edge between x_i and x_j whenever x_i and x_j are not independent ($C(x_i, x_j)$ is not equal to 1). If this graph has no cycles, then Equation 4 is exact.*

As an example, consider the simple case of:

$$\Pr(a \wedge b \wedge c) = \Pr(a) \Pr(b|a) \Pr(c|ba)$$

Our graph consists of the three nodes a, b and c, and zero to three edges depending on the C's. If b and c are independent, there are only two edges in the graph, and no cycle, so the theorem states that Equation 4 is exact. To see this, with b and c independent the above expansion becomes:

$$\begin{aligned} \Pr(a \wedge b \wedge c) &= \Pr(a) \Pr(b|a) \Pr(c|a) \\ &= \Pr(a) \Pr(b) C(a, b) \Pr(c) C(a, c) \end{aligned}$$

Which is the approximation in Equation 4, since $C(b, c) = 1$

More generally, the proof of this theorem relies on the fact that a graph without cycles can be represented as a tree:

Proof: Suppose we have a conjunction $x_1 \wedge \ldots \wedge x_n$ that obeys the conditions of the theorem. Since the graph has no cycles, it can be arranged as a tree. Without loss of generality, assume the conjuncts are in the same order as a depth first traversal of that tree.

In general, we know that:

$$\Pr(x_1 \wedge \ldots \wedge x_n) = \prod_{i=1,\ldots,n} \Pr(x_i|x_1 \ldots x_{i-1})$$

[4] Because we are dealing with actions that have conditional effects, we will be distinguishing between effects in a plan graph, and the subsequent literal or proposition layer, as is done in IPP (Koehler *et al.* 1997) and (Bryce, Kambhampati, & Smith 2006a; 2006b).

But since the conjuncts are ordered according to a depth first traversal of the tree, each conjunct x_i has only one predecessor $x_j = x_{par(i)}$ (its parent in the tree) for which $C(x_i, x_j)$ is not one. As a result,:

$$\begin{aligned}\Pr(x_i|x_1 \ldots x_{i-1}) &= \Pr(x_i|x_{par(i)}) \\ &= \Pr(x_i)C(x_i|x_{par(i)})\end{aligned}$$

This means that:

$$\Pr(x_1 \wedge \ldots \wedge x_n) = \prod_{i=1,\ldots,n} \Pr(x_i)C(x_i, x_{par(i)})$$

But since $C(x_i, x_j) = 1$ for all $j < i$ and $j \neq par(i)$ there is no harm in adding these terms and we get:

$$\begin{aligned}\Pr(a) &= \Pr(x_1 \wedge \ldots \wedge x_n) \\ &= \prod_{i=1\ldots n} \left[\Pr(x_i) \prod_{j=1\ldots i-1} C(x_i, x_j) \right]\end{aligned}$$

which is Equation 4. ■

Computing Correlation Between Actions

As with propositions, the probability that we can execute two actions, a and b, may be more or less than the product of their individual probabilities. If the actions are mutually exclusive (in the classical sense) then the probability that we can execute them both is zero. Otherwise, it is the probability that we can establish the union of the preconditions for the two actions.

$$\begin{aligned}\Pr(a \wedge b) &= 0 && \text{if } a \text{ and } b \text{ are mutex} \\ &= \Pr\left(\bigwedge(\text{Pre}(a) \cup \text{Pre}(b))\right) && \text{otherwise}\end{aligned}$$

Using Equation 4 we can compute the probability of the conjunction $\Pr\left(\bigwedge(\text{Pre}(a) \cup \text{Pre}(b))\right)$. By our definition of correlation, Equation 1, we can then compute the correlation between two actions a and b.

As an example, consider the plan graph in Figure 1 again.

$$\begin{aligned}\Pr(a \wedge b) &= \Pr\left(\bigwedge(\text{Pre}(a) \cup \text{Pre}(b))\right) \\ &= \Pr(p \wedge q \wedge r) \\ &= \Pr(p)\Pr(q)\Pr(r)C(p,q)C(q,r)C(p,r) \\ &= .8(.5)(.4) = .16\end{aligned}$$

assuming that the correlations are all one. The correlation between a and b is therefore:

$$C(a,b) = \frac{\Pr(a \wedge b)}{\Pr(a)\Pr(b)} = \frac{.16}{.4(.2)} = 2$$

Computing Effect Probabilities and Correlation

Given the tools we have developed so far, it is relatively straightforward to compute the probability of an individual action effect. Let Φ_i be an outcome of action a with weight w_i, and let $\phi_{ij} = \rho_{ij} \rightarrow \varepsilon_{ij}$ be a conditional effect in Φ_i. If the effect is unconditional – that is the antecedent ρ_{ij} is empty – then:

$$\Pr(\varepsilon_{ij}) = w_i \Pr(a)$$

However, if the antecedent ρ_{ij} is not empty, there is the possibility of interaction (positive or negative) between the preconditions of a and the antecedent ρ_{ij}. As a result, to do the computation right we have to compute the probability of the conjunction of the preconditions and the antecedent:

$$\Pr(\varepsilon_{ij}) = w_i \Pr\left(\bigwedge(\text{Pre}(a) \cup \rho_{ij})\right)$$

For convenience, we will refer to the weight w_i associated with an effect ε_{ij} as $w(\varepsilon_{ij})$. We will also refer to the union of the action preconditions and the antecedent ρ_{ij} for an effect ε_{ij} as simply the *condition* of ε_{ij} and denote it $\text{Cnd}(\varepsilon_{ij})$. For an effect ε, the above expression then becomes simply:

$$\Pr(\varepsilon) = w(\varepsilon) \Pr\left(\bigwedge \text{Cnd}(\varepsilon))\right)$$

As with actions, we can compute the probability of the conjunction of $\text{Cnd}(\varepsilon)$ using the approximation in Equation 4.

We can also compute the correlation between two different effects just as we did with actions. For two effects, e and f we have:

$$\Pr(e \wedge f) = w(e)w(f) \Pr\left(\bigwedge(\text{Cnd}(e) \cup \text{Cnd}(f))\right) \quad (5)$$

As before, the probability of the conjunction of $\text{Cnd}(e) \cup \text{Cnd}(f)$ using the approximation in Equation 4. By our definition of correlation, Equation 1, we can then compute the correlation between the two effects e and f.

As an example, consider the two unconditional effects e and g from Figure 1. Since both these effects are unconditional, $\text{Cnd}(e)$ and $\text{Cnd}(g)$ are just the preconditions of a and b respectively. As a result:

$$\begin{aligned}\Pr(e \wedge g) &= w(e)w(g) \Pr\left(\bigwedge(\text{Cnd}(e) \cup \text{Cnd}(g))\right) \\ &= w(e)w(g) \Pr(p \wedge q \wedge r) \\ &= 1(.5)(.8)(.5)(.4) \\ &= .08\end{aligned}$$

since p, q and r were assumed to be independent. Using this, we get:

$$C(e,g) = \frac{\Pr(e \wedge g)}{\Pr(e)\Pr(g)} = \frac{.08}{.4(.1)} = 2$$

Note that Equation 5 for $\Pr(e \wedge f)$ applies whether the effects e and f are from the same or different actions. In the case where they are effects of the same action, there will be overlap of the action preconditions between $\text{Cnd}(e)$ and $\text{Cnd}(f)$. However, the antecedents of the conditional effects may be quite different, and there can be interaction (positive or negative) between literals in those antecedents, which will be captured by the probability calculation in Equation 5.

Computing Proposition Probabilities

Computing the probability for a proposition is complicated by the fact that there may be many actions with effects that produce the proposition, and we are not limited to using only one such action or effect. For example, if two action effects e and f both produce proposition p with probability .5, then we may be able to increase our chances of achieving p by performing both of them. However, whether or not this is a good idea depends upon the correlation between the two effects. If the effects are independent or synergistic, then it is advantageous. If the two effects are completely mutex ($C(e,f) = 0$), then it is not a good idea. If there is

some degree of mutual exclusion between the actions (i.e. $0 < C(e, f) < 1$) then the decision depends on the specific probability and correlation numbers.

Suppose we choose a particular set of effects $E = \{e_1, \ldots, e_k\}$ that produce a particular proposition p. Intuitively, it would seem that the probability that one of these effects would yield p is:

$$\Pr(e_1 \vee \ldots \vee e_k)$$

Unfortunately, this isn't quite right. By choosing a particular set of effects to try to achieve p, we are committing to (trying to) establish the conditions for all of those effects, which means establishing both the action preconditions and the antecedents of each of the conditional effects. There may be interaction between those conditions (positive or negative) that increases or decreases our chances for each of the effects. The above expression essentially assumes that all of the effects are independent of each other.

In this case, the correct expression for $\Pr(p)$ using a set of effects E is both complicated and difficult to compute. Essentially we have to consider the probability table of all possible assignments to the conditions for the effects E, and multiply the probability of each assignment by the probability that the effects enabled by that assignment will produce p. Let $\mathcal{T}(E)$ be the set of all possible $2^{|\text{Cnd}(E)|}$ truth assignments to the conditions in $\text{Cnd}(E)$. Formally we get:

$$\Pr(p_E) = \sum_{\tau \in \mathcal{T}(E)} \Pr(\tau) \Pr(p|\tau) \quad (6)$$

where $\Pr(p_E)$ refers to the probability of p given that we are using the effects E to achieve p.

As an example, consider the calculation of the probability for the proposition f in Figure 1 assuming that we are using both the effects from action a and action b. The set of conditions for these (unconditional) effects is just the union of the preconditions for a and b which is $\{p, q, r\}$. There are eight possible truth assignments to this set, but only three of them permit at least one of the actions:

$p \wedge q \wedge \neg r$	permits a but not b
$\neg p \wedge q \wedge r$	permits b but not a
$p \wedge q \wedge r$	permits both a and b

The probabilities for these truth assignments are:

$$\begin{aligned} \Pr(p \wedge q \wedge \neg r) &= .8(.5)(.6) = .24 \\ \Pr(\neg p \wedge q \wedge r) &= .2(.5)(.4) = .04 \\ \Pr(p \wedge q \wedge r) &= .8(.5)(.4) = .16 \end{aligned}$$

The probability for g using both actions is therefore:

$$\Pr(g) = .24(.5) + .04(1) + .16(.5 + 1 - .5(1)) = .32$$

This calculation was fairly simple because we were only dealing with three propositions p, q and r and they were independent. More generally, however, an expression like $\Pr(p \wedge q \wedge \neg r)$ is problematic when r is not independent of the other two propositions, since we do not have correlation information for the negated proposition. There are a number of approximations that one can use to compute such probabilities. For our purposes, we assume that two propositions are independent if correlation information is not available. Thus, in this case we make the assumption that:

$$\Pr(p \wedge q \wedge \neg r) = \Pr(p \wedge q) \Pr(\neg r)$$

We now return to the problem of computing the probability for a proposition p. In theory we could consider each possible subset E' of effects E that match the proposition p and compute the maximum:

$$\max_{E' \subseteq E} \Pr(p_{E'}) \quad (7)$$

and use Equation 6 to expand and compute $\Pr(p_{E'})$. Unfortunately, when there are many effects that can produce a proposition this maximization is likely to be quite expensive, because 1) we would need to consider all possible subsets of the set of effects, and 2) in Equation 6 we would have to consider all possible truth assignments to the conditions for each set of effects. As a result, some approximation is in order. One possibility is a greedy approach that adds effects one at a time, as long as they still increase the probability. More precisely:

1. Let E be the set of effects matching p
2. let E_0 be the empty set of effects, let $P_0 = 0$
3. let e be an effect in E not already in E_{i-1}, and let $P^* = \Pr(p_{e \cup E_{i-1}})$. If

   e maximizes P^*
   and
   $P^* > P_{i-1}$
   then set
   $E_i = e \cup E_{i-1}$
   $P_i = P^*$

Using this procedure the final set P_i will be a lower bound on:

$$\max_{E' \subseteq E} \Pr(p_{E'})$$

Even this approximation is somewhat expensive to compute, because it requires repeated computation of $\Pr(p_{E'})$ at each stage using equation 6. A different approximation that avoids much of this computation is to construct all maximal subsets E' of the effects in E such that there is no pair of effects e and f in E' with $C(e, f) < 1$ (no interference). We then compute or estimate $\Pr(p_{E'})$ for each such subset and choose the maximum. This approximation has the advantage that we must only calculate $\Pr(p_{E'})$ for a relatively small number of sets.

Computing Correlation Between Propositions

Finally, we consider the probability for a pair of propositions p and q which will allow us to compute the correlation between the propositions. As with a single proposition, this calculation is complicated because we want to find the best possible set of effects for establishing the conjunction. If we let E be the set of effects matching proposition p, and F be the set of effects matching proposition q, then what we are after is:

$$\Pr(p \wedge q) = \max_{\substack{E' \subseteq E \\ F' \subseteq F}} \Pr(p_{E'} \wedge q_{F'})$$

In order to compute $\Pr(p_{E'} \wedge q_{F'}$ we must again resort to considering all possible truth assignments for the union of the conditions for E' and F' as we did in Equation 6:

$$\Pr(p_{E'} \wedge q_{F'}) = \sum_{\tau \in \mathcal{T}(E' \cup F')} \Pr(\tau) \Pr(p \wedge q | \tau) \quad (8)$$

Of course this would also be costly to compute, since it involves computing a complex expression for all subsets of effects in E and F. To approximate this, we could use either the greedy strategy developed in the previous section, or the strategy of finding maximal non-interfering effect subsets.

Given $\Pr(p \wedge q)$ and the individual probabilities $\Pr(p)$ and $\Pr(q)$ we can compute $C(p, q)$ trivially from the definition in Equation 1.

Results

We have developed a preliminary implementation of the technique presented above. Correlation and probability information is computed using the above methods. This information is then used to guide construction of a relaxed plan, which is used to guide a heuristic search planner in a manner similar to that described in (Bryce, Kambhampati, & Smith 2006b). The planner is implemented in C and uses several existing technologies. It employs the PPDDL parser (Younes & Littman 2004) for input, and the IPP planning graph construction code (Koehler *et al.* 1997). Because the implementation and debugging is still not complete, we have so far only tested the ideas on the small domains Sandcastle-67 and Slippery gripper. Figures 2 and 3 show some early results for time, plan length, and node expansion for the sandcastle-67 and slippery Gripper domains respectively. The plots compare 4 different planners:

- CPlan (Hyafil & Bacchus 2004)
- McLug-16 (Bryce, Kambhampati, & Smith 2006b), a recent planner that uses Monte Carlo Simulation on plan graphs
- pr-rp, relaxed plan construction using simple plan graph probability information computed using independence assumptions
- corr-rp, relaxed plan construction using probability and correlation information.

The other two entries (pr-rp-mx and corr-rp-mx) represent variants that are not fully debugged and should therefore be regarded as suspect.

Generally, performance of the four methods is similar on these simple domains. Plans are somewhat longer for pr-rp and corr-rp because the objective for these planners is to maximize probability rather than minimize the number of actions. There is some indication that corr-rp is showing less growth in time and number of node expansions as the probability threshold becomes high, but additional experiments are needed to confirm this and examine this behavior more closely.

Discussion and Conclusions

We have introduced a continuous generalization of the notion of mutex, which we call *correlation*. We showed how such a notion could be used to improve the computation of probability estimates within a plan graph. Our implementation of this technique is still preliminary and it is much too early to draw any significant conclusions about the practicality or efficacy of these computations for problems of any size. In addition to finishing our implementation and doing more significant testing, there are a number of issues that we wish to explore:

Correlation vs Relaxed Plans The approach of keeping correlation information is different from the method of using a relaxed plan to estimate probability in an important way: relaxed plans are constructed greedily, so a relaxed plan to achieve $p \wedge q$ would normally choose the best way to achieve p and the best way to achieve q independently. This will not always lead to the best plan for achieving the conjunction. Correlation information can be used to guide (relaxed) plan selection and would presumably give better relaxed plans. This is the approach we have taken in our preliminary implementation. Of course there is always a tradeoff between heuristic quality and computation time, and this is something we intend to investigate further.

Admissibility Although probability estimates computed using correlation information should be more informative, they are not admissible. The primary reason for this is that keeping only binary correlation information, and approximating the probability of a conjunction using only binary correlation information can both underestimate and overestimate the probability of the conjunction. Note, however, that the usual approach of estimating probability by assuming independence is also not admissible for the same reason. Similarly, relaxed plans do not provide an admissible heuristic - they can underestimate probability because the relaxed plan may not take full advantage of synergy between actions in the domain. It is possible to construct an admissible heuristic for probability by taking:

- the probability of a conjunction to be the minimum probability of the conjuncts,
- the probability of a proposition as the sum of all the probabilities of the producing effects.

However, this heuristic is very weak and not likely to be very effective. It is not yet clear whether we can construct a stronger admissible heuristic using correlation.

Correlation in the Initial State The mechanism we have described easily admits the use of correlation information between propositions in the initial state. That information would be treated in the same was as at any other level in the plan graph. Thus, if the initial state has $\Pr(p \wedge q) = .5$ and $\Pr(\neg p \wedge \neg q) = .5$ we could represent this as $\Pr(p) = \Pr(q) = \Pr(\neg p) = \Pr(\neg q) = .5$ and $C(p, q) = C(\neg p, \neg q) = \frac{.5}{.5 \cdot .5} = 2$. The limitation of this approach is that binary correlation can only approximate joint probability information for conjunctions larger than two.

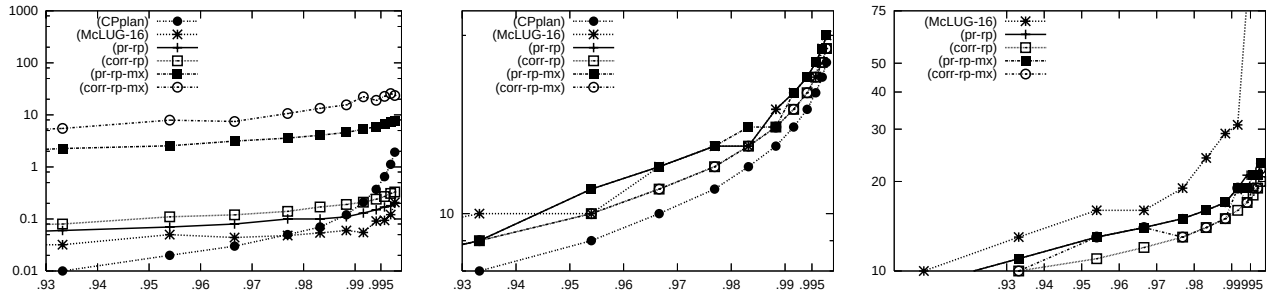


Figure 2: *Run times (s), Plan lengths, and Expanded Nodes vs. probability threshold for sandcastle-67*

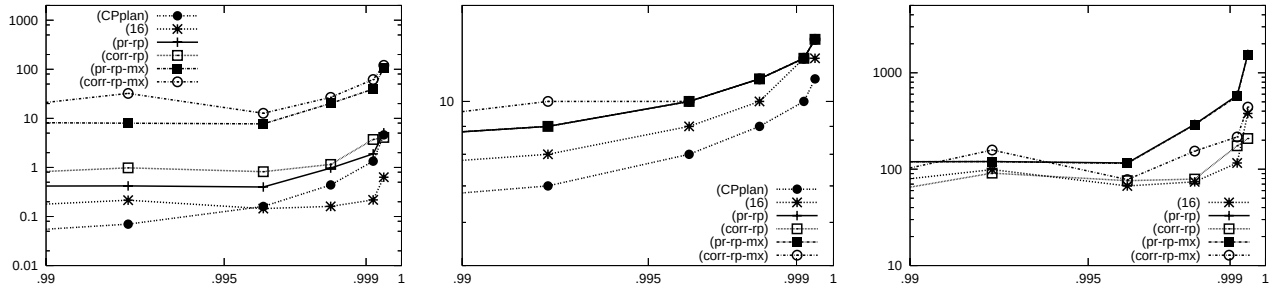


Figure 3: *Run times (s), Plan lengths, and Expanded Nodes vs. probability threshold for slippery gripper*

Bayesian Networks There are a number of similarities between techniques we have used here, and methods used in Bayesian Networks. We speculate that the calculation of probability information for individual actions and pairs of actions could be modeled using a simple Bayes net with nodes for the preconditions and actions, arcs between the preconditions and corresponding actions and arcs between pairs of preconditions that are dependent (correlation not equal to one). These later arcs would be labeled with the conditional probability corresponding to the correlation. It would be necessary to structure the network carefully to avoid cycles among the preconditions. The more complex calculations for propositions would require influence diagrams with choice nodes for each of the establishing effects. There doesn't seem to be any particular advantage to doing this, however. Solution of this influence diagram would require investigating all possible sets of the decisions, which corresponds to the unwieldy maximization over all subsets of establishing effects.

Cost Computation in Classical Planning The idea that we have explored here – a continuous generalized of mutex – is not strictly limited to probabilistic planning. A similar notion of the "interference" between two proposiitons or two actions could be used in classical planning to improve plan graph estimates of cost or resource usage. To do this we could define "interference" as:

$$\begin{aligned} I(x,y) &= Cost(x \wedge y) - (Cost(x) + Cost(y)) \\ &= Cost(y|x) - Cost(y) \\ &= Cost(x|y) - Cost(x) \end{aligned}$$

Positive interference means that there is conflict between two propositions, actions or effects, and that it is more expensive to achieve the conjunction than to achieve them separately. Interference of plus infinity corresponds to mutex. Negative interference corresponds to synergy between the propositions, meaning that achieving them together is easier than achieving them independently. Interference of zero corresponds to independence. Essentially, this can be seen as the logarithm of the definition for correlation given in Equation 1.

The computation of interference for actions, effects and propositions is very similar to what we have described above. The primary difference is that computations for propositions are significantly simpler because there is no need to maximize over all subsets of possible effects that give rise to a proposition. Although we have worked out the equations and propagation rules for this notion of interference, we have not yet implemented or tested this idea. We intend to investigate this in the near future.

