Conditional progressive planning under uncertainty

Lars Karlsson
Center for Applied Autonomous Sensor Systems
Örebro University, SE-701 82 Örebro, Sweden
http://www.aass.oru.se/
lars.karlsson@tech.oru.se

Abstract

In this article, we describe a possibilistic/probabilistic conditional planner called PTLplan. Being inspired by Bacchus and Kabanza's TLplan, PTLplan is a progressive planner that uses strategic knowledge encoded in a temporal logic to reduce its search space. Actions effects and sensing can be context dependent and uncertain, and the information the planning agent has at each point in time is represented as a set of situations with associated possibilities or probabilities. Besides presenting the planner itself — its representation of actions and plans, and its algorithm — we also provide some promising data from performance tests.

1 Introduction

In this article, we describe a conditional planner called PTLplan. A conditional planner does not presuppose that there is complete information about the state of its environment at planning time, but that more information may be available later due to sensing, and that this new information can be used to make choices about how to proceed. Hence, a conditional planner generates plans that may contain conditional branches. In addition, PTLplan can handle degrees of uncertainty, either in possibilistic or in probabilistic terms. The "P" in PTLplan stands for exactly that.

PTLplan is a progressive (forward-chaining) planner; it starts from an initial situation and applies actions to that and subsequent resulting situations, until a situation where the goal is satisfied is reached. Being progressive, PTLplan has the disadvantage that the search space is potentially very large even for small problems The advantage is that PTLplan can reason from causes to effects, and always in the context of a completely specified situation. The latter makes it possible to apply a technique that can reduce the search space considerably: the use of strategic knowledge that helps pruning away unpromising plan prefixes.

PTLplan builds on a conditional planner ETLplan [Karlsson, 2001], which in turn built on a sequential (non-conditional) planner called TLplan [Bacchus and Kabanza, 1996; 2000] ("TL" stands for "temporal logic"). Two of the features that makes TLplan interesting are its fairly expressive first-order representation and its good performance due to its use of strategic knowledge. The latter is indicated by its outperforming most other comparable planners in empirical tests, as has been documented in [Bacchus and Kabanza, 2000]. ETLplan added two features to TLplan:

- The representation of actions and situations used in ETLplan permitted actions that have sensing effects, that is the agent may observe certain fluents (state variables).
- Based on these observations, the planning algorithm could generate conditional plans.

In addition to these features, PTLplan also incorporates:

- Assignments of degrees of uncertainty, either in possibilistic or probabilistic terms, to effects, observations and situations/states.
- Based on these degrees, measures of how likely a plan is to succeed or fail. In the absence of plans that are completely certain to succeed, PTLplan is still capable of finding plans that, although they may fail, are likely enough to succeed.

PTLplan is the first progressive planner utilizing strategic knowledge to incorporate the features mentioned above, and as indicated by tests, it does so successfully. In the rest of the article, we describe the plan representation of PTLplan, the use of strategic knowledge and the planning algorithm, and we also briefly provide some data from performance tests.

2 Related work

Planning in partially observable domains has been a hot topic since the mid 90's. There has been a good amount on work on POMDPs (partially observable Markov decision processes), see e.g. [Kaebling *et al.*, 1998]. POMDPs typically utilize explicit enumerations of states, but there are also results on the use of more compact propositional representations [Boutilier and Poole, 1996]. Note that this work is on the more general problem of generating policies that maximize utilities, and not plans with sequences and branches that achieve goals.

One of the earliest conditional probabilistic systems originating from classical planning was the partial-order planner C-BURIDAN [Draper *et al.*, 1994], which had a performance

which made in impractical for all but the simplest problems. The partial-order planner MAHINUR [Onder and Pollack, 1999] improved considerably on C-BURIDAN by handling contingencies selectively. MAHINUR focuses its efforts on those contingencies that are estimated to have the greatest impact on goal achievement. C-BURIDAN and MAHINUR are regressive planners: they perform means-ends reasoning from effects to causes.