Acknowledgements Thanks to Ronen Brafman, Nicolas Meuleau, Martha Pollack and Sailesh Ramakrishnan, and Rao Kamhampati for discussions on this subject and on early versions of these ideas. This work has been supported by the NASA IS-NRA program.

References

Bonet, B., and Geffner, H. 2005. mGPT: A probabilistic planner based on heuristic search. *Journal of Artificial Intelligence Research* 24:933–944.

Bryce, D.; Kambhampati, S.; and Smith, D. 2006a. Plan-

ning graph heuristics for belief space search. *JAIR*. (To appear).

Bryce, D.; Kambhampati, S.; and Smith, D. 2006b. Sequential monte carlo in probabilistic planning reachability heuristics. In *Proceedings of ICAPS'06*.

Hyafil, N., and Bacchus, F. 2004. Utilizing structured representations and CSPs in conformant probabilistic planning. In *Proceedings of ECAI'04*.

Koehler, J.; Nebel, B.; Hoffmann, J.; and Dimopoulos, Y. 1997. Extending planning graphs to an adl subset. In *Proceedings of ECP'97*.

McDermott, D. 1998. PDDL-the planning domain definition language. Technical report, Available at: www.cs.yale.edu/homes/dvm.

Rintanen, J. 2003. Expressive equivalence of formalisms for planning with sensing. In *Proceedings of ICAPS'03*.

Younes, H., and Littman, M. 2004. PPDDL1.0: An extension to PDDL for expressing planning domains with probabilistic effects. Technical report, CMU-CS-04-167, Carnegie Mellon University.

Younes, H.; Littman, M.; Weissman, D.; and Asmuth, J. 2005. The first probabilistic track of the International Planning Competition. *Journal of Artificial Intelligence Research* 24:851–887.

Controlability and Makespan Issues with Robot Action Planning and Execution

Matthieu Gallien and **Félix Ingrand**
LAAS-CNRS*
7, Avenue du Colonel Roche, 31077 Toulouse Cedex 4, France
{matthieu.gallien,felix}@laas.fr

Abstract

In recent years, there has been an increasing interest in getting planners to "execute" their plans in the real world. Among them, the temporal "partial order causal link" planner IxTeT has been endowed with a temporal executive, and has been successfully tested on an ATRV robot. Yet new problems arises from the fact that the plan are now being executed. In domains such as robot planning, many tasks have an uncertain duration (a move in a partially unknown environment takes an uncertain amount of time). The current solution is to use an interval constraint to represent the possible duration in a Simple Temporal Network. By doing this, the planner or the executive are allowed to reduce the possible durations by propagating other constraints. Hence, this reduction may lead to a plan execution failure. This problem, known as the "controllability" problem, as been addressed by a number of people. For example, the 3DC+ algorithm has been proposed to provide dynamic controllability. Yet to our knowledge, no implementation has been made showing the advantage of such approach. This paper presents the results we got using 3DC+ in IxTeT.

Having now a temporal planner which now "complies" with non controllable temporal constraints, we addressed a second problem which arises at execution time. IxTeT being a least commitment planner, the system does not try to minimize the makespan. In fact, the default heuristic is tuned more to spread the actions over the given horizon (to minimize conflict) than to finish the job at the earliest. We implemented a new heuristic which keeps the least commitment strategy yet minimizing the makespan of the plan. This heuristic gives good statistical results on the model we use for our exploration rover type problem.

We have tested these two recent modifications on the robot, but also using a very accurate rover simulator. We present real results and simulated one which show the improvement over the previous version/heuristic. We also proposed a solution to a drawback identified during these tests.

Introduction

Nowadays, many robotic applications need autonomous decisional capabilities. Among them, some make intensive use of planning. Yet, planning is an activity which algorithmic complexity is often incompatible with the reactivity requirement of an exploration rover. On the opposite, one can use an automaton or a simple decisional rule. But, these approaches are not capable to make a forward search to avoid possible conflicts during execution. The main difference is that planning is a proactive method contrary to the more reactive approach of the automaton or decisional rules.

During the past years some planners have proven their ability to handle complex situations required by autonomous systems. Some of these systems (e.g. RAXPS [Jonsson *et al.* 2000], CASPER [Chien *et al.* 2005]) have been deployed. Reasoning about time is necessary to adress these planning problems. The planner must be able to take into account strict deadline, temporal windows for some tasks, durative actions, and durative goals.

Many temporal formalisms were developed in the past years[1]. The STN[2] formalism is often used. The requests on these networks are solved very efficiently by polynomial algorithms. Nowadays, an extension to uncertain constraints has been studied and polynomial algorithms have been proposed.

Actual robotic space exploration mission are very expensive, with a high requirement for quality scientific returns. During the MER mission, the use of MapGen [Ai-Chang *et al.* 2003] has allowed a 25% increase of such return [Rajan 2004]. In an autonomous planner, optimization can be made in two ways: finding directly a good plan or searching several plans to find the better one. Due to limited computational capacity, the second approach is often unreasonable. So we have to modify the planner to search for high quality solutions.

The IxTeT planner[3] [Ghallab & Laruelle 1994] was developed to handle robotic planning problems. It was extended to handle complex resources [Laborie & Ghallab 1995], continuous constraints and constraints between both atemporal and temporal variables [Trinquart & Ghallab 2001]. Further work [Lemai 2004] added a temporal executive to IxTeT. The resources management has been extended to handle continuous resources usage [Lemai & Ingrand 2004].

New issues were raised while experimenting with IxTeT

*Part of this work has been funded by a grant from the ESF (European Social Fund)

[1] Originally IxTeT was a temporal formalism [Ghallab & Mounir-Alaoui 1989] used to check consistency other qualitative temporal relations.

[2] STN: Simple Temporal Network

[3] IxTeT is a system used for chronicle recognition, planning and temporal execution.

new executive. Many plans failed because the executive or the planner reduced the allowed duration of some tasks. In fact, most of the tasks (move, communication , image compression, etc) have uncertain durations. So we decided to implement an existing temporal formalism with explicit uncertainties (STNU). The planner produces temporal contingent plans, i.e. plans that do not fail because of a reduction of an uncertain duration.

Another issue is that while executing the plan, the robot often remains inactive. In fact, the plans produced with a least commitment heuristic may contain unjustified wait periods. Though, they may be necessary to adapt the plan during execution, they reduce the performance of the overall system. This raises a need for plans with a shorter makespan. We solve this problem by modifying the planning heuristic.

These new contributions must be validated by experimental returns. So, we integrated IxTeT and a procedural executive (OpenPRS) to a simulated rover controlling architecture. This simulator allows to make accurate comparisons between the modified versions and the old one.

During the analysis of the results, we identify a drawback of the plan repair mechanism. We propose a solution to it by modifying the way the plan are repaired. This is an ongoing work and some preliminary results are shown.

This paper is organized as follows. The first and second sections describe the IxTeT planner and the embodied executive. The third one describes the validation scenario and the last section describes the experimental results. The fourth introduces a work on improving the plan repair method. We conclude the paper with a discussion and future works.

The IxTeT Planner

Plan-space Search

IxTeT is a temporal constraint-based causal link planner. It uses CSP techniques[4] to maintain the consistency of the plan constraints. In particular, the planner is using a Simple Temporal Network [Dechter, Meiri, & Pearl 1991] for the temporal constraint.

The underlying representation of the plan is state variables ranging over finite and real valued domains. These plans use chronicles [Ghallab, Nau, & Traverso 2004] to describe the world, its evolution and the planning problem. The explicit representation of the time permits to have temporally extended goals, durative actions and rendez-vous or visibility windows.

Definition 1 *A* temporal assertion *on a state variable v is either an* event *or a* persistence condition *on v.*

- *An event, denoted $x@t : (v_1, v_2)$, specifies an instantaneous change of the value of x from v_1 to v at time t, with $v_1 \neq v_2$.*
- *A persistence condition, denoted $x@[t_1, t_2) : v$, specifies that the value of x persists as being equal to v over the interval $[t_1, t_2)$.*

In any temporal assertion, v, v_1 and v_2 can be defined by atemporal variables. t, t_1 and t_2 are temporal variables.

[4]Constraint Satisfaction Problem [Mackworth 1977] (CSP)

Definition 2 *A* chronicle *for a set of state variables $v_1, v_2, ..., v_n$ is a pair $\Phi = (F, C)$, where F is a set of temporal assertions about the state variables $v_1, v_2, ..., v_n$ and C is a set of constraints on variables used in the chronicle.*

Definition 3 *A plan $\mathcal{P}(S, \Phi, G, CA, F, T)$ is described by the state variables contained in S. Φ is a chronicle describing all the temporal assertions of the plan. F is the set of defaults in the plan. $CA \subset \Phi$ contains temporal assertions on variable of S describing the predicted evolution of contingent attributes. The goals are in $G \subset \Phi$, they are temporal persistences on state variables of S. T is the set of tasks in the plan.*

The planner begins with a plan describing the initial situation, the initial goals of the problem and the known predicted evolutions of contingent attributes such as visibility windows. The search is performed until the plan contains no default. These defaults are temporal assertions unexplained in the current plan[5], conflicts between two temporal assertions or possible resource conflicts. At each search step, a default is chosen. One of the resolvants of this default is then applied.

The search toward a solution is performed using a dynamic abstraction hierarchy [Garcia & Laborie 1995] on the attributes of the planning domain. The hierarchy provides a dynamic ordering on the resolution of the defaults in the plan. This ordering can dramatically improve the speed of the planner.

The heuristic computes a cost for each resolver of each default. Then, it chooses the next default to solve ρ using an opportunistic strategy. A default is prefered if it has less resolvants than the other and if it is easier for the planner to choose one among them. The cheapest resolvant is then chosen. The choice is considered as easier for example if one resolvant cost is 0.1 and the other cost is 0.9. We have a better confidence in the automated choice if the difference between costs is high.

$$\frac{1}{\text{Opp}(\rho)} = \sum_{r \in \text{ resolvant}(\rho)} \frac{1}{1 + \text{cost}(r) - \text{cost}_{\min}(\text{resolvant}(\rho))}$$

Theoretically the commitment of one resolvant is computed by estimating the number of completely instantiated reachable solution plans removed by the resolvant r. In fact, estimation were developed. A complete description of the cost functions can be found in [Lemai 2004].

We have implemented a new version of the heuristic with two modified costs. The first considers one single ordering resolvant. The second evaluates the cost of one persistence condition on a state variable.

1. Without the ordering constraint $(t_1 < t_2)$, t_1 can be instantiated any time before or after t_2 with respect to the other temporal constraints. The cost of the resolvant depends on the number of possible instantiation of t_1 after t_2 removed by the constraint.

[5]A temporal assertion is not explained by a plan if it is not an initial condition or if no causal link establishes the assertion.

$c(t_i, t_j)$ is the temporal minimum constraint between t_i and t_j. $d_{\min}(c(t_i, t_j))$, respectively $d_{\max}(c(t_i, t_j))$ is the minimum, respectively maximum duration of the constraint $c(t_i, t_j)$.

$$\text{old\_cost}(t_1 < t_2) = \frac{d_{\max}(c(t_1, t_2) \cap [-\infty, 0])}{d_{\max}(c(t_1, t_2))}$$

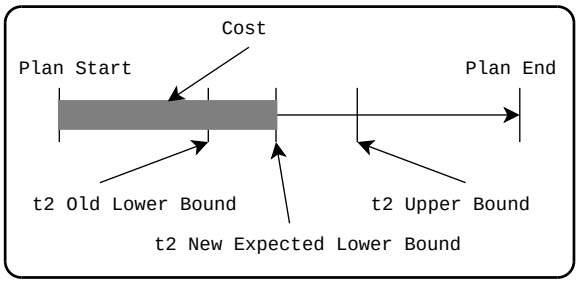


Figure 1: How the new heuristic evaluates the cost of an ordering constraint.

The new heuristic measures the earliness of the resolvant from the point of view of one timepoint. We choose the point of view of t_2[6] (see Fig.1). The new expected lower bound of t_2 is its old lower bound if the lower bound of t_1 is lower than the one of t_2. In the other case, it is the lower bound of t_1.

If $d_{\min}(c(t_{\text{plan\_start}}, t_1)) < d_{\min}(c(t_{\text{plan\_start}}, t_2))$ then

$$\text{new\_cost}(t_1 < t_2) = \frac{d_{\min}(c(t_{\text{plan\_start}}, t_2))}{d_{\min}(c(t_{\text{plan\_start}}, t_{\text{plan\_end}}))}$$

else

$$\text{new\_cost}(t_1 < t_2) = \frac{d_{\min}(c(t_{\text{plan\_start}}, t_1))}{d_{\min}(c(t_{\text{plan\_start}}, t_{\text{plan\_end}}))}$$

2. A causal link resolvant is the conjunction of an ordering constraint, an equality constraint between the values of the two connected temporal assertions. The cost of such resolvants is the sum of the cost of each elementary constraints and of a cost depending on the duration of the value persistence.

   The old cost of the temporal persistence is the ratio of the minimum duration of the persistence by the minimum duration of the plan.

   The new cost is the sum of the equality constraints and of the cost of persistence. The temporal cost is ignored even if the constraint is necessary. This allows the heuristic to order the causal links only by their respective durations.

   We modified the cost of the persistence. The new cost is the ratio of the maximum duration of the persistence by the maximum duration of the plan. By using the maximum duration, possible negative effects are better represented.

[6]Tests have been made with t_1 without significant differences.

Underlying CSPs

Definition 4 *A CSP [Mackworth 1977]* $\Gamma = (V, D, C)$ *is defined by* $V = \{v_i \mid i = 1, ..., n\}$ *the set of all variables,* $D = \{d_i \mid i = 1, ..., n\}$ *the set of variable domains. Finally,* $C = \{c_i \mid i = 1, ..., p\}$ *is the set of constraints on variables contained in* V.

IxTeT uses classical CSPs algorithms for managing constraints on atemporal variables. It uses an STN for managing all the temporal constraints, and a general arc-consistency filtering algorithm [Mackworth & Freuder 1985] for managing symbolic and numeric constraints.

In some cases, we want to link the effects of a task to its duration. For example, if the consumption of a resource depends on the duration of the task, you need a mixed constraint between temporal and atemporal variables. IxTeT features a mechanism [Trinquart & Ghallab 2001] to propagate these constraints.

For example, let t be a task beginning at t_{start} and ending at t_{end}. t is a navigation of expected duration d, speed s and length l. The mixed constraint $c_0 = \{d = (t_{end} - t_{start})\}$ and the constraint $c_1 = \{d * s = l\}$ link the duration of the task to the length of the trajectory.

Definition 5 *An STN [Dechter, Meiri, & Pearl 1991]* $\Theta = (V, D, C)$ *is a restricted form of CSP.* $V = \{v_1, ..., v_n\}$ *is the set of variables.* $D = \{d_i \mid \forall i \; d_i = \mathbb{R}\}$ *is the set of the variable domains.* C *is the set of constraints on variables of* V. *The constraints are all of the form:* $lb \leq v_i - v_j \leq ub$ *which is equivalent to* $v_i - v_j \in [lb, ub]$

On the STN, IxTeT uses a path consistency algorithm like PC-2 [Mackworth & Freuder 1985] that is able to compute the minimal network of an STN (thus removing all values not belonging to a solution). The time complexity is $O(n^3)$ for the complete algorithm and an incremental one[7] is only in $O(n^2)$. During planning, the planner uses the incremental algorithm. During a nominal execution, the same algorithm is used to propagate instantiation of actions start and end timepoints. If a temporal failure invalidates some temporal constraints, a relaxation is done by removing the failed constraints and by repropagating all the remaining variables using the complete algorithm.

It is possible to use a fast algorithm [Muscettola, Morris, & Tsamardinos 1998] for STN execution. It is based on the property of dispatchability. However it is much efficient than using an $O(n^2)$ algorithm, we cannot use it because the propagation of mixed constraints needs a complete algorithm.

Simple Temporal Network with Uncertainties

Definition 6 *An STNU [Vidal & Fargier 1999]* $\Theta = (V, D, C_{clb}, C_{ctg})$ *with* V *the set of variables,* D *the set of domains. These definitions are the same than the STN ones. The set* C_{clb} *is all the controllable constraints equivalent to STN constraints.* C_{ctg} *is a set of contingent constraints of the form* $lb \leq v_i - v_j \leq ub$. *The duration of this constraints can only be observed.*

[7]If only one constraint of the STN is restricted, the incremental algorithm can be used.

The introduction of a new type of constraint changes the consistency notion inherited from the STN. Three main levels of controllability have been defined [Vidal & Fargier 1999]. Schematically, an STNU is weakly controllable if the execution controller needs to know all the contingent durations before executing the plan. An STNU is strongly controllable if the execution controller instantiates the controllable timepoint at the same time whatever the observed contingent durations are. Finally, an STNU is dynamically controllable if the execution controller must take decisions knowing only the past observations and timepoint instantiations. Finally an STNU is pseudo-controllable if it is contingent and no contingent durations are squeezed [Morris, Muscettola, & Vidal 2001].

The two possible choices are dynamic or strong controllability. The two possible properties can be checked in polynomial time. The first one is more flexible and may have a lower makespan[8]. The second one has a lower complexity during execution. This controllability is more restrictive than the dynamic one, fewer STNUs are strongly controllable. We decided to experiment the dynamic controllability due to these advantages.

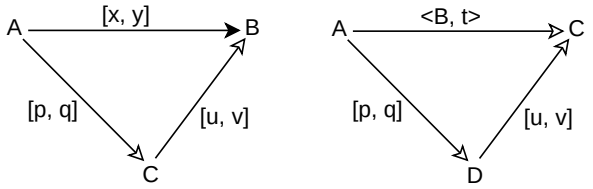


Figure 2: Basic constraint triangles analyzed by 3DC+. The left triangle is used when searching for new necessary constraints. The right one is checked during the propagation of a new ternary constraint called a "wait".

The algorithm 3DC+ [Morris, Muscettola, & Vidal 2001] checks if an STNU is dynamically controllable. It is polynomial in time, computes the minimal graph and the most flexible one. It is complete and correct. It allows to produce plans and to safely execute them.

The algorithm performs several cycle of checking on triangles containing at least one contingent constraint. On each triangle (see Fig.2), a test is performed in order to identify the possible necessary constraints to enforced dynamic controllability. Four cases exist:

- In the case where C necessary follows B, nothing new is needed.
- In the case where C necessary precedes B, a new constraint on AC is needed : $(C - A)$ in $[y - v, x - u]$.
- In the other cases C and B are unordered. If $y - x \leq v$ then we must add the constraint $(C - A)$ in $[y - v, +\infty[$.
- Otherwise, a new type of constraint called "wait" is needed: $< B, y - v >$. It means that before executing C you must wait $y - v$ after the execution of A or the observation of B.

At the start of each cycle, the APSPG[9] is computed. Then, the new necessary constraints are added. After the analyzis of all triangles, the algorithm propagates all the waits to ensure they will never break during execution. The termination condition is reached if the network is not pseudo-controllable or if no new constraint has been added.

We have made two little improvements to this algorithm. The first concerns the fact that we want to incrementally maintain an STNU dynamically controllable during planning. Before the planning adds a new constraint to the network, our algorithm removes all existing waits. This is due to the fact that some of them are no more necessary due to the new constraint. More work is needed to find a better removal criterion.

The original algorithm performs several complete propagation with a complexity of $0(n^3)$ in order to test the property of pseudo-controllability. We have replace this by maintaining always the network pseudo-controllable by incrementally propagating all addition of STN constraints during enforcement of dynamic controllability. If we suppose that k constraints are added during a cycle, the complexity is in $O(kn^2)$. Experimentally, we observed that k is rather constant. Yet the maximum bound of k is in $O(n^2)$.

Autonomous Mission Execution

Temporal Plan Execution

During execution, IxTeT controls the beginning, ending and interruption of tasks running onboard the rover, by sending commands to the robot. The executive begins with an initial plan produced by the planner.

Definition 7 *A* timepoint t *is executable if all timepoints* t_i *that must precede* t *have been executed and if the temporal constraints between* t *and the* t_i *are such that the instantiation of* t *at time* u *is possible.*

IxTeT executes the plan following a cycle. If a new event is received, it updates the plan. After that, if needed, a plan repair process occurs. At the end, it converts the executable timepoints into commands and updates the plan accordingly. The cycle has an expected maximal duration denoted μ.

Definition 8 *An* execution task $t(t_{start}, t_{end}, T, P)$ *starts at the timepoint* t_{start}*, ends at timepoint* t_{end} *and contains a set* P *of temporal propositions on state and resource variables.* T *may contain internal timepoints used to describe temporal propositions of* P*.*

For simplicity, the executive only considers the start timepoint t_{start} and the end timepoint t_{end} of tasks.

Definition 9 *A task* $t(t_{start}, t_{end}, T, P)$ *is* supported by the plan $\mathcal{P}(S, G, CA, F, T, C)$ *if no temporal proposition* P *of the tasks is contained in* F*.*

[8] If all the contingent durations are equal to their upper bound, dynamic controllability may be as efficient as the strong one. Otherwise, dynamic controllability is more time efficient.

[9] APSPG: is the All Pairs Shortest Path Graph of the STNU. If one contingent duration is squeezed, then the STNU is not pseudo-controllable and thus not dynamically controllable. Our implementation uses a PC-2 algorithm [Mackworth & Freuder 1985] with a complexity of $O(n^3)$.

Typically, the system starts a task $t(t_{start}, t_{end}, T, P)$ as soon as possible if its first timepoint t_{start} is executable and the actual plan supports its execution.

Definition 10 *A plan $\mathcal{P}(S, G, CA, F, T, C)$ is valid if all running tasks $t_i(t^i_{start}, t^i_{end}, T^i, P^i)$ are supported by $\mathcal{P}$.*

The system only interrupts tasks that can be preempted. Other tasks report their termination by sending a message summarizing the exact result of their execution.

In general, the system decides to interrupt a task $t(t_{start}, t_{end}, T, P)$ when the current time $u > t_{end}$ or if the plan is no more valid. If the task is controllable and the plan still remains valid, the decision will be made at:

early time: $u_{early} \geq t^{lb}_{end}$ if the task is an early preemptive one.

late time: $u_{late} = t^{ub}_{end} - \mu$ if the task is a late preemptive one.

The executive has to check the validity of the task reports considering the current plan. If the task has modified the level of resources, the executive checks for possible future conflicts. If the report is not nominal, it is considered as a task failure report.

Unexpected Events During Execution of a Mission

The system may receive messages of new goals to satisfy, changes of resource capacity or task failures.

IxTeT integrates these messages into its current plan. During this insertion, it partially invalidates the current plan by removing some causal links and some temporal constraints. If the new plan is not valid, execution is immediately aborted. Otherwise it always tries to make local plan repair in order to restore the plan correctness.

The plan repair search is similar to the plan search. Except for the new time variables those are inserted after the end of the current cycle. Currently this plan repair process does not remove any constraints on the plan and by extension any tasks. IxTeT only allows a part of the maximal cycle duration to be allocated to plan repair process. It is defined before executing the mission.

IxTeT interleaves the planning process with perception of new events and new decisions. During this search, the system has to always keep a valid plan for integrating the reports or to take the decisions.

The plan repair process keeps information about which plans are valid in the search tree. Thanks to that, if the time limit is reached, it extracts the last valid plan for the other parts of the cycle. If an event is received or if a decision is taken, a new search tree is started with the current plan as its root, otherwise the search continues. This process implies that the search cannot backtrack to choices made in precedent repair cycles.

The plan repair process can fails in two cases. The first is when there is no correct plan in all the search space. The second case is when a time failure occurs before finding a new plan. These time failures correspond to the impossibility to instantiate a temporal variable corresponding to the beginning of an action, or a goal. These failures occur when the current plan repair process does not yet find a plan supporting the task. If the plan repair process fails, all the running actions are interrupted and a complete replanning is done.

When a complete replanning is needed, the system builds a new initial plan containing the current state and the remaining predicted evolution of contingent state variables. The remaining goals are kept.

During the search, at each step of the planning process IxTeT verifies that enough time remains for executing the plan. This is done by adding to the plan a temporal variable corresponding to the end of the replanning. All new inserted tasks are constrained to occur after this temporal variable. If the upper bound of the special variable is lower than the actual time minus a constant time needed to initialize the executive, the search is stopped.

When a replanning fails, the system abandons one goal. The goals are ranked according to a fixed priority value. The goal or one of the goals with the fewer priority is suppressed. A new replanning attempt is made until a solution plan is found. The abandonned goals are reinserted before any new complete replanning, and if their temporal constraints are still valid. A goal is retried only one time.

The Experimental System

The Simulated Architecture

IxTeT runs on the robot Dala (see Fig.3) and on a simulator of this robot. The simulator allows us to perform accurate tests of the different IxTeT strategies presented in the paper. The environment and the initial conditions are exactly the same.

We use the LAAS architecture [Alami *et al.* 1998] (see Fig.3). The basic functions of the robot are encapsulated in software modules generated with GenoM. Normally, these basic modules are then run onboard a robot. The simulator is describbed in [Joyeux *et al.* 2005].

In the top of the simulated functional layer, there is OpenPRS [Ingrand *et al.* 1996]. It is used for actions monitoring and refining of the high level tasks into low level commands.

Evaluation Scenario

The main goal of the scenario is to evaluate the new modification of IxTeT. A typical mission is defined as part of the evaluation scenario. The mission is then instantiated with different parameters to test the stability of the results.

Typical Mission This is an exploration rover like mission. The robot must acquire scientific data from several places. During its mission, it has to communicate with an orbiter during visibility windows. It has to be at a specific location at the end of the mission (for example back at its starting point).

For the need of the demonstration, we add a constraint on the motion of the cameras[10] (i.e. the MOVE_PAN_TILT_UNIT task, see Fig.7) to heat the motors before using them. The power constraints are such that heating is compatible with motion of the robot.

[10] On the robot, the cameras are mounted on a pan and tilt unit.

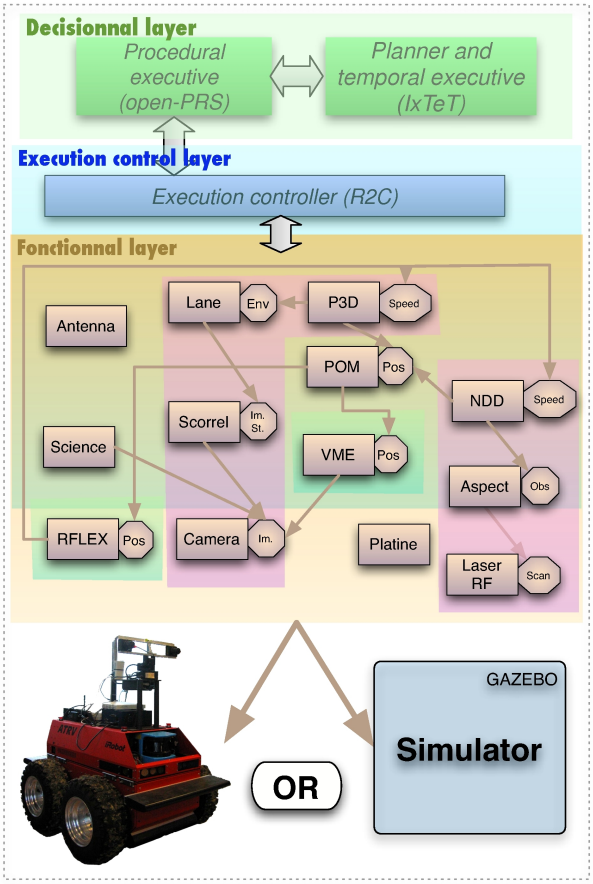


Figure 3: The LAAS architecture used for controlling the DALA mobile robot.

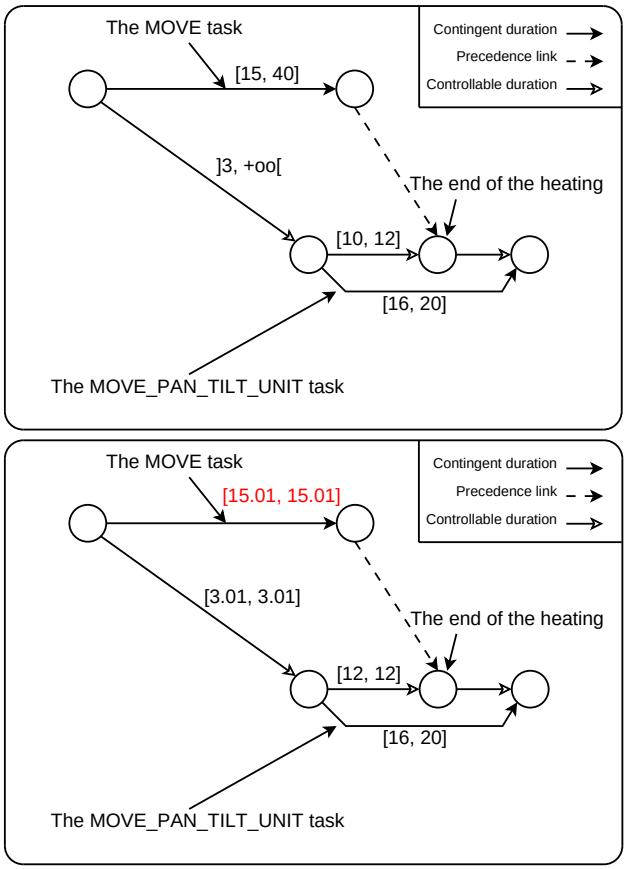


Figure 4: STN resulting of the insertion of a MOVE and a MOVE_PAN_TILT_UNIT tasks in a plan. The top one is before execution. The bottom one shows how the possible duration of the MOVE can be squeezed without using an STNU and 3DC+.

In the contingent plan model, the heating is allowed to be during a ground navigation. For the controllable model, the heating is made incompatible with the robot motions. This unnecessary constraint is added because the planner starts task as soon as possible. By doing this, it will start the “move cameras" task at a date whose propagation will squeezed the possible durations of the robot move (see Fig.4). This may induces an execution failure. The use of 3DC+ removes the need for the new constraint by adding a “wait".

Different Configurations Tested

IxTeT now features two different planning heuristics and two different time management systems. This defines four IxTeT instantiations.

Four different worlds have been defined. The first one has no obstacles. The second and the third ones each contain three obstacles. The last world contains all the obstacles in worlds two and three.

We have tested some goals configurations and some temporal window specifications. The values have been chosen randomly. One specific set of goals has also been generated considering a specific test case.

Results

The makespan:

1. The new heuristic has negative effect only with STN. Effectively the plans are too constrained for taking advantage of the plan adaptation capacity of the executive. The executive has difficulty to achieve the communication during the visibility windows. The communication are modeled to take between 18 s and 25 s. The effective duration was about 21 s. The execution often reduces the possible duration of the communication to less than 21 s, leading to a failure of the task.
2. If we compare the two STNU configurations, the heuristic is very robust to the world or the goal configurations. Yet it is not a big difference, in a particular test, it was about 10% better. We discuss in the next paragraph a reason of inefficiency of plan repair which may explains this result.
3. If we compare the two configurations using the new heuristic, the STNU has bad results and is longer than the STN. This is due to two reasons. The first is that the planner produces longer plan with 3DC+ because it

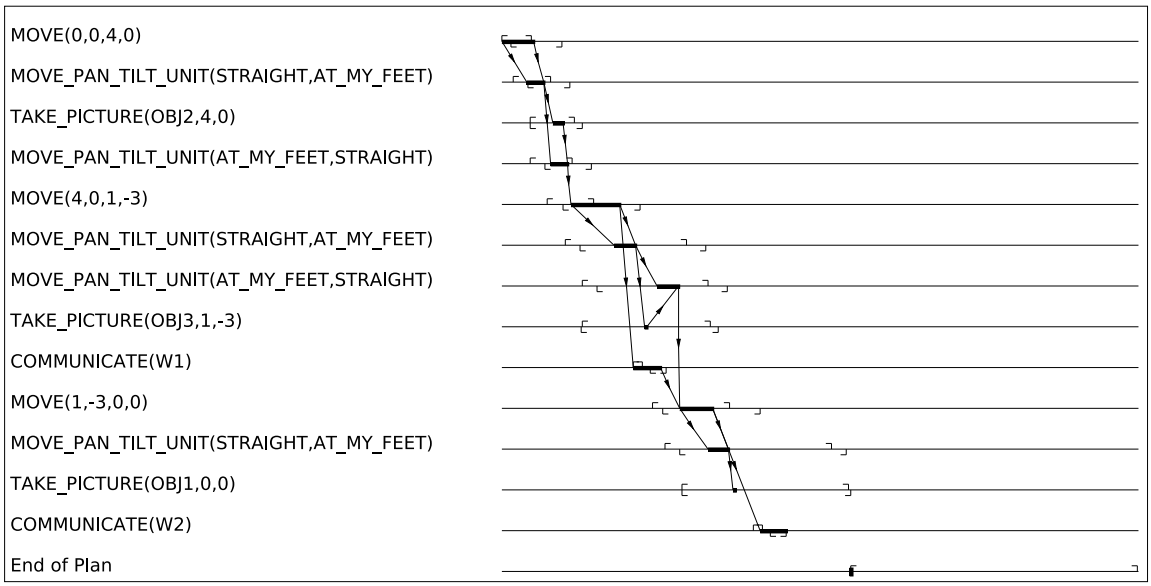


Figure 5: Typical initial plan of the robot during our experiment using an STNU as time formalism and using the makespan minimizing heuristic.

cannot squeeze of task durations at execution time and so produces less constrained plans. The second reasons is that plans with 3DC+ are often repaired while plans without cannot be repaired and are canceled. Currently the plan repair process produces plans with a bad quality including useless tasks. So these plans are ineffective. We discuss this problem latter.

The robustness We have compare the results from each flavor of the system. The use of the new heuristic against an STN gives very bad results. In most case, plans failed because the task duration were too squeezed by propagation.

If we compare the STN and the STNU for each heuristics, we have less plan failures. In general all goals are achieved by the STNU configurations. The STN configurations can fail in some world and can success in other worlds. Clearly the system is less reliable.

Planning duration and other time measures The planning time is really similar on little problem (≤ 5 scientific goals). There is no significant difference using one heuristic or the other. In our tests, the problems are always easy due to needed flexibility (in order to have time for new goals). So this heuristic is not tested on hard problem. Yet, the usage of a complex world representation (numeric domains and actions effect depending of the duration of the task) makes harder to prove that a planning problem has no solution even if we use the old one.

The use of 3DC+ implies a higher temporal complexity at each planning and execution step. But contrary to the worst case, with a good implementation of the algorithm, the added cost is acceptable. During execution, the instantiation of a task start or stop takes 0.001 s with an STN and no more than 0.01 s with an STNU. The duration of plan repair steps are about 3 times higher with an STNU. This results shows that the cost of 3DC+ is compatible with its usage on an autonomous robot.

Prospectives and Current Work

We identify a drawback of the current plan repair process of IxTeT during our test. Sometimes, a repaired plan contains unnecessary tasks or not optimal tasks. For example, during our tests, we add new picture goals. The planner produces a plan resulting in two-way navigation from a new goal location and an existing waypoint. This may lead to a very low quality plan.

This situation arises when a new unsatisfied goal is added directly or during plan repair. This unsatisfied goal must not yet be an establisher of any temporal assertion in the plan. This goal must be on a state variables which has to take at least three different values (e.g. the position of the robot).

During the plan repair process, the planner establishes the new goal. If this goal is satisfied after any temporal assertion in the original plan, all is good. In the other case, the planner must insert tasks to reestablished the precedent values of the state variables. These tasks may be unnecessary, or their insertions impossible. The fig.8 illustrates the problem with a navigation.

A basic solution

We describe this work using an exemple run. It is a mission with initially 5 picture goals and 2 communication goals. The initial plan is found in 1.7 s. It contains 22 tasks, 90 atemporal variables and 75 temporal variables. One picture goal is added during the first communication.

The problem illustrated in the fig.8 comes from a limited relaxation of the plan before the plan repair process. The

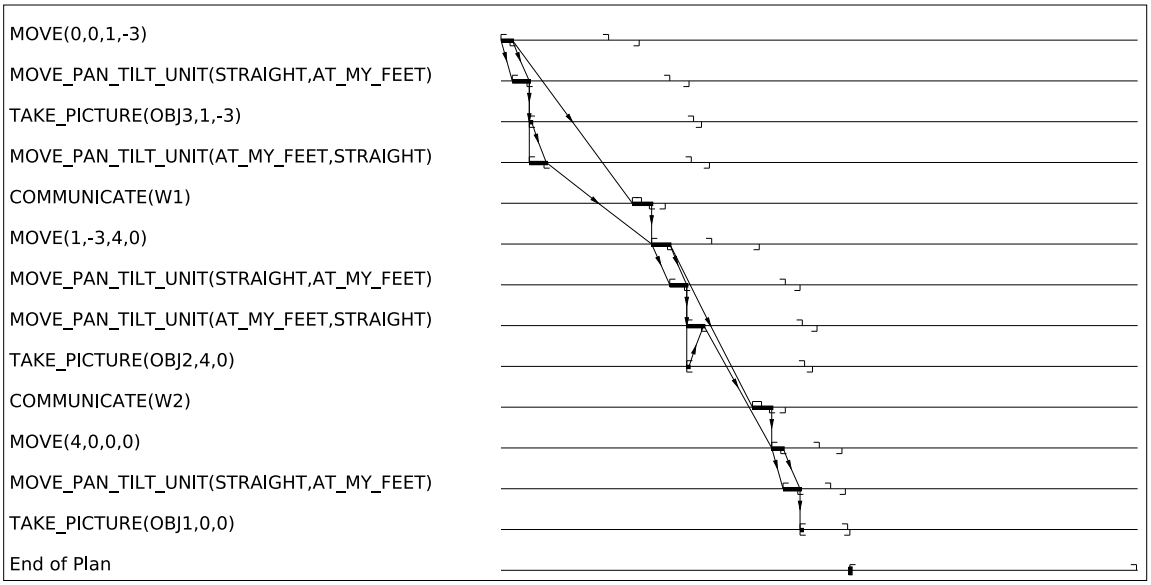


Figure 6: Typical initial plan of the robot during our experiment using an STNU and the commitment minimizing heuristic.

```
task MOVE(?initL,?endL)(st,et){
  ?initL,?endL in LOCATIONS;
  event(ROBOT_POS():(?initL,IDLE_POS),st);
  hold(ROBOT_POS():IDLE_POS,(st,et));
  event(ROBOT_POS():(IDLE_POS,?endL),et);
  event(ROBOT_STATUS():(STILL,MOVING),st);
  hold(ROBOT_STATUS():MOVING,(st,et));
  event(ROBOT_STATUS():(MOVING,STILL),et);
  hold(PTU_POS():FORWARD,(st,et));
  variable ?di,?du,?dist;
  variable ?duration;
  distance(?initL,?endL,?di);
  distance_uncertainty(?du);
  ?dist = ?di * ?du;
  speed(?s);
  ?dist = ?s * ?duration;
  contingent ?duration = et − st;
}latePreemptive

task MOVE_PTU(?initL,?endL)(st,et){
  timepoint end_heat;
  ?initL,?endL in PTU_POSITIONS;
  hold(ROBOT_STATUS():STILL,(end_heat, st));
  event(PTU_STATUS():(COLD, HEAT),st);
  hold(PTU_STATUS():HEAT,(st,end_heat));
  event(PTU_STATUS():(HEAT,MOVING),end_heat);
  hold(PTU_STATUS():MOVING,(end_heat,et));
  event(PTU_STATUS():(MOVING,COLD),et);
  hold(PTU_INIT():TRUE,(st,et));
  hold(PTU_POS():?initL,(st,end_heat));
  event(PTU_POS():(?initL,PTU_POS_IDLE),end_heat);
  hold(PTU_POS():PTU_POS_IDLE,(end_heat,et));
  event(PTU_POS():(PTU_POS_IDLE,?endL),et);
  (end_heat − st) in [10,12];
  contingent (et − st) in [16,20];
}latePreemptive
```

Figure 7: An example of move and camera orientation (the cameras are mounted on a pan&tilt unit).

solution described in the precedent sections of this paper, removes only causal links. A planner like IxTeT, adds constraints on variables to make causal links valid. If these constraints remains, the plan repair produces a suboptimal solution. The solution is to remove the constraints at the same time than the link.

We integrated the algorithms describbed in [Surynek & Barták 2004; Barták & Surynek 2005]. We adapt the first one to continuous domains and use it to manage the filtering in the atemporal CSP. This permits to remove the atemporal constraints supporting a causal link.

The temporal constraints are not removed because the result is not satisfying. The main reason is that the planner introduces many more temporal constraints for solving conflicts between two temporal assertions than for justifying causal links. Relaxing only the constraints associated with causal links does not significantly relax the plan contrary to atemporal constraints.

We made another modification to the precedent algorithm. The executive selects all the causal links that may prevent the planner from finding a solution. It removes from this set the causal links that were added to justify a task. Precisely, during planning, the planner adds tasks to justify unestablished temporal assertions. The new task is then linked to the temporal assertion by a causal link. The final set is about 41 causal links in our exemple run.

We have try our solution using the simulator and the repaired plan is similar to the one illustrated on fig.9. The time needed for remove constraints is negligeable: 0.02 s for 21 constraints. The planner finds a solution containing only necessary tasks or navigations. The duration of plan repair is approximatively 1.2 s for a plan with five picture goals plus a new one. The needed relaxation of the mixed constraints[11] takes about 1.5 s. This is longer than a complete replanning but keep a plan more stable and allows to execute valid part of the plan.

This solution is yet limited to simple cases where actions partial order allows the planner to find a new solution. In the next part we discuss a solution to this problem.

A future work

Actually, we store the constraints associated to causal links. We are able to produce better plans using plan repair. We lack the capacity to change the order between tasks. In a typical mission, communications are made during fixed temporal windows while navigations are free. If we want the

[11] A mixed constraint is relaxed by removing both the temporal constraint and the atemporal one. This leads to a complete propagation of the STN.

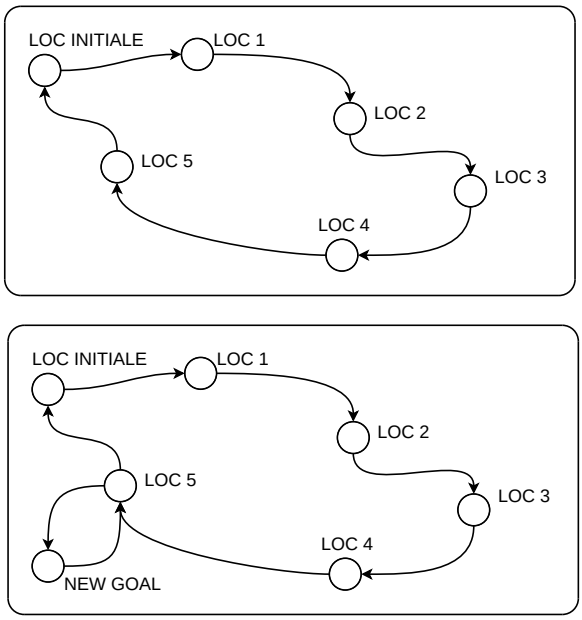


Figure 8: Initial plan (at top) and repaired plan after a new goal inserted (at bottom).