WEAVER [Blythe and Veloso, 1997] is a system that iterates between a classical planner and a probabilistic evaluator. It can model stochastic exogenous events but not sensing.

C-MAXPLAN and ZANDER [Majercik and Littman, 1999] are two conditional planners based on transforming propositional probabilistic planning problems to stochastic satisfiability problems (E-MAJSAT and S-SAT, respectively). The latter can be solved in a highly efficient manner, in particular when certain pruning techniques are used. But, as pointed out in [Majercik and Littman, 1999], the translation from planning problems to satisfiability problems can lead to blow-ups in size and obscures the problem structure.

On the possibilistic side, there is Guéré's and Alami's conditional planner [Guéré and Alami, 1999], which also has been demonstrated to have a practical level of performance. Their planner is based on Graph Plan [Blum and Furst, 1995]. It assumes that sensing actions are perfectly reliable and do not have preconditions. Another Graph Plan derivative is Sensing Graph Plan (SGP) [Weld *et al.*, 1998], which can deal with incomplete information but has no means for quantifying uncertainty.

3 Representation

3.1 Uncertainty

PTLplan can represent uncertainty either using possibility theory [Dubois and Prade, 1988] or probability theory, simply by interpreting the connectives used as follows.[1]

Connective	Possibility	Probability	Comment
$\otimes$	min	$\cdot$	"and"
$\oplus$	max	$+$	"or"

Thus, PTLplan can be used both when one is only interested in the relative likelihood of different facts and outcomes, and when one has precise information about probabilities.

3.2 Syntax: Fluents

The state of the world is described in terms of fluents (state variables) and their values. A fluent-value formula has the form f=v, denoting that the fluent f has the value v; if the value is omitted, it is implicitly assumed to be `T` (true). Examples of fluent-value formulae are `door(d1)` and `robot-at(table1)=F`. A fluent formula is a logical combination of fluent-value-formulae using the standard connectives and quantifiers. A fluent-assignment formula has the form f`:=`v denoting that f is caused to have the value v; e.g. `robot-at(table1):=F`.

[1] The probabilistic versions of the connectives are justified by the fact that they are only applied to mutually exclusive and independent situations or branches and Markovian transitions.

3.3 Syntax: Actions

An action schema consists of a tuple $\langle a, P, R\rangle$ where a is the action name, P is a precondition (a fluent formula) and R is a set of result descriptions. Each result description $r \in R$ is a tuple $\langle C, p, E, O\rangle$ where

- C is a context condition (a fluent formula) that determines when (in what states) the result is applicable.
- p is the possibility or probability of the result.
- E is a set of fluent assignment formulae (f`:=T` or f`:=F`) which specifies the effects of the result. We let E^+ denote the fluents with positive assignments in E and we let E^- be those with negative ones.
- O is a set of fluents f, denoting that the current value of f is observed, and/or fluent-value-formulae f=v denoting that f is (correctly or incorrectly) observed to have the value v. These observations are assumed to be made by the agent executing the plan.

Note that restrictions apply to the p values; the sum of the p in applicable results must always be 1 for the probabilistic case, and the maximum of the p must always be 1 for the possibilistic case.

Example 1 The following are action schemas for the tiger scenario (the scenario is due to [Kaebling *et al.*, 1998]). There are two doors, one to the left (`ld`) and one to the right (`rd`). Behind one of them lurks a tiger, and behind the other there is a reward. The fluent `tiger(`*d*`)` stands for that the tiger is behind door *d*. Probabilities are used.