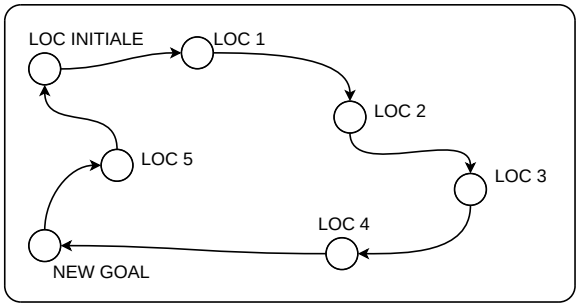


Figure 9: Expected plan after a plan repair

plan repair process to be complete, we must be able to move navigations around communications.

This may be done by storing which constraints have been inserted to solve conflict between temporal assertions. The information is associated with each involved assertions. If a plan repair process is needed, the executive removes only the constraints associated to temporal assertions of not yet executed tasks.

Conclusion

We have describe a temporal planner and executive whose plan execution raises new issues.

The first one is to deal with uncontrollable durations. We use a temporal framework with explicit uncertainties.

The second one is the bad quality of the plans when compared with a time optimal plan. We modify the search control of the planner to find better plans by modifying the planning heuristic.

A simulation architecture is used to evaluate the two solutions. An evaluation scenario is described and used to acquire results.

During the test, the heuristic has shown a good robustness. Yet, an identified drawback limits the performance of this work. A solution using the plan repair ability of IxTeT is briefly describbed in the last part of the paper.

The integration of an STNU shows that it is usable on a rover, contrary to the worst time complexity of $O(n^5)$ [Morris & Muscettola 2005][12], with n the number of temporal variables. It shows a better robustness of the execution of the mission. If one goal is achievable, with 3DC+, it is executed.

But clearly, 3DC+ algorithm lacks the capacity to handle dynamic STNU. Some work have been made in this way [Stedl & Williams 2005]. Yet to our knowledge, more work is needed to handle constraint removals.

References

Ai-Chang, M.; Bresina, J.; Charest, L.; Jónsson, A.; Hsu, J.; Kanefsky, B.; Maldague, P.; Morris, P.; Rajan, K.; and Yglesias, J. 2003. Mapgen: Mixed initiative planning and scheduling for the mars 03 mer mission. In *Proceedings of iSAIRAS*.

Alami, R.; Chatila, R.; Fleury, S.; Ghallab, M.; and Ingrand, F. 1998. An architecture for autonomy. *International Journal of Robotics Research, Special Issue on Integrated Architectures for Robot Control and Programming* 17(4):315–337.

Barták, R., and Surynek, P. 2005. An Improved Algorithm for Maintaining Arc Consistency in Dynamic Constraint Satisfaction Problems. In *International Florida AI Research Society Conference*.

Chien, S.; Tran, D.; Rabideau, G.; Cichy, B.; Davies, A.; Sherwood, R.; Castano, R.; Mandl, D.; Frye, S.; Trout, B.; D'Agostino, J.; Shulman, S.; and Boyer, D. 2005. The autonomous sciencecraft on earth observing one. In *i-SAIRAS-2005*.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal Constraint Network. *Artificial Intelligence* 49(1-3):61–95.

Garcia, F., and Laborie, P. 1995. Hierarchisation of the search space in temporal planning. In *EWP*.

Ghallab, M., and Laruelle, H. 1994. Representation and Control in Ixtet, a Temporal Planner. In *AIPS*, 61–67.

Ghallab, M., and Mounir-Alaoui, A. 1989. Managing Efficiently Temporal Relations Through Indexed Spanning Trees. In *IJCAI*.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning : Theory and Practice*. Morgan Kaufmann.

Ingrand, F.; Chatila, R.; Alami, R.; and Robert, F. 1996. PRS: A High Level Supervision and Control Language for Autonomous Mobile Robots. In *IEEE International Conference on Robotics and Automation*.

Jonsson, A. K.; Morris, P. H.; Muscettola, N.; Rajan, K.; and Smith, B. D. 2000. Planning in Interplanetary Space:

[12] It is a slightly modified version of 3DC+ with a cutoff mechanism.

Theory and Practice. In *Artificial Intelligence Planning Systems*, 177–186.

Joyeux, S.; Lampe, A.; Alami, R.; and Lacroix, S. 2005. Simulation in the LAAS Architecture. In *ICRA Workshop on Interoperable and Reusable Systems in Robotics*.

Laborie, P., and Ghallab, M. 1995. Planning with sharable resource constraints. In *IJCAI*.

Lemai, S., and Ingrand, F. 2004. Interleaving temporal planning and execution in robotics domains. In *AAAI*.

Lemai, S. 2004. *IxTeT-eXeC : planning, plan repair and execution control with time and resource management*. Ph.D. Dissertation, LAAS-CNRS and Institut National Polytechnique de Toulouse, France.

Mackworth, A. K., and Freuder, E. C. 1985. The Complexity of Some Polynomial Newtork Consistency Algorithms for Constraint Satisfaction Problems. *Artificial Intelligence* 25(1):65–74.

Mackworth, A. 1977. Consistency in networks of relations. *Artificial Intelligence* 8:99–118.

Morris, P. H., and Muscettola, N. 2005. Temporal Dynamic Controllability Revisited. In *AAAI*.

Morris, P. H.; Muscettola, N.; and Vidal, T. 2001. Dynamic control of plans with temporal uncertainty. In *IJCAI*.

Muscettola, N.; Morris, P.; and Tsamardinos, I. 1998. Reformulating temporal plans for efficient execution. In *Principles of Knowledge Representation and Reasoning*.

Rajan, K. 2004. Invited talk: Mapgen. In *IWPSS 2004, 4th International Workshop on Planning and Scheduling for Space, June 23 - 25*.

Stedl, J., and Williams, B. 2005. A fast incremental dynamic controllability algorithm. In *ICAPS Workshop on Plan Execution: A Reality Check*.

Surynek, P., and Barták, S. 2004. A New Algorithm for Maintaining Arc Consistency After Constraint Retraction. In *Principles and Practice of Constraint Programming*.

Trinquart, R., and Ghallab, M. 2001. An extended functional representation in temporal planning : towards continuous change. In *ECP*.

Vidal, T., and Fargier, H. 1999. Handling contingency in temporal constraint networks: from consistency to controllabilities. *JETAI* 11(1):23–45.

Using Planning, Scheduling and Execution for Autonomous Mars Rover Operations

Tara Estlin, Daniel Gaines, Caroline Chouinard, Forest Fisher, Rebecca Castano,

Michele Judd, Benjamin Bornstein, Robert C. Anderson and Issa Nesnas

Jet Propulsion Laboratory
California Institute of Technology
Pasadena, CA 91109
{firstname.lastname}@jpl.nasa.gov

Abstract

With each new rover mission to Mars, rovers are traveling significantly longer distances. This distance increase raises not only the opportunities for science data collection, but also amplifies the amount of environment and rover state uncertainty that must be handled in rover operations. This paper describes how planning, scheduling and execution techniques can be used onboard a rover to autonomously generate and execute rover activities and in particular to handle new science opportunities that have been identified dynamically. We also discuss some of the particular challenges we face in supporting autonomous rover decision-making. These include interaction with rover navigation and path-planning software and handling large amounts of uncertainty in state and resource estimations. Finally, we describe our experiences in testing this work using several Mars rover prototypes in a realistic environment.

Introduction

NASA has demonstrated that mobile robotic craft are a viable and extremely useful option for exploring the surface of other planets. The 2003 Mars Exploration Rovers (MER) have traveled across thousands of meters of terrain and gathered large amounts of valuable scientific data that is being used to answer many questions about the Martian environment. Future missions are being planned to send additional robotic explorers to Mars as well as to the moon and outer planets.

High-level decision making for these efforts, including for the MER Mission and the 1997 Mars Pathfinder mission, is performed on Earth through a predominantly manual, time-consuming process. For MER, a ground-based AI planning and scheduling tool (Ai-Chang, 2004) is used to support science plan evaluation, however, a large team of engineers is still required to perform a number of manual steps in order to generate a daily command sequence and uplink it to the rovers.

One significant problem with this approach to rover operations is that it can result in frequent underutilization of the robotic assets. Typically, command sequences are generated using conservative estimates on their time and resource requirements (e.g., overestimate time to drive to a new location to ensure that rover arrives by end of day). If the rover performs nominally or better than predicted, then the overestimates result in significant idle time for the rover, time in which the rover could be acquiring additional observations or sleeping to conserve energy.

If activities take longer to execute than expected then they will be aborted and future activities will be dropped. This can result in higher priority science observations not being performed because earlier observations ran long. Finally, new science opportunities can only be identified after scientists on the ground have been able to review downlinked data. This approach means many opportunities may not be realized, since once identified on Earth, the rover may have already traveled far past the object of interest. Further, not all image data can be downloaded thus some interesting terrain features may be completely missed. This case becomes even more prominent as rovers perform longer traverses (e.g., the MER rovers have driven more than 100 meters in a day).

A primary objective of our work is to use onboard planning, scheduling, and execution techniques to increase utilization of rover resources by enabling the rover to appropriately respond to unexpected problems and to take advantage of unanticipated opportunities. The Closed-Loop Execution and Recovery (CLEaR) system (Fisher, 2002; Estlin, 2005) is intended to run with little communication with ground. It accepts science and engineering goals and creates a rover command sequence (or plan) that respects relevant constraints, while achieving as many goals as possible. The system executes the produced plan by dispatching commands to the rover's low-level control software and monitoring relevant state

information to identify potential problems or opportunities. If problems or new opportunities are detected, the system is designed to handle such situations by using re-planning techniques to add, move, or delete plan activities. Through this work, we have also identified a number of challenges for an onboard planning and execution system to not only produce valid plans, but also promote robust and efficient rover behavior. These challenges include properly interacting with the appropriate rover navigation software, handling uncertainty in state and resource estimations, as well as handling dynamic events, such as new science opportunities.

For the past several years, we have spent significant time testing the CLEaR system on several different rovers in the JPL Mars Yard. We will discuss our scenario designs for this testing and give an overview of the results including a discussion of how the system handled major scenario elements. Our main objectives for testing include simulating situations that might arise in future rover missions, (such as the Mars Science Laboratory or MSL mission, planned for launch in 2009), providing feedback on our approach, and identifying future directions that should be investigated. In the following section we outline some key challenges that we have identified for onboard decision-making software. Next, we present our current system approach and explain how this system fits into a larger rover architecture and other supporting software that contributed to our testing. We then describe several Mars rover scenarios, which were used to test our system on rover hardware, and describe how our system performed during that testing.

Challenges for Onboard Decision Making

Autonomous rovers have the potential for increasing science return by reducing rover idle time, reducing the need for entering *safe-mode*, and dynamically handling opportunistic science events without required communication to Earth. New missions are being designed that will require rovers to support more autonomous endeavors such as long-range traversals, complex science experiments, and longer mission duration. However, autonomy software designers face a number of challenges in providing software to support these types of operations. In this paper, we consider a few key challenges for using planning, scheduling and execution techniques to provide onboard decision-making capabilities.

To generate and/or modify its own command sequence for carrying out a set of science goals, the onboard planning and execution software will need to reason about a rich model of resource and temporal constraints. For example, it will need to predict power consumption of variable duration activities such as downlinks and traverses, keep track of available power levels, and ensure that generated plans do not exceed power limitations. When resources are over-taxed, the rover should be capable of making science/resource trade-offs in an effort to produce the highest science return. The rover will also require execution and monitoring capabilities to carry out the generated plan on the rover platform. An execution system must be capable of commanding the control software, collecting state updates from sensors, monitoring plan behavior, and smoothly handling activity failures or unexpected events.

Over the course of a mission, the rover will be asked to perform a variety of science operations. The number and scope of these operations are typically limited by the rover onboard resources (e.g., power, memory, lifetime of hardware). Thus, science operations may have varying priorities that indicate their overall mission value. Onboard planning and execution software must reason about these priorities and handle newly identified science opportunities (which may be identified through onboard data analysis software) in a dynamic and efficient manner. For instance, the value of newly identified science observations must be weighed against current resource availability and other scheduled activities.

Sequence generation for rover surface missions also raises a number of interesting challenges regarding spatial reasoning capabilities. One of the dominating characteristics of rover operations is traverses to designated waypoints and science targets. This element is especially important in future missions that intend to explore large geographic areas. Onboard planning and execution software needs to coordinate with several levels of rover navigation software to generate an efficient and achievable rover plan. This coordination will likely include querying a path planner for route information, using position estimates to track rover progress, and correctly modifying the plan when navigation and obstacle avoidance software cause the rover to move off the predicted route.

Another predominant challenge in developing onboard autonomy software is dealing with the inherent uncertainty in predicting rover navigation and science operations. The difficulty is compounded by the tight resource and time constraints that a rover typically faces. At the resource and temporal level, the estimation of items such as power, memory and even activity duration can be highly uncertain. Rover missions are directed at exploring unknown planetary terrains. Requirements for traversing these new terrains are hard to predict. For instance, it is unknown what type of sand consistency a rover will be traversing, which can dramatically affect the required duration and power for a traverse. Similarly, the duration and resource requirements for science operations can vary as well. These variations could be simple, such as a lower than expected image compression ratio, or more complex, such as a drilling operation taking more power and time than originally estimated.

Furthermore, at the state level, since rovers lack an absolute positioning system, the uncertainty in the estimate of the rover pose grows with the distance traversed. This growing uncertainty creates a constant source of error in the knowledge of rover pose. The Sojourner rover used dead-reckoning and a single z-axis gyroscope to estimate

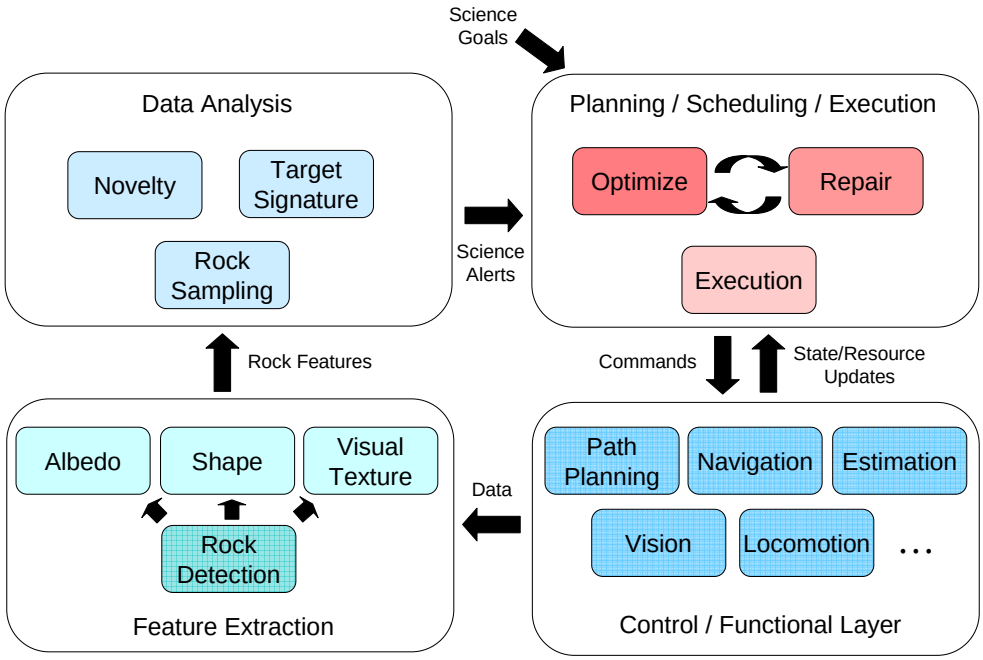


Figure 1: Onboard intelligent decision-making system framework. This framework shows how different decision-making capabilities interact. This paper focuses on the planning, scheduling and execution element of this framework.

rover position, which produced a position error of roughly 5-10% of distance traveled and an average heading drift of 13 degrees per day of traverse (Mishkin, et al., 1999). The MER rovers use more sophisticated techniques to provide position estimation, including a 3-axis gyro and visual odometry. However, these rovers still accrue significant position estimation error and on the ground localization software is often used to recalculate position. Since a large part of a rover schedule consists of rover moves to different locations, the onboard autonomy software must use estimations of position to predict the duration and resource requirements of different operations. If these predications are inaccurate, the autonomy software must be able to continuously modify the schedule to handle the uncertainty in the knowledge of actual rover position.

Planning, Scheduling, and Execution for Rover Operations

To address the issues outlined in the previous section, we have developed a system for high-level decision-making capabilities for future Mars rovers. The overall system framework and data flow is shown in Figure 1. This paper primarily focuses on the planning, scheduling and execution element of this framework, which provides autonomous rover command-sequencing capabilities. Other components will only be briefly described, but are further detailed in related publications.

CLEaR System

In this framework, planning, scheduling, and execution techniques are applied to provide rover-plan generation, execution, and monitoring, and the continuous modification of that plan based on changing operating context and goal information. These capabilities are provided by the CLEaR (Closed-Loop Execution and Recovery) system. CLEaR was developed to pursue a tight integration of planning and execution capabilities. To provide these capabilities, CLEaR closely integrates the CASPER (Continuous Activity Scheduling, Planning, Execution and Re-planning) continuous planner and the TDL (Task Description Language) executive system, which are described further below.

In our system framework, CLEaR handles the following functionality:

- Creating an initial plan based on an input set of goals
- Maintaining resource, temporal and other rover operability constraints
- Executing a plan by interacting with basic rover control functionality, such as navigation, pose estimation, locomotion and stereo vision
- Monitoring plan execution to ensure plan objectives are met
- Dynamically modifying the current plan based on plan activity, state and resource updates
- Performing plan optimization to reason about soft constraints and goal priorities

Figure 2: Sample rover plan displayed in planner GUI. Plan activities are shown in upper portion of window, where bars represent the start and end time of each activity. State and resource timelines are shown in bottom portion of the screen and show the effects of the plan as time progresses. Time is depicted as advancing from left to right.

- Handling newly identified science goals (called *science alerts*) that are generated through onboard data analysis

Planning in CLEaR is provided by the CASPER continuous planning system (Chien, et al., 2000). Based on an input set of science goals and the rover's current state, CASPER generates a sequence of activities that satisfies the goals while obeying relevant resource, state and temporal constraints, as well as additional flight rules. Plans are produced using an iterative repair algorithm that classifies conflicts and resolves them individually by performing one or more plan modifications. CASPER also monitors current rover state and the execution status of plan activities. As this information is acquired, CASPER updates future-plan projections. This update may cause new conflicts and/or opportunities to arise, requiring the planner to re-plan in order to accommodate the unexpected events. An example of a rover plan displayed in the CASPER GUI is shown in Figure 2.

The executive functionality in CLEaR is performed by the TDL executive system (Simmons and Apfelbaum, 1998). TDL was designed to perform task-level control for a robotic system and to mediate between a planning system and low-level robot control software. It expands abstract tasks into lower-level commands, executes the commands, and monitors their execution. It also provides direct support for exception handling and fine-grained synchronization of subtasks. In CLEaR, TDL also handles relaying appropriate activity and state data to CASPER, so that CASPER can adjust its plan accordingly. TDL is implemented as an extension of C++ that simplifies the development of robot control programs by including explicit syntactic support for task-level control capabilities. It uses a construct called a task tree to describe the tree structure that is produced when tasks are broken down into lower-level commands.

One of CLEaR's primary objectives is to provide a tightly coupled approach to coordinating goal-driven and event-driven behavior. Many past approaches have followed a three-level architecture style where the planning and executive processes are treated as *black box* systems. This is in contrast to how CLEaR enables the planner and executive to interact with each other and more effectively share the responsibility for decision making. In part this is managed through shared plan information and continual updates of state being made available to both the planner and executive. CLEaR also provides heuristic support for deciding when certain plan conflicts should be handled by the planner vs. the executive. For instance if a rover gets off track during a traverse, the reaction of the planner and executive need to be coordinated. If the executive believes it can resolve the navigation delay within the planned time constraints it will manage the plan changes. However, once the executive identifies that the repair will require more time or resources than allotted by the planner, it will then fail the task, which will result in the planner using its global perspective to fix the problem.

Currently, CLEaR has a separate planner and executive and thus does share similarities to other three-layer architecture approaches. However, as compared to these approaches where planning is typically done in a batch fashion and takes on the order of minutes to hours, this integration uses a continuous planning approach, where plans are updated and repaired in a matter of seconds. This enables CLEaR to use planning techniques at a finer timescale for tracking the progress of plan execution, quickly identifying potential problems in future parts of the plan, and responding accordingly. As we expect minor portions of the plan to change frequently, we use a lightweight plan runner to dispatch activities to the executive a few seconds before the task's scheduled *start time*. This approach differs from the more common batch approach of turning the entire plan over to the executive for execution. Executive techniques are then used in reactive situations or at times where procedural reasoning is preferred (e.g., using a looping construct to represent the act of trying to grasp a rock, which may need to be repeated several times).

Another way that CLEaR differs from previous approaches is in how the delegation between the planner and the executive is managed. We have primarily taken a planning centric approach to this management. The planner handles the decision of when an activity should be sent to the executive as well as when to perform re-planning. Once the planner has mapped a planning activity to an executive task for execution, control over that one task is given to the executive. The executive may then perform further task expansions as a result of updates and/or exception handling. The executive also provides task completion status back to the planner by either marking an activity as completed or failed. A task is marked as completed when the executive decides the task has met its objective, or marked as failed if the executive concludes that relevant constraints cannot (or even might not) be met. The re-planning process is driven by applying and propagating updates to the plan, and then taking corrective actions to address any conflicts or opportunities that may arise. Re-planning can also be performed synchronously with any already executing task.