The agent can listen at the doors, and this gives a indication of behind which door the tiger lurks. Unfortunately, there is a 15% chance of error. This is an action that yields observations but no concrete effects.

act: `listen()`
pre: `true`

	context	*p*	*effects*	*observations*
res:	`(tiger(ld),`	`0.85,`	`{},`	`{tiger(ld)=T}),`
	`(tiger(ld),`	`0.15,`	`{},`	`{tiger(rd)=T}),`
	`(tiger(rd),`	`0.85,`	`{},`	`{tiger(rd)=T}),`
	`(tiger(rd),`	`0.15,`	`{},`	`{tiger(ld)=T})`

The agent can open one of the doors, which either leads to the reward (`rew`), or to tiger-induced death (`dead`).

act: `open(?d)`
pre: `door(?d)`

	context	*p*	*effects*	*observations*
res:	`(¬tiger(?d),`	`1.0,`	`{rew:=T},`	`{rew=T}),`
	`(tiger(?d),`	`1.0,`	`{dead:=T},`	`{dead=T})`

3.4 Semantics: Situations and epistemic situations

We use a model of knowledge and action which is based on the concepts of a situation and an epistemic situation. In short, a situation describes one possible state of the world at a given point in time, where time is defined in terms of what actions has occurred. An epistemic situation, or e-situation for short, is essentially a set of situations with associated possibilities/probabilities that describes the agent's knowledge at a point in time. In this PTLplan is similar

to e.g. C-BURIDAN [Draper *et al.*, 1994] which employs a probability distribution over states. A transition relation res over situations provides the temporal dimension; the transitions are due to applications of actions. The global possibility/probability of a situation s is denoted $p(s)$. This represents the possibility/probability that some given choice of actions (plan) will lead to s. The global possibility/probability $p(\overline{s})$ of an e-situation is computed from those of its constituents: $p(\overline{s}) = \bigoplus_{s \in \overline{s}} p(s)$. The local possibility/probability $p_l(s)$ of a situation s is obtained by normalizing relative to its containing e-situation $\overline{s}$: $p_l(s) = p(s)/p(\overline{s})$. This represents the possibility/probability the agent assigns to s inside $\overline{s}$.

The state $state(s)$ of a situation s is a mapping from fluents to values (i.e. a set of fluents), and the observation set $obs(s)$ is a set of fluent-value-pairs $\langle f, v \rangle$. We impose the constraint that all situations in an e-situation $\overline{s}$ should have the same observation set; we denote this set by $obs(\overline{s})$.

Example 2 The initial e-situation for the tiger scenario has no observations (*obs*= { }), but contains two situations with associated probabilities and states. In one situation where $p = 0.5$, the tiger is behind the left door, and in the other, where $p = 0.5$, it is behind the right door.

obs=	{ }
0.5:	{tiger(ld), door(ld), door(rd)}
0.5:	{tiger(rd), door(ld), door(rd)}

3.5 Semantics: results of actions

The results of an operator/action A in a situation is defined as follows: for each result description $\langle C_i, p_i, E_i, O_i \rangle$, if context condition C_i is true in s then there is a situation s' resulting from s (i.e. $res(s, s')$) where the effects in E_i occur (i.e. $state(s') = (state(s) \cup E_i^+) \setminus E_i^-$) and the observations in O_i are made (if $f \in O_i$ and f has value v in s, or if "f=v" $\in O_i$, then $\langle f, v \rangle \in obs(s')$). Finally, $p(s') = p(s) \otimes p_i$. For an epistemic situation $\overline{s}$, the result of an action A is determined by applying A to the different situations $s \in \overline{s}$ as above, and then partitioning the resulting situations into new e-situations according to their observation sets.

Example 3 We attempt to apply some different actions to the initial e-situation. Applying `open(ld)` yields the following two new e-situations ("..." stands for "`door(ld),door(rd)`"):

obs=	{dead=T}
0.5:	{tiger(ld), dead,...}

obs=	{rew=T}
0.5:	{tiger(rd), rew,...}

Applying `listen()` to the initial e-situation yields the following two new e-situations:

obs=	{tiger(ld)=T}
0.425:	{tiger(ld),...}
0.075:	{tiger(rd),...}

obs=	{tiger(rd)=T}
0.425:	{tiger(rd),...}
0.075:	{tiger(ld),...}

Note how the resulting situations are partitioned into e-situations based on their observations.

3.6 Syntax: Conditional plans

Plans in PTLplan are conditional. This means that there can be points in the plans where the agent can choose between different ways to continue the plan depending on some explicit condition. Therefore, in addition to the sequencing plan operator (`;`), we introduce a conditional operator (`cond`). The syntax of a conditional plan is as follows:

plan ::= `success` | `fail` | *action* `;` *plan* |
`cond` *branch*`*`
branch ::= `(`*cond* `:` *plan*`)`
action ::= *action-name*`(`*args*`)`

A condition *cond* is a conjunction of fluent-value formulae. The conditions for a branch should be exclusive and exhaustive relative to the potential e-situations at that point in the plan. `Success` denotes predicted plan success, and `fail` denotes failure.

Example 4 The following is a plan which is a solution to the tiger scenario (see example 6 for how it could be generated).

```
listen();
cond(tiger(rd)=T : open(ld);
     cond(rew=T:success)(dead=T:fail))
     (tiger(ld)=T : open(rd);
     cond(rew=T:success)(dead=T:fail))
```

3.7 Semantics: Application of conditional plans

The application of a plan to an epistemic situation $\overline{s}_i$ results in a set of new e-situations. First, the application of an action a to $\overline{s}_i$ is defined as in section 3.5. Next, the application of a sequence $a;p$ is defined as applying the first action a to $\overline{s}_i$, and then the rest of the plan p to each of the resulting e-situations. Finally, the application of a conditional plan element `cond` $(c_1 : p_1) \ldots (c_n : p_n)$ is defined as an application of the branch p_j whose context condition c_j matches with $obs(\overline{s}_i)$ (there should only be one such branch).

This concludes how actions and plans are represented in PTLplan. Next, we proceed to investigate how strategic knowledge can be represented and used.

4 Strategic knowledge

In order to eliminate unpromising plan prefixes and reduce the search space, PTLplan (and TLplan and ETLplan before it) utilizes strategic knowledge. This strategic knowledge is encoded as expressions (search control formulae) in an extension of first-order linear temporal logic (LTL) [Emerson, 1990] and is used to determine when a plan prefix should not be explored further. One example could be the condition "never pick up an object and then immediately drop it again". If this condition is violated, that is evaluates to `false` in some e-situation, the plan prefix leading there is not explored further and all its potential continuations are cut away from the search tree. A great advantage of this approach is that one can write search control formulae without any detailed knowledge about how the planner itself works; it is sufficient to have a good understanding about the problem domain.

LTL is based on a standard first-order language consisting of predicate symbols, constants and function symbols and the

Algorithm *Progress(f,$\overline{s}$)*
Case:

1. $f = \mathcal{K}f_1 :\ f^+ := \begin{cases} \texttt{true} \text{ if } K \leq (1 - \bigoplus_{s \in S^-} p_l(s)) \\ \text{where } S^- = \{s \in \overline{s} | s \models \neg f_1\}, \\ \texttt{false} \text{ otherwise} \end{cases}$
2. $f = \mathcal{O}(f_1{=}v) :\ f^+ := \begin{cases} \texttt{true} \text{ if } \langle f_1, v\rangle \in obs(\overline{s}), \\ \texttt{false} \text{ otherwise} \end{cases}$
3. $f = \mathcal{G}f_1 :\ f^+ := \begin{cases} \texttt{true} \text{ if } s \models f_1 \text{ for all } s \in G, \\ \texttt{false} \text{ otherwise} \end{cases}$
4. $f = f_1 \wedge f_2 :\ f^+ := \ Progress(f_1, \overline{s}) \wedge Progress(f_2, \overline{s})$
5. $f = \neg f_1 :\ f^+ := \neg Progress(f_1, \overline{s})$
6. $f = \bigcirc f_1 :\ f^+ := f_1$
7. $f = f_1 \mathcal{U} f_2 :\ f^+ := \ Progress(f_2, \overline{s}) \vee (Progress(f_1, \overline{s}) \wedge f)$
8. $f = \Diamond f_1 :\ f^+ := Progress(f_1, \overline{s}) \vee f$
9. $f = \Box f_1 :\ f^+ := Progress(f_1, \overline{s}) \wedge f$
10. $f = \forall x[f_1] :\ f^+ := \bigwedge_{c \in U} Progress(f_1(x/c), \overline{s})$
11. $f = \exists x[f_1] :\ f^+ := \bigvee_{c \in U} Progress(f_1(x/c), \overline{s})$