Problem Recovery

Due to the uncertainty of science and especially drive operations, a number of things can go wrong in the currently executing rover plan. Recovering from problems or other plan failures was the first general area to which we applied the CLEaR system. One type of plan problem this system addresses is over-subscription of resources (such as power, memory) or time. A number of elements can contribute to this problem, including terrain variability, rover hardware degradation and data compression inconsistencies. If plan activities end up taking additional resources or time during execution, the onboard system must ensure that critical activities that occur later in the plan will still be able to correctly execute. To handle these situations, CLEaR continually tracks the state of the plan including activity completion times and resource usage. If problems (or conflicts) are introduced into the plan based on recent state data, CLEaR can move or delete activities to accommodate the changes. For example, if more power is used on a drive than expected, the system may need to delete several low priority science goals to ensure that enough power will be available for an end of day communication with Earth.

Another problem that CLEaR addresses is when obstacles in the terrain require plan changes to successfully visit goal targets. CLEaR currently employs a set of TSP heuristics to order science target visits in an optimal fashion. However terrain information may be incomplete when an initial plan is generated. During a rover mission, images from the rover navigation cameras can be used to build navigation and obstacles maps, however obstacles can be missed due to poor stereo or obstructions in the rover's sight path. If during plan execution, CLEaR determines that the drive to a particular target is running significantly behind schedule, it may re-evaluate the current target ordering to determine if a new, more optimal ordering can be found or if targets need to be deleted. This re-evaluation can also use any new terrain map information that may have been gathered from the rover's current position.

Science Alerts

To handle opportunistic science, we extended CLEaR to recognize and respond to *science alerts*, which are new science opportunities detected by onboard science-data analysis software. For example, if a rock is detected in navigation imagery that has a previously unseen texture or shape, a science alert may be generated to take additional measurements of that rock. Currently, science alerts can have different levels of reaction from the CLEaR system. The most basic reaction is to adjust the rover plan so that the rover holds at the current position and the flagged data is sent back to Earth for further analysis at the next communication opportunity. The next level of reaction is to collect additional data at the current site before transmitting back to Earth. Further steps include having the rover alter its path to get closer to objects of interest before taking additional measurements. These operations would provide new data that could not be obtained through analysis of the original image.

Plan Optimization

To reason about goal priorities and other soft constraints we used the CASPER optimization framework to continually search for a higher quality plan. User-defined preferences are used to compute plan quality based on how well the a plan satisfies these preferences. Optimization proceeds similarly to iterative repair. For each preference, an optimization heuristic generates modifications that could potentially improve the plan score. A modification is then selected and applied to the plan. After a set number of

iterations, the plan with the best score is selected to replace the current plan.

One key area where plan optimization was used was to take advantage of extra time or resources in the schedule. Since traverse times and rover resource usage are difficult to predict, it is often the case that a rover operation takes less time or power than expected. For instance, a traverse could take much less time than expected due to a benign terrain. For these cases, the optimization framework was used to dynamically add additional science goals to the plan that could not be fit in the original plan due to time and resource constraints. This capability enables the scenario where scientists on the ground specify a number of prioritized science goals, but not all of them may be achievable due to limited rover resources. However, some goals may be fit into the plan as time progresses due to resource usage being lower than predicted.

CLEaR also uses the optimization framework to decide how to respond to science alerts. Because it may not be possible to accommodate all alerts, a science alert is represented as an optional goal, which indicates its achievement is not mandatory but may improve the plan's optimization score if included in the plan. Before attempting to handle a science alert, CASPER protects the current plan by saving a copy before optimization. If the quality has not increased after a certain time limit, the previous plan is restored. If CASPER can handle a new science alert (e.g., by adding additional science measurements) without causing other negative affects, such as resource over-subscriptions or the deletion of ground-specified science goals, then the new plan that accommodates the science alert is used.

We created a set of plan modification functions that are invoked when the optimizer attempts to satisfy a science alert. How the plan is modified depends on the type of alert that is considered. When a science alert is received that requires holding at the current position until data is communicated with earth (called a *stop and call home* alert), the planner alters the plan to remove any non-engineering critical activities and wait for the next communication opportunity. If activities are currently executing, the planner requests the executive component of CLEaR to abort them. If activities are scheduled in the future, the planner deletes them and resolves any inconsistencies created by these deletions.

To handle a science alert that requests additional measurements (called a *data sample request* alert), the planner must generate a plan that achieves the new goals without deleting existing activities or causing conflicts that cannot be resolved (e.g., scheduling more activities than can be executed in a certain time window). To handle a data sample request, the planner must be able to add a new science observation and a new move command to correctly place the rover in position to take the observation.

Science Data Analysis

The Feature Extraction and Data Analysis modules, shown in Figure 1, are responsible for onboard science alert generation. Together with the planning and scheduling component, these capabilities comprise the OASIS onboard science system (Castano, et al., 2006). OASIS enables the rover to perform onboard analysis of collected science data and to trigger science alerts if interesting science opportunities are detected. For instance, if a rover is performing a long traverse, OASIS can analyze navigation images as they are taken to search for interesting rocks or other terrain features that the rover is passing.

As shown in Figure 1, new science data is first processed by the Feature Extraction component. Currently, we have focused on analyzing rocks (and other terrain data) within image data, but plan to expand to other types of data, such as spectrometer measurements. Images are broken down by first locating individual rocks, and second, by extracting a set of rock properties (or features) from each identified rock. Extracted rock properties (e.g., shape, albedo, visual texture) are then passed to the Data Analysis component of the system. This component consists of different prioritization algorithms, which analyze the data by searching for items such as rocks with features that match pre-known signatures of interest (identified by scientists on Earth), or novel rocks (i.e., outliers) that have not been seen in past traverses. If the analysis component detects new science opportunities of significant interest, it will generate a *science alert* that is sent to the planner.

CLARAty Robotic Architecture

The planning, scheduling, and execution component is also integrated with the Coupled Layered Architecture for Robotic Autonomy (CLARAty) (Nesnas, et al., 2006). CLARAty was developed at JPL to simplify the integration and testing of different robotic technology software on multiple hardware platforms and it provides a large range of basic robotic functionality. Through CLARAty, the CLEaR system has been tested with several JPL rover platforms, including Rocky 7, Rocky 8, and FIDO, which are shown in Figure 3.

To run realistic scenarios with rover hardware, a number of supporting pieces of software were used. These components were provided through CLARAty and could run on the relevant JPL rover platforms. This software includes the Morphin navigation system (Urmson, et al., 2003), which enables the rover to avoid obstacles and navigate to specified waypoints, a position estimation algorithm, which integrates IMU (Inertial Measuring Unit) measurements with wheel odometry to estimate rover position and attitude (roll, pitch and heading), and other software that provide mobility and stereo processing.

Figure 3: Rocky 8 rover (left), FIDO rover (middle), Rocky 7 rover (right)

System Testing

To evaluate our system we have performed a large number of tests both in simulation and using rover hardware in the JPL Mars Yard. These tests covered a wide range of scenarios that included the handling of multiple, prioritized science targets, limited time and resources, opportunistic science events, resource usage uncertainty causing under or over-subscriptions of power and memory, large variations in traverse time, and unexpected obstacles blocking the rover's path.

Our testing scenarios typically consisted of a random number of science targets specified at certain locations. A map was used that would represent a sample mission-site location where data would be gathered using multiple instruments at a number of locations. Figure 4 shows a sample scenario that was run as part of these tests. This particular map is of the JPL Mars Yard. The pre-specified science targets (shown in Figure 4 as the larger circles) represented targets that would be communicated by scientists on Earth. These targets were typically prioritized and for most scenarios, constraints on time, power or memory would limit the number of science targets that could be handled. A large focus of these tests was to improve system robustness and flexibility in a realistic environment. Towards that goal we used a variety of target locations and consistently selected new science targets and/or new science target combinations that had not been previously tested.

A primary scenario element was dynamically identifying and handling opportunistic science events. For these tests, we concentrated on finding rocks with a particular target signature, which was described through specifying a target rock albedo level and shape. If rocks were identified in hazard camera imagery that had a certain interest score, then a science alert was created and sent to the planner. Science alerts would typically come in during rover traverses to new locations, but it was also possible for them to come in while the rover was at a science target location due to a small lag caused by image processing time. If a science alert was detected, the planner attempted to modify the plan so an additional image of the rock of interest would be acquired. A sample image that was taken in response to a science alert for a rock with low albedo (i.e., light colored) is shown in Figure 5.

Other important scenario elements included adding or deleting ground-specified science targets based on resource under or over-subscriptions. For instance, in some tests, the rover covered distances more quickly than expected and the planner was able to add in additional science targets that could not be fit into the original plan. Conversely, in other tests, the rover used more power than expected during traverses or science activities, which often caused a power over-subscription, where enough power was not being preserved for later plan activities. The planner resolved this situation by deleting some lower priority science targets. Unexpected energy drops during a traverse could also be handled by the executive, which detects the shortfall and stops the current traverse if there is not enough energy to complete it. In all cases, the planning and execution system attempts to preserve as many high priority science targets as possible while still adhering to required resource and state constraints.

Testing in Simulation

Since testing with rover hardware can be an expensive and time-intensive process, we ran a large number of tests in simulation using a relatively simple simulator. This simulator could execute rover sequence commands and simulate their effects at a coarse level of granularity. For instance the simulator handled items such as rover position changes and energy usage over straight-line movements, but did not simulate obstacle avoidance or rover kinematics. Another capability that was used in simulation was triggering multiple science alerts at pre-set or random times. This capability helped in evaluating the planner's capacity to correctly handle different opportunistic science scenarios.

To easily run and evaluate large numbers of tests, we also invested in a testing infrastructure, which allowed tests to be run offline and statistics automatically gathered, including information such as number of plan conflicts found and resolved, plan generation and re-planning time, number of goals satisfied, overall plan traverse distance and plan optimization scores. This testing infrastructure also enabled the automatic creation of mpeg movies that showed plan changes using snapshots of a plan visualization tool. This tool showed the results of plan generation and execution on an overhead map of the

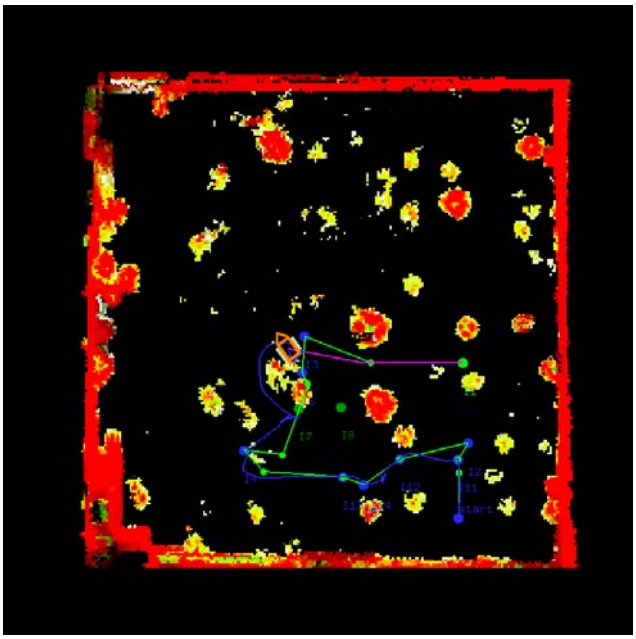


Figure 4: Sample plan shown in the Grid Visualization Tool (GriViT). Green lines show the planned path of the rover; blue lines shown the real path; and pink lines show the path that is currently executing.

world, and could be used for both simulated and hardware testing. An example plan snapshot displayed by this tool is shown in Figure 4. Planning and execution results were evaluated by examining gathered statistics and by viewing created mpegs to flag incorrect or non-optimal behavior.

Testing with Rover Hardware

In addition to testing in simulation, a large number of tests were run in the JPL Mars Yard (shown in Figure 6) using different rover hardware platforms. For the past several years the FIDO rover (shown in Figure 3) was used for the majority of tests. FIDO is an advanced technology prototype rover similar to the Mars Exploration Rover (MER). FIDO's mobility sub-system consists of a six-wheel rocker-bogie suspension capable of traversing over obstacles up to 30 cm in height. All demonstrated software has been designed to run onboard the rover, however during testing, only functional-level CLARAty modules, such as navigation and vision, and the OASIS rockfinding software were run onboard FIDO. Other modules, including the planning and execution module and the analysis module, were run on offboard workstations that communicated with the rovers using wireless ethernet, since a port of these components to the onboard operating system (VxWorks) was not complete.

Tests in the Mars Yard typically consisted of 20-50 meter runs over a 100 square meter area with many obstacles that cause deviations in the rover's path. Science measurements using rover hardware were always images,

Figure 5: Sample image that was taken in response to a science alert for a rock of low albedo using the JPL FIDO rover.

since other instruments were not readily available (e.g., spectrometer). However different types of measurements were included when testing in simulation.

Testing in simulation and with real hardware provided important steps in the evaluation of our system. Many bugs were caught early through simulated testing, but others did not surface until significant runs had been performed on rover hardware. Furthermore, running with hardware often allowed a perspective that was difficult to attain through simulated testing. For example, the accuracy of rover turns towards new science opportunities was much easier to judge when running with hardware.

Related Work

A number of planning and executive systems have been successfully used for robotic applications and have similarities to the approach we describe in this paper. Most of these approaches have used some combination of planning and execution, however they differ in not only the behavior of these individual components, but also in how these systems interface with each other and with other system modules.

The Autonomous Sciencecraft Experiment (ASE) (Chien, et al., 2005) has demonstrated the capability of planning and data analysis systems to autonomously coordinate behavior of the EO-1 Earth orbiting satellite. ASE can also detect and respond to new science events, however it uses very different detection and analysis algorithms. The Remote Agent Experiment (RAX) (Jonsson, et al., 2000) was flown on the NASA Deep Space One (DS1) mission. It demonstrated the ability of an AI planning, execution and diagnosis system to respond to high-level spacecraft goals by generating and executing plans onboard the spacecraft. However, RAX did not incorporate data analysis to identify new science targets

Figure 6: The JPL Mars Yard with terrain of various difficulties.

and used a batch approach to planning. Furthermore, since RAX and ASE were applied to spacecraft, neither handle issues associated with the uncertainty of surface navigation.

Another approach directed towards rover command generation uses a Contingent Planner/Scheduler (CPS) that was developed to schedule rover-scientific operations using a Contingent Rover Language (CRL) (Bresina, et al., 1999). CRL allows both temporal flexibility and contingency branches in rover command sequences. Contingent sequences are produced by the CPS planner and then are interpreted by an executive, which executes the final plan by choosing sequence branches based on current rover conditions. In this approach, only the executive is onboard the rover; planning is intended to be a ground-based operation. Since only a limited number of contingencies can be anticipated, our approach provides more onboard flexibility to new situations. In the CRL approach, if a situation occurs onboard for which there is not a pre-planned contingency, the rover must be halted to wait for communication with ground.

Other similar approaches include Atlantis (Gat 1991), 3T (Bonasso, et al., 1997), and a robotic control architecture developed at the LAAS-CNRS lab (Alami, et al., 1998), which uses the IxTeT planner. These approaches all use a deliberative planner and an executive (or sequencing component) on top of a set of reactive controllers. These approaches have distinctly separate planning and execution techniques, have not closely interacted with navigation software used for rover missions, and are not integrated with onboard analysis system for dynamically identifying new goals.

Future Work

In future work, we plan to extend our capabilities for opportunistic science handling to include adding observations for different types of science instruments and performing close-contact measurements for high priority alerts. We also will extend our system to handle the characterization of larger terrain features and areas that have been identified as important science targets. For instance, our system would handle science operations failing in different fashions such as an unsuccessful data acquisition (e.g., an over-exposed or miss-targeted image frame or an unsuccessful grasping of a rock).

Conclusions

This paper discussed a number of challenges for using planning, scheduling and execution techniques to provide autonomous rover capabilities for future NASA missions. We described our approach for using an onboard decision-making system and explained how it provides capabilities for sequence generation, execution, monitoring, re-planning, sequence optimization, and opportunistic science handling. Through a series of tests in simulation and on rover platforms, we have demonstrated our system's ability to robustly respond to unexpected problems and take advantage of unforeseen opportunities, thus achieving higher utilization of rover resources.

Acknowledgements

This work was performed by the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration.

References

Ai-Chang, M., Bresina, J., Charest, L., Chase, A., Cheng-jung Hsu, J., Jonsson, A., Kanefsky, B., Morris, P., Rajan, K., Yglesias, J., Chafin, B., Dias, W., and Maldague, P. 2004. MAPGEN: Mixed-Initiative Planning and Scheduling for the Mars Exploration Rover Mission. *IEEE Intelligent Systems*. 19:1, pp. 8-12.

Alami, R., Chautila, R., Fleury, S., Ghallab, M., and Ingrand, F. 1998. An Architecture for Autonomy. *International Journal of Robotics Research*, 17(4).

Bonasso, R., Firby, R., Gat, E., Kortenkamp, D., Miller, D., and Slack, M. 1997. Experiences with an Architecture for Intelligent, Reactive Agents. *Journal of Experimental and Theoretical Artificial Intelligence Research*, 9(1).

Bresina, J., Golden, K., Smith, D., and Washington, R. 1999. Increased Flexibility and Robustness of Mars Rovers. *Proceedings of the International Symposium, on AI, Robotics and Automation for Space*, Noordwijk, The Netherlands.

Castano, R., Estlin, T., Gaines, D., Castano, A., Chouinard, C., Bornstein, B., Anderson, R.C., Chien, S., Fukanaga, A., and Judd, M. 2006. Opportunistic Rover Science: Finding and Reacting to Rocks, Clouds and Dust Devils. *Proceedings of the 2006 IEEE Aerospace Conference*. Big Sky, Montana.

Chien, S., Knight, R., Stechert, A., Sherwood, R., and Rabideau, G. 2000. Using Iterative Repair to Improve the Responsiveness of Planning and Scheduling. *Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling*, Breckenridge, CO.

Chien, S., Sherwood, R., Tran, D., Cichy, B., Rabideau, G., Castano, R., Davies, A., Mandi, D., Frye, S., Trout, B., Shulman, S., Boyer, D. 2005. Using Autonomy Flight Software To Improve Science Return on Earth Observing One. *Journal of Aerospace Computing, Information, and Comm-unication.*

Estlin, T., Gaines, D., Chouinard, C., Fisher, F., Castano, R., Judd, M., Anderson, R., and Nesnas, I. 2005. Enabling Autonomous Rover Science through Dynamic Planning and Scheduling *Proceedings of the 2005 IEEE Aerospace Conference*, Big Sky, Montana.

Fisher, F., Estlin, T., Gaines, D., Schaffer, S., Chouinard, C., Knight, R. 2002. CLEaR: Closed Loop Execution and Recovery – A Framework for Unified Planning and Execution. Technology and Science IND News Issue 16, pg. 15-20.

Gat, E., 1991. Integrating planning and reacting in a heterogeneous asynchronous architecture for mobile robots. *SIGART Bulletin 2*, 70-74.

Jonsson, A., Morris, P., Muscettola, N., Rajan, K., and Smith, B. 2000. Planning in Interplanetary Space: Theory and Practice. *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems*, Breckenridge.

Mishkin, A., Morrison, J., Nguyen, T., Stone, H., Cooper, B., Wilcox, B. 1998. Experiences with Operations and Autonomy of the Mars Pathfinder Microrover. *Proceedings of the 1998 IEEE Aerospace Conference*, Aspen, Colorado.

Nesnas, I.A., Simmons, R., Gaines, D., Kunz, C., Diaz-Calderon, A., Estlin, T., Madison, R., Guineau, J., McHenry, M., Shu, I, and Apfelbaum, D. 2006. CLARAty: Challenges and Steps Toward Reusable Robotic Software, International Journal of Advanced Robotic Systems.

Simmons, R. and Apfelbaum, D. 1998. A Task Description Language for Robot Control. *Proceedings of the Intelligent Robots and Systems Conference*, Vancouver, CA.

Urmson, C., Simmons, R., and Nesnas, I. 2003. A Generic Framework for Robotic Navigation. *Proceedings of the IEEE Aerospace Conference*, Big Sky, Montana.

Inducing non-deterministic actions behaviour to plan robustly in probabilistic domains

Sergio Jiménez, Fernando Fernández and **Daniel Borrajo**
Departamento de Informática
Universidad Carlos III de Madrid
Avda. de la Universidad, 30. Leganés (Madrid). Spain
sjimenez@inf.uc3m.es, ffernand@inf.uc3m.es, dborrajo@ia.uc3m.es

Abstract

In the probabilistic track of the last International Planning Competition two main approaches were used, Markov Decision Processes (Boutilier, Dean, & Hanks 1998) and decision-theoretic planning (Blythe 1999). Both approaches use a domain representation with an explicit definition of the probabilities of the actions effects. But when planning in realistic domains, most of the times, the non deterministic effects and the probabilities associated to them are unknown or hard to be obtained accurately. In this paper we present the LUCK architecture (Learning Uncertainty information as Control Knowledge). This architecture plans to solve problems in probabilistic domains using an initial deterministic domain representation. Then, it learns information about the success and the failure of the actions applying Inductive Logic Programming Techniques. And, finally, it uses this information to generate better plans (in terms of robustness) in the future.

Introduction

In this paper we present the LUCK architecture. This architecture solves problems in probabilistic domains planning from a deterministic representation of the domain and learning knowledge about the reasons that cause the execution of actions to be a success or a failure. This knowledge is translated into Control Knowledge so that the planner can reason about the uncertainty of the plans. The main two contributions of this paper are:

- Our system does not reason explicitly about a probabilistic description of the domain in the searching process. The knowledge related to the uncertainty of the world is modelled through Control Knowledge, which is separated from the domain model. So, in the search tree, there is no probabilistic knowledge.
- The probabilistic information of the domain is represented as Control Knowledge. This Control Knowledge is generated automatically from the experience using Inductive Learning Programming Techniques. Specifically we use the ALEPH system that induces theories that explain when an action will succeed or fail according to the current state.

In the last International Planning Competition (IPC4[1]) practically all the planners that took part in the probabilistic track were solving MDPs based on planners. Among these planners there were only one using machine learning techniques to deal with probabilistic domains. This planner implements the ideas explained in (Fern, Yoon, & Givan 2004) to find policies specifying what action to take given a goal and a current state. And in the whole competition only the Probapop system, a conformant probabilistic planner (Onder, Whelan, & Li 2004), generates plans instead of policies given a probabilistic planning problem.

Another different approach to plan in non deterministic domain is planning based on a model checking approach (Kabanza, Barbeau, & St.-Denis 1997). This kind of systems is able to deal with complex goals as they use temporal model logic to describe them and don't need to know the probabilities of the effects of the actions. But they need a non-deterministic description of the domain, and again they try to find policies specifying what action to take given a goal and a current state.

These kinds of policies cannot easily be communicated to humans and typically is a difficult task to transfer this knowledge to other similar problems. In our approach, we don't learn policies but explanations of the causes that lie behind the success or failure of the actions. So the domain designer can get feedback information about the world dynamics from the execution of plans.

This fact also motivates the work (Cocora *et al.* 2006). In this paper is described how to generate Relational Markov Decision Processes (RMDPs) (Kersting, Otterlo, & Raedt 2004) and how to learn policies for these RMDPS using relational decision trees which are relational declarative representations of the policies. But this work is not precisely comparable with ours as it starts from a set of example plans, whereas our system starts from a deterministic domain description of the world.

There is previous work by Karen Haigh (Haigh & Veloso 1998) very related to ours, in which Control Knowledge is learned from the experience of executing actions to improve the planning process. In her work, the attributes that affect the execution of actions are known "a priori". In our work, we do not know "a priori" which attributes from the

[1] http://ipc.icaps-conference.org/

state are relevant with respect to the execution of actions. Reinforcement Learning approaches (Kaelbling, Littman, & More 1996) can also be considered as related work as they also repeat cycles of planning, acting and learning. However, since our approach is based on deliberative planning, it is able to reason with a richer representation of the domain. Thus, we can solve more flexibly problems with different goals and infinite number of potential states.

Instead of learning policies to achieve particular goals, Kaelbling et al. proposed to learn plan operators. In (Pasula, Zettlemoyer, & Kaelbling 2004) and in (Zettlemoyer, Pasula, & Kaelbling 2005), they described how to learn completely the actions model from examples. Our approach is different, since we initially have a deterministic representation of the domain and we only want to learn when the execution of an action is going to succeed or fail. From a planning point of view we believe that designing and maintaining a deterministic domain is a simpler task for the users. So, separation of domain knowledge from probabilistic knowledge can benefit in the process of generating applications, since declarative Control Knowledge can also be accessed.

The rest of the paper is organized as follows: first, we present the general architecture of the system and the planning, execution and learning processes. Next, we describe the experiments carried out to evaluate the architecture, and finally we discuss some conclusions.

The planning, acting and learning cycle

LUCK is an architecture that integrates planning, execution and learning. Figure 1 shows a high level view of its architecture. When LUCK faces a planning problem, it first proposes a plan to solve it, and then tries to execute the plan actions one by one. While LUCK executes actions it observes the results of these executions. When the execution of an action is a failure, LUCK plans to obtain a new plan that solves the problem from this current state. LUCK considers the execution of an action a failure when the new state resulting from the execution of this action is different from the state expected, according to its deterministic representation of the domain.

Initially, LUCK proposes plans to solve the first problem only taking into account the deterministic description of the domain. And, as it starts to observe the results of executing the actions in the real world, it will generate Control Knowledge that will guide it towards solutions that consider the uncertainty in the domain.

Planning

For the planning task we have used the non-linear backward chaining planner IPSS (Rodrguez-Moreno *et al.* 2004), based on PRODIGY4.0 (Veloso *et al.* 1995). The inputs to the planner are the usual ones in planning (domain theory and problem definition), plus declarative Control Knowledge, described as a set of Control Rules. The output of the planner, as we have used it in this paper, is a totally-ordered plan. The Control Rules act as domain dependent heuristics. They are one of the main reasons why we have used this planner, given that they provide a declarative representation of Control Knowledge.

The IPSS planning-reasoning cycle involves as 'decision points': choose a goal from the set of pending goals and subgoals; choose an operator to achieve the selected goal; choose the bindings to instantiate the chosen operator; and apply an instantiated operator whose preconditions are satisfied or continue subgoaling on another unsolved goal. The default decisions at all these decision points can be directed by Control Rules in order to guide the planner. In our approach, initially, the planner is executed without any Control Rules.

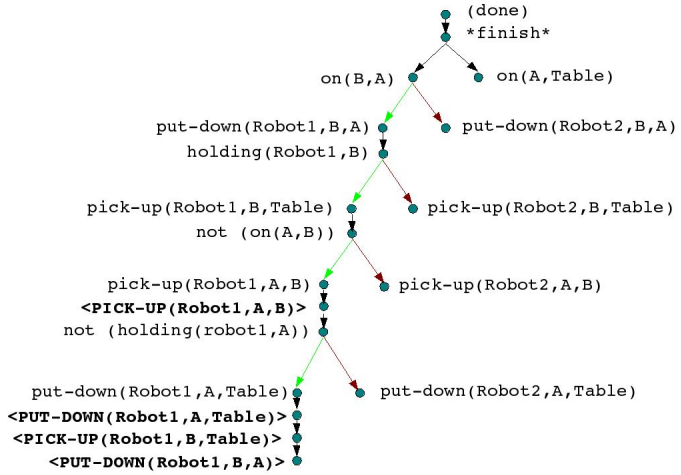


Figure 2: Example of a IPSS planner search tree.

Figure 2 shows a search tree where all the bindings decisions has been directed to prefer the Robot1. This tree belongs to the search process of the planner IPSS solving a problem consisting on reaching the goals `on(B, A)` and `on(A,table)` from a initial state described by the predicates `on(A, B)` and `on(B,table)`.

Actions execution

We simulate the execution of the actions in the non deterministic world. We use the simulator provided by the probabilistic track of the last International Planning Competition, IPC4[2], to evaluate probabilistic planners.

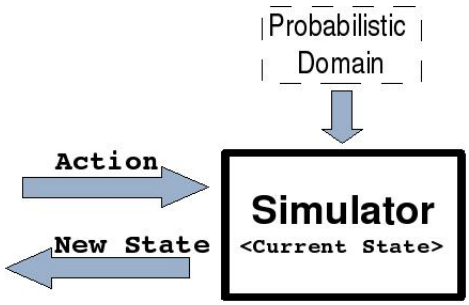


Figure 3: High level view of the simulator module.

[2]http://ipc.icaps-conference.org/

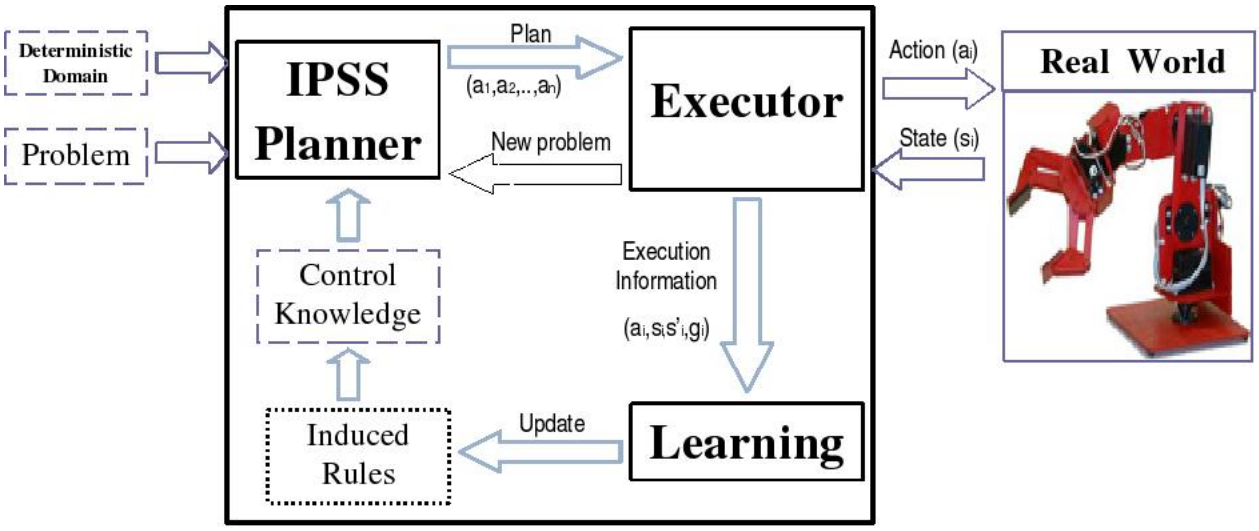


Figure 1: High level view of the LUCK planning-execution-learning architecture.

This simulator uses PPDDL1.0 (Younes & Littman 2004) to describe the world we want to simulate. This language allows us to describe actions with probabilistic and conditional effects. The simulator maintains a representation of the current state and updates it when an action is executed. Figure 3 shows the inputs and outputs of the simulator.

Learning from execution episodes

LUCK uses ILP (Inductive Learning Programming) techniques to analyze the data it obtains from executing actions. The examples used by the inductive learning techniques are tuples of the form: `(action, result, state)`, where `action` is the name of the executed action; `result` is the result of the action execution, that is `success` or `failure`, considering an action execution to be a failure when the new state caused by this action is different from the expected one according to the deterministic representation of the domain; and `state` is the current state when the action was executed. Figure 4 shows an execution step in a probabilistic blocksworld.

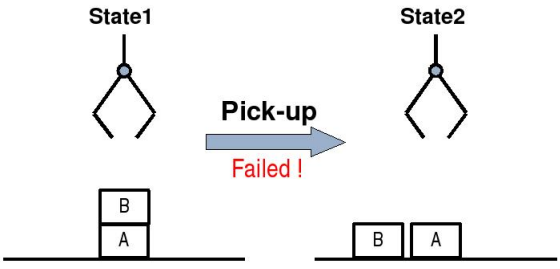


Figure 4: LUCK considers the execution of the action `pick-up-block-from(Robot1, B, A)` a failure.

For every different action of the domain, LUCK maintains the examples of success and failures of execution. From these examples, LUCK induces two different kinds of theories: theories about why the actions succeed and theories about why the actions fail.

To induce these theories, we use ALEPH [3] (A Learning Engine for Proposing Hypotheses). The ALEPH system is based on the Stephen Muggleton's ideas of inverse entailment (Muggleton 1995). This system proposes hypothesis, PROLOG programs, that cover a set of examples described using first order predicates. It can deal with noisy data (Dzeroski & Bratko 1992). And, it also can take as input PROLOG programs describing background knowledge that assists the induction process.

ALEPH receives three inputs:

- The Background Knowledge, which contains PROLOG clauses that encode information relevant to the domain. In our case, which are the types of the planning domain (blocks, robots, ...), which are the predicates that describe the examples (on, clear, ...), and which is the target concept to be learned (success-put-down-block-on, failure-put-down-block-on, ...).

  It also contains PROLOG clauses that encode the information obtained from the executions of actions, which are the learning examples. Figure 5 shows a learning example associated to the action `pick-up-block-from` for the state1 represented in Figure 4.
- The positive examples, that is a set of ground facts representing the positive learning examples of the concept to be learned.
- The negative examples which is a set of ground facts representing the negative learning examples of the concept to be learned.

The ground facts from the positive and negative examples have all the same appearance. They are PROLOG facts of the form

[3] http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph

```
type-of-object(robot1,robot).
type-of-object(blockA,block).
type-of-object(blockB,block).
true-in-state-dirty(example1,robot1)
true-in-state-on-top-of(example1,blockB,blockA).
true-in-state-on-top-of(example1,blockA,table).
```

Figure 5: A learning example for the operator `pick-up-block-from`.

```
target-concept (example-id, parameters).
```

Where `target-concept` is the concept to be learned, `example-id` is a number to link the positive example to the associated leaning example, and `parameters` are the arguments of the operator. So, a ground fact from the positive or negative examples could be:

```
success-pick-up-block-from(example14,robot2,block4,table)
```

The output of ALEPH is a theory, a set of PROLOG clauses, that tries to cover as many positive learning examples as possible, covering the less number possible of negative examples . In LUCK this output represents a set of rules that will try to explain when an action succeeds and when it fails.

```
[Rule 1]
[Pos cover = 5 Neg cover = 27]
failure-pick-up-block-from(A,B,C,D) :-
    true-in-state-dirty(A,B).
```

Figure 6: Rule induced by AELPH to learn the concept of `failure` for the operator `pick-up-block-from`

These induced rules are composed of:

1. The number of positive and negative examples that the rules covers. That gives us an estimation of the rule credibility.
2. The head of the rule, that is target concept.
3. The body of the rule, predicates that describe when the target concept is true.

As an example, Figure 6 shows a rule that is part of a theory induced by ALEPH associated to the action `pick-up-block-from` to learn the concept of `failure`.

This rule means that when LUCK tries to `pick-up` a `block` using a `dirty robot-arm` from the `table` it is going to fail in about 15% of the times.

$$15\% \simeq Positives/(Positives + Negatives)$$

For each operator in the domain two sets of rules like the one in Figure 6 are generated: one set to explain the concept of success and other set to explain the concept of failure. Finally, LUCK is ready to use this information induced from the experience to tell the planner how to generate plans that consider the uncertainty.

Using the learned experience

LUCK uses the induced theories described in the previous section to automatically generate Control Rules that guide the planner in the decision points of the search tree. The *if-part* of the Control Rules is composed of the set of conditions that have to be satisfied in a given node of the search tree in order to fire that Control Rule. These conditions refer to aspects of the planner search process, such as what the current state is, in what goal the planner is working on, or what operator can achieve the current goal. The *then-part* of the rules describes what decision the planner should make in a 'decision point'.

```
(control-rule prefer-bindings-pick-up-block-from
 (IF
 (and
  (current-operator pick-up-block-from)
  (generate-best-binding-pick-up-block-from <best-binding>)))
 (THEN prefer bindings <best-binding>))
```

Figure 7: Example of control rule.

For the time being LUCK only generates Control Rules to guide the planner in decision points for choosing bindings. It generates automatically a Control Rule for every operator in the domain to choose the best bindings for that operator. Figure 7 shows an example of one of these Control Rules automatically generated by LUCK for the operator `pick-up-block-from`.

All these Control Rules have the same structure. They have two metapredicates:

- `(current-operator op)`. To fire the rule only when the current operator in the search tree is `op`.
- `(generate-best-binding-op <best-binding>)`. This metapredicate acts as a generator and it sets the variable `<best-binding>` to the best bindings for the current operator. The COMMONLISP code of this metapredicete is automatically generated taking into account the theories induced by ALEPH.Figure 8 shows the pseudocode of the metapredicate `generate-best-binding` for the operator `pick-up-block-from`.

generate-best-binding-pick-up-block-from (best-binding)

best-binding: The best bindings for the operator

B ← generate-possible-bindings(pick-up-block-from);
P ← Initialize-bindings-probabilities();
RS ← get-success-rules(pick-up-block-from);
RF ← get-failure-rules(pick-up-block-from);
For each possible binding b_i in B
  When b_i matches a success rule rs_j,
    increase probability of choosing b_i with reliability of rs_j
  When b_i matches a failure rule rf_j,
    decrease probability of choosing b_i with reliability of rf_j
best-binding ← choose-best-binding(B,P);
Return best-binding;

Figure 8: generate-best-binding pseudocode

First, the metapredicate generates all the possible bindings for the operator. Second, it associates a probability value to every possible binding. This probability is initially zero for all possible bindings. Then, it selects a binding, and tests if a success rule can be fired with it. If so, it increases the probability of choosing it considering the reliability of the rule. We are currently studying different ways of updating it. The initial method we are considering is just adding previous probability with the rules reliability. However, we will explore using no-regret techniques in order to update it. Then, it tests if a failure rule can be fired with this binding. If so, it decreases the probability of choosing the binding. This is done for all the possible bindings. And, finally, the function returns the best binding according to these probabilities.

Currently this function returns always the binding with the greater probability value. We are planning to use roulette mechanisms in the near future for selecting the best binding. But at this point we can consider using different exploration/exploitation strategies or even algorithms that also take into account the risk of executing actions.

Experimental Results

We have carried out experiments to evaluate the behaviour of the architecture using a modified version of the blocksworld domain from the probabilistic track of the IPC4. We have introduced three modifications:

- There are two robot arms (instead of one) to handle the blocks. Therefore, the operators `pick-up-block-from` and `put-down-bock-on` have another extra argument indicating the robot that carries out the action.
- There is a new predicate, indicating when a robot arm is dirty.
- When a robot is dirty, actions are going to fail 25% of the times. Obviously, the planner does not know it. It will have to learn it by executing actions.

The learned theories

In this version of the blocksworld domain LUCK has tried to solve ten *5-blocks* problems generated with the random problem generator provided by the probabilistic track of the IPC4. Then, it has generated theories about the success and failure of actions. Figure 9 shows the learned theories by LUCK for the operators `pick-up-block-from` and `put-down-block-on`.

- `Rule 1` means to LUCK that attempts to `pick-up` a block `C` from another block `D` succeeds 92% of the times.
- `Rule 2` means to LUCK that attempts to `pick-up` a block `C` from another block `D` with a dirty robot `B` fails 15% of the times.
- `Rule 3` means to LUCK that attempts to `put-down` a block `C` that is holded by the robot `B` on another block `D` succeeds 87% of the times.
- `Rule 4` means to LUCK that attempts to `put-down` a block `C` on another block `D` with a dirty robot `B` fails 25% of the times.

Induced Rules for `pick-up-block-from`

Success Rules

```
[Rule 1]
[Pos cover = 58 Neg cover = 5]
success-pick-up-block-from(A,B,C,D) :-
    true-in-state-on-top-of(A,C,D).
```

Failure Rules

```
[Rule 2]
[Pos cover = 5 Neg cover = 27]
failure-pick-up-block-from(A,B,C,D) :-
   true-in-state-dirty(A,B).
```

Induced Rules for `put-down-block-on`

Success Rules

```
[Rule 3]
[Pos cover = 42 Neg cover = 6]
success-put-down-block-on(A,B,C,D) :-
   true-in-state-holding(A,B,C).
```

Failure Rules

```
[Rule 4]
[Pos cover = 6 Neg cover = 18]
failure-put-down-block-on(A,B,C,D) :-
   true-in-state-dirty(A,B).
```

Figure 9: Learned theories for the operators `pick-up-block-from` and `put-down-block-on`.

For every operator two theories are automatically generated. A first one describing why an action succeeds and a second one describing when an action fails. In this example all the induced theories have just one rule.

As is done in our current work the estimation of the probability of the rules is not perfect, we explain how we plan to improve this estimation in the conclusions section. For example the induced probability for `Rule 2` should be 25% instead of 15%. When the amount of problems solved by LUCK is bigger, these values will be more accurate. For `Rule 1` and `Rule 3` these values depend on the number of times that these actions are tried with a dirty robot.

The point is that the estimation is not perfect but has information about why the execution of an action fails or succeeds. What allows LUCK to generate Control Knowledge to guide the planning module.

Measuring the quality improvement

To evaluate the worth of the learned theories we have solved a set of twenty-five *8-blocks* problems generated with the random problem generator provided by the probabilistic track of the IPC4. We have made LUCK to solve this twenty-five problem set with and without the acquired Control Knowledge described in the previous section. And we have measure two different magnitudes:

1. The length of the plan executed to solve a problem.
2. The number of failed actions. That is the number of replanning process.

As the success or failure of the actions is non deterministic, to obtain reliable values we have solved 15 times every problem and we have extracted the average values. Table 1 shows the obtained experimental results.

In all the problems, except in problem 12 and in problem 19, the number of failed actions is less or equal planning with the induced Control Knowledge than planning without it. In problem 12 and in problem 19 these values are practically the same (they differ just a little because we are testing in a probabilistic domain). So we can state that planning using the induced Control Knowledge makes LUCK finding more robust plans.

As the plans found using the Control Knowledge are more robust, less replanning process has been needed so the length of the executed plans is also shorter or equal than planning without the induced Control Knowledge.

And also, as the induced Control Knowledge acts as heuristics to the LUCK planning module, it makes LUCK to find solution in problems that couldn't be solved before within a time bound of 30 seconds (problem2, problem8, problem15, problem17, problem24).

Planning Time

To evaluate how the use of the Control Knowledge affects to the planning process we have measured the time that takes the planner to solve the twenty-five problem set. We have solved 15 times every problem and we have extract the average values. Table 2 shows the obtained experimental results.

Problem	Time	Time using CK
Problem1	0.37	0.68
Problem2	Unsolved	0.7
Problem3	0.28	0.61
Problem4	0.77	0.39
Problem5	0.21	0.42
Problem6	2.31	0.53
Problem7	0.73	0.38
Problem8	Unsolved	3.66
Problem9	0.42	1.72
Problem10	2.59	1.5
Problem11	0.36	1.59
Problem12	0.16	0.52
Problem13	0.4	0.46
Problem14	0.36	1.15
Problem15	Unsolved	2.96
Problem16	0.79	2.37
Problem17	Unsolved	1.8
Problem18	0.25	0.95
Problem19	0.78	1.33
Problem20	0.47	0.88
Problem21	0.36	2.1
Problem22	0.81	1.57
Problem23	0.2	1.7
Problem24	Unsolved	2.04
Problem25	1.61	2.705

Table 2: Experimental planning times for a set of twenty-five
8-blocks problems.

On one hand when planning with the Control Knowledge solutions are more robust, less actions fail and less replanning processes are needed so it takes less time to solve a problem. But on the other hand when the number of failures is practically the same planning with and without the Control Knowledge, the time that takes LUCK to solve a problem is a little bit bigger using Control Knowledge. Because planning using Control Knowledge implies making some extra computations in the planner search process that cost time.

Conclusions

In this paper, we present the LUCK architecture for acting in domains with uncertainty. The LUCK system acquires automatically information about the behaviour of the actions and acts according to plans that are obtained using this information.

Initially LUCK only needs a deterministic description of the action model since it handles the uncertainty learning declarative control rules that modify its default deterministic behaviour. We have designed this approach basically for three main reasons:

1. Defining a probabilistic domain for realistic problems is not an easy task. Usually the non deterministic effects of the actions and the probabilities associated to them are unknown or difficult to predict. Our approach automatically learns information that deals with this kind of uncertanity.
2. Leaning completely a domain theory without any kind of bias for a realistic problem is also a hard task (Pasula, Zettlemoyer, & Kaelbling 2004).
3. The induced declarative Control Rules give information understandable for a human about the causes that lie behind the success or failure of the actions

Experimental results reflects that LUCK is able to learn the reasons that causes the success and failure of the actions of a simple probabilistic domain. This information is kept separated from the action theory, represented as Control Knowledge and it is used to generate Control Rules that guide successfully the planning process towards solutions of a better quality.

In this paper the quality of the plans has been interpreted as the robustness of its actions. And the final goal lie in finding solutions to problems using the most robust actions in the minimum amount of steps. As it is done in the IPC4. So the learning process has been focused on the concepts of the failure or success of actions execution. An interesting extension for future efforts is working in domains where some other attribute of the actions has to be learned. For example, the time that takes an action to finish its execution or the cost of executing an action in a given domain.

Our current work lie in using symbolic statistical techniques to estimate the probabilities of the induced rules in an accurately way. Precisely we are working with the programming language for symbolic-statistical modelling PRISM[4]. PRISM perform a Maximum Likelihood estimation of the program parameters from incomplete data by

[4]http://sato-www.cs.titech.ac.jp/prism/

Problem	Failures	Failures using CK	Plan Length	Plan Length using CK
Problem1	1	1	40	40
Problem2	Unsolved	1	Unsolved	38
Problem3	1	1	30	30
Problem4	7.35	1	27.85	16
Problem5	1	1	24	24
Problem6	14.42	1	60.28	30
Problem7	5.71	1	28.75	20
Problem8	Unsolved	1	Unsolved	118
Problem9	1	1	46	46
Problem10	14.5	1	61.57	34
Problem11	1	1	38	38
Problem12	1.35	1.42	12.92	13
Problem13	4.24	1	13	10
Problem14	1	1	36	36
Problem15	Unsolved	1.71	Unsolved	72.71
Problem16	1	1	76	76
Problem17	Unsolved	1	Unsolved	52
Problem18	1	1	20	20
Problem19	1.85	2.07	29.71	29.85
Problem20	4.28	1	16.78	14
Problem21	1	1	40	40
Problem22	6	1	28.92	20
Problem23	1	1	22	22
Problem24	Unsolved	1	Unsolved	32
Problem25	6	2.07	45.5	36.64

Table 1: Experimental plan quality measures for a set of twenty-five *8-blocks* problems.

the Expectation-Maximization algorithm (Sato & Kameya 2001).

At the same time we are also working in how we can use the information learned from the experience not only to help the LUCK planner module choosing the correct bindings but also choosing the correct goals. Therefore, improving the quality of the solutions.

As the IPSS planner is not a fast planner, in the near future we plan to study how Control Knowledge can be learned to guide the search process of more efficient planners such as the heuristic planner FF (Hoffmann 2001) and thus compare our system to the *state-of-the-art planners* that take part in the probabilistic track of the International Planning Competition.

Acknowledgements

This work has been partially supported by the Spanish MEC project TIN2005-08945-C06-05 and regional CAM-UC3M project UC3M-INF-05-016.

References

Blythe, J. 1999. Decision-theoretic planning. *AI Magazine, Summer*.

Boutilier, C.; Dean, T.; and Hanks, S. 1998. Planning under uncertainty: structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*.

Cocora, A.; Kersting, K.; Plagemann, C.; Burgard, W.; and Raedt, L. D. 2006. Learning relational navigation policies. *Kunstliche Intelligenz*.

Dzeroski, and Bratko, I. 1992. Handling noise in inductive logic programming. *Workshop on Inductive Logic Programming, ICOT-TM-1182, Inst. for New Gen Comput Technology, Japan*.

Fern, A.; Yoon, S.; and Givan, R. 2004. Learning domain-specific control knowledge from random walks (draft). *Proceedings of the International Conference in Automated Planning and Scheduling*.

Haigh, K. Z., and Veloso, M. M. 1998. Planning, execution and learning in a robotic agent. *AIPS* 120–127.

Hoffmann, J. 2001. Ff:the fast forward planning system. *AI Magazine, 22(3)* 57–62.

Kabanza, F.; Barbeau, M.; and St.-Denis, R. 1997. Planning control rules for reactive agents. *Artificial Intelligence* 95(1):67–11.

Kaelbling, L. P.; Littman, M.; and More, A. 1996. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*.

Kersting, K.; Otterlo, M. V.; and Raedt, L. D. 2004. Bellman goes relational. In *In Proceedings of the Twenty-First International Conference on Machine Learning (ICML-04)*.

Muggleton, S. 1995. Inverse entailment and Progol. *New*

Generation Computing, Special issue on Inductive Logic Programming 13(3-4):245–286.

Onder, N.; Whelan, G. C.; and Li, L. 2004. Probapop: Probabilistic partial-order planning. *Proceedings of the International Conference in Autommated Planning and Scheduling*.

Pasula, H.; Zettlemoyer, L.; and Kaelbling, L. 2004. Learning probabilistic relational planning rules. *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling*.

Rodrguez-Moreno, M. D.; Borrajo, D.; Cesta, A.; and Meziat, D. 2004. An ai tool for scheduling satellite nominal operations. *AI Magazine*.

Sato, T., and Kameya, Y. 2001. Parameter learning of logic programs for symbolicstatistical modeling. *Journal of Artificial Intelligence Research* 391–454.

Veloso, M.; Carbonell, J.; Prez, A.; Borrajo, D.; Fink, E.; and Blythe, J. 1995. Integrating planning and learning: The PRODIGY architecture. *Journal of Experimental and Theoretical AI* 7:81–120.

Younes, H. L. S., and Littman, M. L. 2004. Ppddl1.0: An extension to pddl for expressing planning domains with probabilistic effects. *Technical Report CMU-CS-04-167, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania.*

Zettlemoyer, L.; Pasula, H.; and Kaelbling, L. 2005. Learning planning rules in noisy stochastic worlds. *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05).*

Learning Behaviors Models for Robot Execution Control

Guillaume Infantes and **Félix Ingrand** and **Malik Ghallab**
LAAS-CNRS
7, Avenue du Colonel Roche, 31077 Toulouse Cedex 4, France
{infantes,felix,malik}@laas.fr

Abstract

Robust execution of robotic tasks is a difficult problem. In many situations, these tasks involve complex behaviors combining different functionalities (e.g. perception, localization, motion planning and motion execution). These behaviors are often programmed with a strong focus on the robustness of the behavior itself, not on the definition of a "high level" model to be used by a task planner and an execution controller. We propose to learn behaviors models as structured stochastic processes: Dynamic Bayesian Network. Indeed, the DBN formalism allows us to learn and control behaviors with controllable parameters. We experimented our approach on a real robot, where we learned over a large number of runs the model of a complex navigation task using a modified version of Expectation Maximization for DBN. The resulting DBN is then used to control the robot navigation behavior and we show that for some given objectives (e.g. avoid failure, optimize speed), the learned DBN driven controller performs much better (we have one order of magnitude less failure) than the programmed controller. We also show a way to achieve efficient incremental learning of the DBN. We believe that the proposed approach remains generic and can be used to learn complex behaviors other than navigation and for other autonomous systems.

Introduction and Motivations

Tasks execution on autonomous robots is a very complex process: exploration rovers are expected to plan and properly execute their science observation missions; museum guide robots have to robustly execute their tour plans for visitors interested in a given subject; service robots must plan and execute daily activities for the elderly people they assist etc. The building blocks of these plans, i.e. the tasks and actions, can be quite complex. We refer to them as behaviors. These behaviors usually have an intrinsic complexity (e.g. navigation for a museum guide robot involve localization, perception, motion planning, etc.). Moreover, often no explicit model exists of how it performs in various environments. Last, even if a deterministic model of these behavior exists, it may not be appropriate to handle the intrinsic nondeterminism of the environment and of the behavior execution outcomes. As a result, one as to cope with a planning problem where one must plan actions execution with poor model, or even, in some situation, with no model at all.

Some approaches [Amir 2005] try to model actions deterministically, and apply a probabilistic bias afterward. This kind of approach may not be relevant for intrinsically stochastic systems, which are more common in real-world domain. Bayesian Networks [Pearl 1988] give to the authors of [Dearden & Demiris 2005] the ability to model the actions and predict the effects of a manipulation task of a robot. But while this kind of model captures the stochastic nature of the system, it does not take into account its long-term dynamics.

Systems may be modeled as a Hidden Markov Model (HMM) [Rabiner 1989]. In these models, the internal state is *hidden* to the observer, who can only see the *observation*, that represent effects of the system on the environment. These stochastic models have proved quite adapted in domains such as speech recognition [Rabiner 1989], modeling human activity [Bobick & Davis 2001; Liao, Fox, & Kautz 2004; Osentoski, Manfredi, & Mahadevan 2004], facial expression recognition [Cohen *et al.* 2003], gesture recognition [Nam & Wohn 1996; Wilson & Bobick 1999; 2001], probabilistic plan recognition [Bui 2003; Bui, Venkatesh, & West 2002] and learning topological and metric maps [Koenig & Simmons 1996b; 1996a]. In [Fox *et al.* 2006], the authors present an approach to models robot actions using HMM. Indeed, the resulting HMM actions can be used to recognize or to plan the modeled actions.

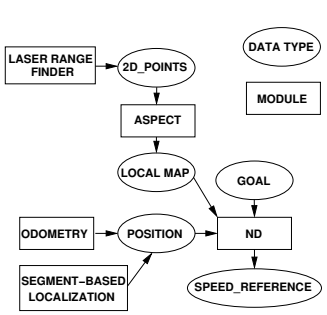
Yet, this representation does not allow the finer control of the action execution. An aspect of complex actions we want to address is that some of these activities may be controllable. Finding the proper parameter values with respect to the current environment and goal can be quite a challenge if again no model of the underlying action is available. Hence we would like to learn action models rich enough to allow us to use them to perform any of recognition, planning, control or supervision of the action. We propose to learn action models as highly structured stochastic processes: Dynamic Bayesian Network (DBN) [Dean & Kanazawa 1990].

We detail how such a model can be obtained from a number of real-world runs. We then show how it can be used to control the action itself while executing. We also sketch how this could be embedded in a more general controller able to supervise the action execution (to avoid failure) and to decide when the model has to be refined for new situations arising. The paper is organized as follow. The next section presents a complex robot behavior to model and to adapt.

We then present how we can learn such a structured stochastic model. The following section presents how we can use the learned model, followed by a section on results obtained on a real robot (i.e. the learning phase as well as controlling the robot using the learned model). We then conclude the paper with a discussion on the current work as well as some perspectives we are currently pursuing.

Modeling and Adapting Robot Behavior

To conduct our experiments, we modeled a robotic navigation task, based on ND[1] reactive obstacle avoidance [Minguez, Osuna, & Montano 2004]. A schematic view of this navigation modality can be seen on figure 1(a). The laser range finder gives a set of points representing obstacles in front of the robot. This points are used to build a local map around the robot by the *aspect* module. ND uses the current position of the robot to compute the local goal in the robot's coordinate system. Then with this local goal and the map around the robot, it computes a speed reference that tends to move the robot towards the goal position while avoiding obstacles. This modality is used on our B21R robot (figure 1(b)).



(a) ND-based navigation modality

(b) Platform

Figure 1: The RWI B21R modified to be a museum guide robot [Clodic *et al.* 2005]

Structure of a Navigation Task

A navigation controller can be seen as a "black box" taking a relative goal position and some data about the environment as inputs and giving as output a speed reference for the robot to execute. We can model this by having an internal hidden state of the robot, with hidden variables. The control parameters of the navigation are observable and have an influence on the internal state. It changes the environment, because it is measured through the sensors of the robot: if the robot goes into a dead-end, the environment will seem very cluttered. The general structure can be seen on figure 2.

Choosing the variables is highly dependent on the process to model. The control parameters are given by the navigation controller, but we must choose wisely environment

[1]Note that despite the fact that ND was partially developed at our lab (over a number of year), it is so intrinsically complex, and so "difficult" to tune, that no model exists which could help us supervise, control and plan its execution.

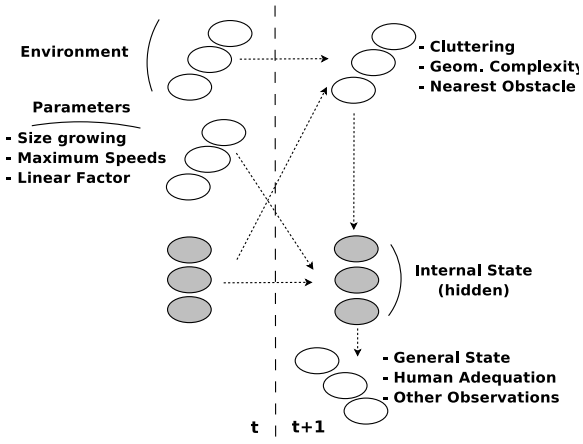


Figure 2: Abstract DBN structure for a navigation task

variables that represent important parameters for decisions made by the control process. In our reactive navigation task, the cluttering of the environment is to be taken into account, so is the closest obstacle, which has much influence on the future sight of the robot. On the other hand, we avoid including the position of the robot, so that the model can be used in places different than the one where the learning was performed.

The effects of the navigation task we need to recognize are the *fail* and *success* states, but we also aim at controlling more precisely the behavior in a qualitative way: some behaviors are successful, but not optimal. We also include as many variables as we can that could give us a hint on the internal state, for the learning process to be more effective. Finally, we could also model resource usage.

Instantiation

For our navigation task, the control parameters are: the two **size growing** parameters of the robot: one surface represents the robot (where obstacles are totally forbidden); and a larger security area where there should be as few obstacles as possible; the **maximum linear and angular speeds** allowed; a **linearity factor** between the two speeds given by the motion generator controller.[2]

The environment variables chosen are: the **cluttering** of the environment, defined as a weighted sum of distances to nearest obstacles around the robot; the **angle of the nearest obstacle**; the **distance to the nearest obstacle** and the global **number of segments** in the scene.

The output observable variables are: the current **linear and angular speeds** of the robot; its current **linear and angular** accelerations; the **variation of the distance to the goal** (estimated as an euclidean distance); the number of possibles ways to go (**valleys**) built by ND; the **achieved ratio** of the mission; the **current strategy** chosen by ND; the **general state** in *begin*, *end*, *fail*, *normal*; the **human ad-**

[2]The linear and angular speeds are mutually constrained by a polynomial function that influences the type of trajectory produced.

equation of the behavior[3].

All the variables are discretized (into clusters from 3 to 6 values). The number of different potentially possible observations is more than 15×10^9. This indicates that even with a very large number of runs, all possible observations will never be seen, leading to many non-informative transition probabilities into the DBN. We will show later how to deal with such a sparse DBN for decision making.

DBN Learning

To learn a DBN, we use an adaptation of the classical Expectation-Maximization (EM) algorithm defined for Hidden Markov Models [Rabiner 1989]. The adaptations to make this algorithm tractable for DBN are not straightforward: we need to maintain an approximated "belief state", i.e. a probabilistic hypothesis over the possible assignment of the values of the variables because they are strongly correlated. We choose to approximate this belief state as a particle filter, with a variable number of particles.

Expectation Maximization Algorithm

A DBN λ is defined by a set of variables (hidden or observable), a set of possible values for each variable, and a set of probabilistic causal links between the variables [Dean & Kanazawa 1990]. An evidence O is as a sequence of observations (i.e. instantiations of all observable variables of λ).

A good model λ with respect to O gives a high $P(O|\lambda)$ (likelihood of the evidence). In the DBN framework, λ may be adapted to the evidence either by modifying its structure or the probabilities attached to the causal links in a given structure. The EM algorithm can deal with hidden variables, but not with structure of the model.

In the HMM case, EM first compute probabilities of being in every hidden state, then every transition probability and every probability of seeing each observation from each hidden state, this for every time step. This is often referred as the Expectation step. For a DBN, the hidden state becomes an instantiation of the corresponding variables, and so does an observation. The state has to be approximated as a *belief state*, i.e. a set of weighted hypotheses over all possible instances. Furthermore, the observable variables are not decoupled from the hidden ones, so we compute an observation probability not only from hidden variables, but from the whole belief state. Then we can update the transition probabilities to maximize $P(O|\lambda)$. Then we go again into Expectation and Maximization steps. This algorithm is proved to reach a local maximum of $P(O|\lambda)$.

Definitions

Let q_i with $i \in [1..N]$ be the N variables of the DBN.

Each variable q_i can take the values S_{i_j}, and $Va(i) = \{S_{i_j}\}$ is the set of V_i different possible values for the variable q_i. Thus we have $V_i = card(Va(i))$.

[3]Some behaviors leads to success, but may be considered as too aggressive for humans around the robot, or on the contrary too shy, and thus to slow. Its value has to be given by the operator during the learning phase.

For a DBN, we can consider a transition not from a value for the hidden state variable to another one like in HMMs, but from a set of parent variables to only one. We note $Pa(i)$ the set of possible instantiations of the causal parents of the variable q_i, and $q_{Pa(i)}$ the parents of the variable q_i. So one instantiation is:

$$\sigma_i = (S_{i_{0_{j_0}}}, S_{i_{1_{j_1}}} ... S_{i_{k_{j_k}}}) \in Pa(i)$$

with $\{q_{i_0}, q_{i_1}, ..., q_{i_n}\} = q_{Pa(i)}$ are the parents of q_i.

A transition probability will then be:

$$a(\sigma_i, i_j) = P(q_{Pa(i)} = \sigma_i, q_i = S_{i_j})$$

with $\sigma_i \in Pa(i)$, and S_{i_j} the j^{th} value that can take the variable q_i.

We denote q_{it} the value taken by variable q_i at time t and O_t the observation made at time step t, so we define the probability of seeing O_t knowing $q_{it} = S_{i_j}$:

$$b_{i_j}(O_t) = P(O_t|q_{it} = S_{i_j})$$

and the probability of seeing O_t knowing $q_{it} = S_{i_j}$ and $Pa(i) = \sigma_i$ at time step t:

$$b_{i_j,\sigma_i}(O_t) = P(O_t|q_{Pa(i)t} = \sigma_i, q_{it} = S_{i_j})$$

We also define the probability of the variable q_i having the value S_{i_j} at time step t, knowing the whole observation sequence $O = (O_1 ... O_T)$, and the model λ:

$$\gamma_t(i_j) = P(q_{it} = S_{i_j}|O, \lambda)$$

In the same way, the probability of a transition occurring at time t is:

$$\xi_t(\sigma_i, i_j) = P(q_{Pa(i)t} = \sigma_i, q_{it} = S_{i_j})$$

The difficulty for the Expectation step is to compute efficiently all γ_t and ξ_t. This is done using the recursive Forward-Backward algorithm.

Forward Backward Algorithm

The Forward Backward algorithm lets us compute the desired probabilities by using a message passing scheme, where the two messages are the probabilities of having variable q_i taking one of its possible values S_{i_j} knowing respectively the beginning of the observation sequence $O_1, .., O_t$ and the end of it $O_{t+1}, .., O_T$.

Forward Message The forward message for one variable is defined as the probability of reaching value S_{i_j} of variable q_i at time t and seeing the start of the observation sequence $O_1 ... O_t$:

$$\alpha_t(i_j) = P(O_1 .. O_t, q_{it} = S_{i_j}|\lambda)$$

We need to propagate recursively this value over t. But as the transitions are from a set of variables to a standalone variable, we have to define the same forward message for an instantiation of variables:

$$\alpha_t(\sigma_i) = P(O_1, ..., O_t, \sigma_i|\lambda)$$

And we compute these values as:

$$\alpha_t(\sigma_i) = [\prod_{i_j \in \sigma_i} \alpha_t(i_j)] B_t(\sigma_i)$$

The coefficient $B_t(\sigma_i)$ (which implicitly depends on the history of observations $O_1...O_t$) is a weight over the different belief states. It is necessary because all σ_i have not the same probability (the components of σ_i are strongly correlated). To do exact inferences, we should compute this $B_t(\sigma_i)$ taking the whole history into account, which would lead to an exponential complexity in t. We chose to approximate this calculus by maintaining a particle filter which gives us an approximation of the current belief state of the process (see [Koller & Fratkina 1998] to have an overview of techniques to approximate the belief state). This belief state gives us a relative weight of the possible instantiations of variables.

Now we simply have (as in HMMs):

$$\alpha_t(i_j) = \sum_{\sigma_i \in Pa(i)} \alpha_t(\sigma_i) a(\sigma_i, i_j) b_{i_j,\sigma_i}(O_t)$$

So we can compute recursively $\alpha_t(i_j)$ for every variables for every time step, starting from the parents at time 1 and propagating the message.

Backward Message The backward message is the probability of seeing the end of the observation sequence $O_{t+1}...O_T$ knowing the value S_{i_j} of the variable q_t at time t:

$$\beta_t(i_j) = P(O_{t+1}..O_T | q_{it} = S_{i_j}, \lambda)$$

To compute this value, we have to look at every child of variable q_i, but we do not have any information about the other parents of these children. Thus we can not take hypothesis on the different instantiations of the parents hence we choose to take an average.

Let $Ch(i)$ be the children of variable q_i. For each child q_k of q_i, we build the set of σ_k where $\sigma_k \in Pa(k)$ and $i_j \in \sigma_k$.

Then we compute the probability of seeing the observation at time t and we multiply it by the probability of seeing the observations after time t. This is:

$$b_{k_l,\sigma_k}(O_t)\beta_t(k_l)$$

We weight afterwards this probability by the probability of having $q_k = S_{k_l}$ from $q_i = S_{i_j}$:

$$a(\sigma_k, k_l) b_{k_l,\sigma_k}(O_t)\beta_t(k_l)$$

where $\sigma_k \in Pa(k)$ and $i_j \in \sigma_k$.

Then we compute the average over all possible instantiations and all possible S_{k_l}. We obtain:

$$F_t(k, i_j) = \sum_{\substack{\sigma_k \in Pa(k) \\ i_j \in \sigma_k}} \sum_{l \in Va(k)} \frac{a(\sigma_k, k_l) b_{k_l,\sigma_k}(O_t)\beta_t(k_l)}{V_k.C_k}$$

where $C_k = card(\{\sigma_k \in Pa(k), i_j \in \sigma_k\})$

Finally, $\beta_t(i_j)$ is the sum over every child of q_i with $q_i = S_{i_j}$ of this average, which gives:

$$\beta_t(i_j) = \sum_{k \in Ch(i)} F_t(k, i_j)$$

Maximization

Now we can compute $\xi_t(\sigma_i, i_j)$ as:

$$\xi_t(\sigma_i, i_j) = \frac{\alpha_t(\sigma_i) a(\sigma_i, i_j) b_{i_j,\sigma_i}(O_t)\beta_t(i_j)}{P(O|\lambda)}$$

So we obtain the new transition probability as:

$$\bar{a}(\sigma_i, i_j) = \frac{\sum_t \xi_t(\sigma_i, i_j)}{K}$$

where K is a normalizing factor.

This step computes a new model λ', and we have $P(O|\lambda') > P(O|\lambda)$. So $P(O|\lambda)$ is maximized by this step. With the new model, we can now go again for an Expectation step, which will lead to different α, β and ξ and then update the model again. This process converges to a local maximum of $P(O|\lambda)$.

Notes on Implementation

The main difficulty on implementing this algorithm is that α and β values tend exponentially fast to zero with t. For Hidden Markov Models, a scaling process has been proposed [Rabiner 1989] to solve this issue. In the DBN case, we need to extend this scaling process to multiple variables. We chose to use a scaling factor for each S_{i_j} of variable q_i and not for each q_i, which seems fine because for every j, all $P(q_i = S_{i_j})$ have not necessarily the same magnitude. The scaling is applied after every α and β computation, because underflow may occur inside a time step. For the same reason, $P(O|\lambda)$ may exceed the precision range of any machine; thus we do not consider this probability, but $log[P(O|\lambda)]$. This is called the *log-likelihood* of the observation.

Underflow may also occur while computing $\bar{a}(\sigma_i, i_j)$, depending on the precision used for the calculus, and having some of these transitions probabilities artificially stuck to zero prevents the algorithm to converge.

Choosing a good representation for the belief state is critical for the algorithm to behave correctly. A flat representation over the variables would lose all the correlations among them, being a very bad approximation. On the other side, maintaining an exact belief state implies keeping all correlations over the time, and thus would be exponentially complex over the length of the evidence [Boyen & Koller 1998]. Approximating the belief state with a particle filter seems to be a good trade-off. But while the size of the hypothesis state is very large, we chose to have a variable number of particles, because the probability of seeing the observations knowing the belief state might become very low. We maintain this probability by creating more particles (from the most probable ones) until this probability reaches its initial value.

Structure Learning

The EM algorithm updates only the transitions probabilities, without changing the structure of the DBN. An algorithm has been proposed [Friedman 1998] to add inside the loop some structural changes. The main issue is to evaluate the new model efficiently knowing the current one. Furthermore, this algorithm acts only as a local search into the

structure space, so the optimal structure may not be reached. The GES algorithm [Chickering 2002] has been proposed to find an optimal structure of a Bayesian network. However, this technique relies upon sufficient statistical knowledge of the process, which is not our case due to the very large size of the observation space. In our case, the global structure is known. In order to decrease the size of the DBN and speed up the learning process, we plan to use these techniques only on the hidden variables of the process (and on the links toward the observable variables),. For now, we use the structure described in section **Results**.

Incremental Learning

An open issue for learning stochastic models is the possibility to add knowledge into a previously learned model. This is difficult because the learning phase is a local search, and the algorithm must take into account all observations with the same weight.

We propose the following algorithm: we first learn λ_1 upon a training sequence $(O_1..O_i)$. Then we learn λ_2 over $(O_j..O_n)$, *taking* λ_1 *as a start point for the local search of* λ_2 [4]. When its done, we simply sum λ_1 and λ_2. For this merging to work, it must be applied after *each iteration* of EM, i.e. the merged DBN must be used for Maximization and not only for Expectation. We think this is because the role of the hidden variables may be changed otherwise, loosing the "meaning" of them. If the merging is done only between the different phases, the newly learned DBN optimizes itself only in the space of the current trunk of observations and goes somewhat "too far" from the previous one. In this case, the sum of the two DBNs does not reflect the reality of the space of the sum of the observations. This will be investigated in future work.

Adaptation of the Behavior

One possible use of the model presented in section is to find optimal values for the ND motion generator controller, depending on the environment. ND is usually used with a fixed set of parameters, that are not optimal in every situation. We aim at having a fine-grained adaption of the parameters of the controller, modeled as a DBN, to optimize the navigation itself.

We need to introduce utilities into some of our variables with respect to an optimality criterion. We need to give high rewards to variable values that will lead to a desired behavior and penalties to values that will lead to a bad one. Typically, we need to avoid the *fail* value of the **general state** variable, while we want to reach the *end* value.

We also introduce a secondary criterion on the **human adequation** of the navigation behavior. Between two successful behaviors, we prefer a behavior where the *normal* value of the **human adequation** appears often, and try to avoid the *aggressive* and *shy* values of this variable. All these utilities will be given as scalar values, so we need to give greater values to the **general state** variable, and smaller ones to the secondary criterion, to avoid the case where the robot could collect reward by having a proper behavior with respect to humans, yet fail, thinking it is better to be gentle than to achieve its task.

A general scheme of a DBN-based behavior adapter can be seen on figure 3.

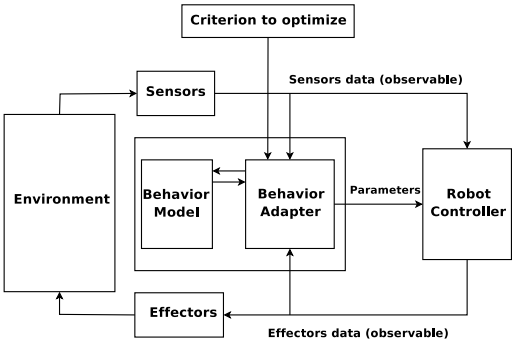


Figure 3: General scheme of a behavior adapter

Decision

In this application, the temporal aspect is primordial. The command process works at a frequency of 2.5 Hertz. The behavior adapter has a frequency of 1.66 Hz to collect the observations, but we do not need it to change the parameters at such a high frequency. Typically the parameters changes should occur at most at 0.5 Hertz, or less. Otherwise, the system could demonstrate a very unstable behavior.

So the problem we face is quite different from a classical Dynamic Decision Network [Zhang, Qi, & Poole 1994] resolution, where a decision is taken after each observation, and the result is a decision tree which associates to each observation sequence an optimal decision sequence. Furthermore, the branching factor for a decision tree would be too high. We need a radically different strategy: we consider that when we take a decision, it will remain the same for a given amount of time. So we independently evaluate every decision towards a given horizon, and choose the best one.

The DBN we build includes a model of the evolution of the environment, thus we can accurately predict what will happen in the future. Starting from a belief state for the current time step (i.e. a probabilistic hypothesis on the current state of the system, including hidden variables), we can infer from the DBN the future belief states for each set of parameters and deduce the corresponding utilities.

The belief state is represented as a particle filter, each particle represents a weighted hypothesis on the values of the variables. When inferring the particles over time, we can compute the expected utility by simply summing utilities encountered by particles. This can be done for a few steps forward (which leads to imprecise expectations), or until all particles reach a final state (defined by the *end* and *fail* values of **general state**). The size of the estimation of the belief state (i.e. the number of particles used) is therefore a critical bottleneck for real-time decision making; to solve this

[4] Note that λ_1 was learned from a random starting point. Thus the next phases of the learning should be better than the first, because the start point is not random, but somewhat represents a part of the dynamic of the system.

issue, we sample the belief state to focus only on the few most probable hypothesis.

Meta Decision

We introduce a confidence factor in every transition probability. This confidence factor is the number of updates of a transition probability during the learning phase. This helps us to differentiate an equiprobability due to a never seen transition into the data set from an informative one. When inferring particles to predict expected utilities of the decisions, we also collect the confidence factors of the transitions the particles used. If a particle comes with a high confidence factor, this means that this particle re-played in a learned set, whereas if the confidence factor is low, the particle went through never learned transitions. Thus the decision has to be taken in a two dimensional space: *utility* $\times$ *confidence*. If we prefer to take a decision with high confidence, this will lead to re-play a previously learned observation, and the robot will preferably exploit its model. On the contrary, if we choose a decision with a low confidence factor, this will lead go through never learned transitions of our model, making the robot more "exploratory" of its capabilities, with a higher risk of failure. We encounter here the classic learning agent "exploration versus exploitation" trade-off, in an original fashion.

Results

Navigation Behavior Learning

The model management is implemented into a real-world robotic architecture [Alami *et al.* 2000], on a B21R robot. The monitoring of raw data is done at a frequency of 2.5 Hertz. The scalar raw data are clustered using k-means clustering [Kanungo *et al.* 2002] in order to obtain a few clusters.

The a priori adjacency structure of the graph is the following:

- in the same time step, every environment variable is linked to every hidden variable, every hidden variable is connected to every other hidden one, and finally every hidden variable is connect to every post-control observable variable;
- from step t to step $t + 1$, every environment variable is linked to itself, every control variable is linked to every hidden variable, and every hidden variable is connected to itself.

We did about 120 navigations of the robot into different environments, choosing randomly the control parameters at random frequencies; for a totalizing a few hundred meters of motion into cluttered environments, with people around the robot. The operator was asked to give the value of the **human adequation** (into *good*, *aggressive*, and *shy*). He also has a *fail* button to stop navigations before dangerous failure like collisions. A failure case was automatically detected when the robot did not move during a given amount of time. The number of observations collected is about 7,000. Yet, it does not cover the total observation space at all. The learning of the DBN takes several minutes per iteration of Expectation-Maximization. The learned DBN stabilizes after less than 10 iterations. As EM is a local search, we use a "conservative" DBN as a starting point, meaning that with this DBN the probability that the variables change their value is low (about 10%).

To evaluate the quality of the learned DBN, we learn the DBN on all observations but one, and try to predict this observation. This is done by inferring a particle filter on the DBN, choosing the most probable observation in this particle filter, and comparing this predicted observation to the actual one. The global observation is seldom exactly the same, but is generally very close, that means that in most cases the particle filter predicts well most of the observable variables. An overview of recognition results is shown on table 4. We can notice that the recognition results are significantly better than a random prediction. Furthermore, if some aspects of the internal mechanisms of ND have not been well modeled (as the strategy choice, or number of valleys build) because of the lack of the environment variables that uses ND "for real", the general behavior and the human adequation are well predicted. And these are the variables we use in particular for control.

variable	random guess	dbn
cluttering	33.3	87.3
angle of obstacle	25.0	86.4
dist. to obstacle	25.0	88.5
# segments	33.3	81.6
# valleys	33.3	79.9
v	33.3	90.8
w	25.0	79.0
dv	33.3	91.9
dw	33.3	92.2
Δ dist. to goal	33.3	77.8
% mission	33.3	97.1
srtategy	20.0	84.8
general	25.0	96.4
adequation	33.3	92.5
average	29.0	87.6

Figure 4: Table of well-guessed one-step predictions (%)

Using more hidden variables gives better results, meaning that 2 hidden variables with an arity of 2 is not enough. But using more hidden variables makes the learning process much longer because of the "complete" structure we use. Here we consider 49 causal links, while with 3 variables it comes up to 73. We intend to decrease this amount of links using structure learning in future work.

Control

The decision level of the behavior adapter was implemented using OpenPRS [Ingrand *et al.* 1996]. The current belief state is maintained along the navigation, and when a choice is to be made, it is sampled in order to limit the size of the particle filter for prediction.

In our control experiments, the choices were made at a fixed period of 2 seconds, changing reactive obstacle avoid-

ance parameters dynamically during navigation, without stopping the robot motion. Therefore, the navigation behaves better than with default static hand-tuned parameters usually used on this robot, and much better than with random parameters. Typically, during the learning phase with random parameters, 1 run out of 3 ends on a failure state due to collisions or to blocked situations, while with the behavior adapter, the failures happen only once out of 10 experiments. This happens while using a parameter set never tried during the learning phase[5]. Also, in the learning phase, we label as *good* the parameters producing very high accelerations when the robots enters an open area. We give rewards to this value for decision making, and this behavior is clearly noticeable in these experiments, while it is not with the default hand-tuned parameters. This gives us quicker navigation for open areas than with default parameters. The average speedup of the navigation will be quantified in a future work.

Collecting observations can be made in the same time than active behavior adapting; the next step is to decide when the robot should use computed behavior or behaviors that extend the variety of the set of collected observations.

Incremental Learning

To evaluate the incremental learning algorithm, we learned a DBN in five phases as explained in section *Incremental Learning*. Each phase uses 20% of the total observation. On figure 5, the phases are numbered on the X-axis. The Y-axis represents the log-likelihood obtained. The upper line shows the log-likelihood obtained with a single phase of learning (upon all observations). While the summed DBN does not behaves as well as the one learned in a single phase, the difference is very small.

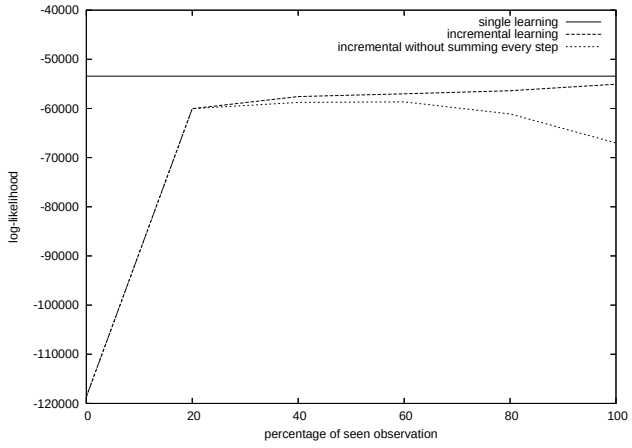


Figure 5: Incremental learning likelihoods

Conclusion and Discussion

We have presented an innovative approach to learn DBN models of complex robot behaviors. In our context, where we want to learn a model where controllable variables remain observable, DBNs are preferable to HMMs. We developed a modified version of EM for DBN. We presented our behavior controller, and how it can be used to learn the model and then use it online. The approach has been implemented and successfully tested on a robotic platform. We showed that, for a fairly complex navigation task, we were able to learn a model which, when used to dynamically set the controllable parameters (according to given objectives), performs much better than the default set, or a random set.

This work is closely related to [Liao, Fox, & Kautz 2004], where the authors tries to model human activities with hierarchical DBN. Our approach tries to deal with more variables, and with sparse data sets. Furthermore, we propose a direct way to use our model not only for diagnosis, but to control the process itself. In [Oates, Schmill, & Cohen 2000], the authors learn to recognize robotic behaviors that fits with human judgment, but the emphasis is more on "how to built variables that make a intuitive sense" than on use of this kind of model. The proposed use is a more classical planning approach [Schmill, Oates, & Cohen 2000]. We propose an efficient way for mixing learning and decisions making. Our approach is in the same spirit than [Belker, Beetz, & Cremers 2002], where the authors model the system as neural nets or with tree induction, then take decision based on a MDP, for robotic autonomous navigation. We propose a more integrated approach, where the decision-taking mechanism is much closer from the model itself. Furthermore, it allows taking into account hidden variables, i.e. a more complex structure of the model itself. We propose a way to build a model using more features than the one proposed. In [Stulp & Beetz 2005] the authors present an original way of optimizing tasks execution. High-level tasks are built with a number of parameters that must be optimized for smooth execution. Performance models are learned for the parameter sets as tree rules, but much of the work is hand-made. Our approach is automated, and the models we build are more expressive (while harshly human-readable), due to the presence of hidden variables. Furthermore, the models we built are fully probabilistic, while this is not the case in this work. Both approaches show very high computational cost when different complex tasks have to be mixed.

In the long run, we keep in mind that these learned behaviors should not only be used to control the robot execution (to avoid failure and to optimize the execution), but can also be taken into account by a high level planner able to choose among a set of such behaviors.

Despite our application to a navigation task of a museum guide robot, we believe that the approach remains applicable to other behaviors and could be used for other autonomous systems.

[5]This is due to a confidence factor allowing learning more than using the model.

References

Alami, R.; Chatila, R.; Fleury, S.; Herrb, M.; Ingrand, F.; Khatib, M.; Morisset, B.; Moutarlier, P.; and Simon, T. 2000. Around the lab in 40 days... In *Proceedings of ICRA*.

Amir, E. 2005. Learning partially observable deterministic action models. In *Proceedings of IJCAI*.

Belker, T.; Beetz, M.; and Cremers, A. 2002. Learning action models for the improved execution of navigation plans. *Robotics and Autonomous Systems* 38(3-4):137–148.

Bobick, A., and Davis, J. 2001. The Recognition of Human Movement using Temporal Templates. *IEEE Transaction on Pattern Analysis and Machine Intelligence* 23(3):257–267.

Boyen, X., and Koller, D. 1998. Tractable inference for complex stochastic processes. In *Proceedings of UAI*.

Bui, H.; Venkatesh, S.; and West, G. 2002. Policy Recognition in the Abstract hidden Markov model. *Journal of AI Research* 17:451–499.

Bui, H. 2003. A General Model for Online Probabilistic Plan Recognition. In *Proceedings of IJCAI*.

Chickering, D. 2002. Optimal structure identification with greedy search. *Journal of Machine Learning Research* 3:507–554.

Clodic, A.; Fleury, S.; Alami, R.; Herrb, M.; and Chatila, R. 2005. Supervision and interaction. In *Proceedings ICAR*, 725–732.

Cohen, I.; Sebe, N.; Chen, L.; Garg, A.; and Huang, T. S. 2003. Facial Expression Recognition from Video Seqences: Temporal and Static Modelling. *Computer Vision and Image Understanding: Special Issue on Face Recognition* 91:160–187.

Dean, T., and Kanazawa, K. 1990. A model for reasoning about persistence and causation. *Computational Intelligence* 5(3):142–150.

Dearden, A., and Demiris, Y. 2005. Learning forward models for robots. In *Proceedings of IJCAI*.

Fox, M.; Ghallab, M.; Infantes, G.; and Long, D. 2006. Robot introspection through learned hidden markov models. *Artificial Intelligence* 170(2):59–113.

Friedman, N. 1998. The bayesian structural EM algorithm. In *Proceedings of UAI*.

Ingrand, F.; Chatila, R.; Alami, R.; and Robert, F. 1996. PRS: A High Level Supervision and Control Language for Autonomous Mobile Robots. In *Proceedings of ICRA*.

Kanungo, T.; Mount, D.; Netanyahu, N.; Piatko, C.; Silverman, R.; and Wu, A. 2002. An efficient k-means clustering algorithm: analysis and implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24(7):881–892.

Koenig, S., and Simmons, R. G. 1996a. Passive Distance Learning for Robot Navigation. In *Proceedings of International Conference on Machine Learning (ICML)*.

Koenig, S., and Simmons, R. G. 1996b. Unsupervised Learning of Probabilistic Models for Robot Navigation. In *Proceedings of ICRA*.

Koller, D., and Fratkina, R. 1998. Using learning for approximation in stochastic processes. In *Proceedings of ICML*.

Liao, L.; Fox, D.; and Kautz, H. 2004. Learning and Inferring Transportation Routines. In *Proceedings of AAAI*.

Minguez, J.; Osuna, J.; and Montano, L. 2004. A "Divide and Conquer" Strategy based on Situations to achieve Reactive Collision Avoidance in Troublesome Scenarios. In *Proceedings of ICRA*.

Nam, Y., and Wohn, K. 1996. Recognition of Space-Time Hand Gestures using Hidden Markov Models. In *ACM Symposium on Virtual Reality Software and Technology*, 51–58.

Oates, T.; Schmill, M. D.; and Cohen, P. R. 2000. A method for clustering the experiences of a mobile robot that accords with human judgements. In *Proceedings of IJCAI*.

Osentoski, S.; Manfredi, V.; and Mahadevan, S. 2004. Learning Hierarchical Models of Activity. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*.

Pearl, J. 1988. *Probabilistic Reasoning in Intelligent Systems : Networks of Plausible Inference*. Morgan Kaufmann.

Rabiner, L. R. 1989. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proceedings of the IEEE* 77(2):257–286.

Schmill, M. D.; Oates, T.; and Cohen, P. R. 2000. Learning planning operators in real-world, partially observable environments. In *Proceedings of ICAPS*.

Stulp, F., and Beetz, M. 2005. Optimized execution of action chains using learned performance models of abstract actions. In *Proceedings of IJCAI*.

Wilson, A., and Bobick, A. 1999. Parametric Hidden Markov Models for Gesture Recognition. *IEEE Transaction on Pattern Analysis and Machine Intelligence* 21(9):884–900.

Wilson, A., and Bobick, A. 2001. Hidden Markov Models for Modeling and Recognizing Gesture Under Variation. *International Journal of Pattern Recognition and Artificial Intelligence* 15(1):123–160.

Zhang, N.; Qi, R.; and Poole, D. 1994. A computational theory of decision networks. *International Journal of Approximate Reasoning* 11(2):83–158.