Return f^+

Figure 1: The PTLplan progression algorithm.

usual connectives and quantifiers. In addition, there are four temporal modalities: $\mathcal{U}$ (until), $\Box$ (always), $\Diamond$ (eventually), and $\bigcirc$ (next). In LTL, these modalities are interpreted over a sequence of situations, starting from the current situation. For the purpose of PTLplan, we can interpret them over a sequence/branch of epistemic situations $B = \langle \overline{s}_1, \overline{s}_2, \ldots \rangle$ and a current epistemic situation $\overline{s}_i$ in that sequence. The expression $\phi_1 \mathcal{U} \phi_2$ means that ϕ_2 holds in the current or some future e-situation, and in all e-situations inbetween ϕ_1 holds; $\Box\phi$ means that ϕ holds in this and all subsequent e-situations; $\Diamond\phi$ means that ϕ holds in this or some subsequent e-situation; and $\bigcirc\phi$ means that ϕ holds in the next e-situation $\overline{s}_{i+1}$.

In addition to the temporal modal operators from LTL, PTLplan also uses a goal operator $\mathcal{G}$, which is useful for referring to the goal in search control formulae. We let $\mathcal{G}\varphi$ denote that it is among the agent's goals to achieve the fluent formula φ. Semantically, this modality will be interpreted relative to a set of goal states G, i.e. the set of states that satisfy the goal. We also introduce two new modal operators: $\mathcal{K}\varphi$ ("knows") means that the necessity/probability that the fluent formula φ is true in the current e-situation exceeds some prespecified threshold *K*, given the information the agent has; and $\mathcal{O}f{=}v$ denotes that f is observed to have the value v in the current e-situation. We restrict fluent formulae to appear only inside the $\mathcal{K}$, $\mathcal{G}$ and (for fluent-value formulae) $\mathcal{O}$ operators.

We can now define the semantics of these modal operators relative to a branch B of epistemic situations, a current epistemic situation $\overline{s}_i$, a variable assignment V, and a set of goal states G, as follows.

- $(B, \overline{s}_i, V, G) \models \phi_1 \mathcal{U} \phi_2$ iff there exists a $j \geq i$ such that $(B, \overline{s}_j, V, G) \models \phi_2$ and for all k such that $i \leq k < j$, $(B, \overline{s}_k, V, G) \models \phi_1$.
- $(B, \overline{s}_i, V, G) \models \Box\phi$ iff for all $j \geq i$, $(B, \overline{s}_j, V, G) \models \phi$.
- $(B, \overline{s}_i, V, G) \models \Diamond\phi$ iff there exists a $j \geq i$ such that $(B, \overline{s}_j, V, G) \models \phi$.
- $(B, \overline{s}_i, V, G) \models \bigcirc\phi$ iff $(B, \overline{s}_{i+1}, V, G) \models \phi$.
- $(B, \overline{s}_i, V, G) \models \mathcal{G}\varphi$ iff for all $s \in G$, $(s, V) \models \varphi$.
- $(B, \overline{s}_i, V, G) \models \mathcal{K}\varphi$ iff $K \leq 1 - (\bigoplus_{s \in S^-} p_l(s))$ where $S^- = \{s \in \overline{s}_i | (s, V) \models \neg\varphi\}$.
- $(B, \overline{s}_i, V, G) \models \mathcal{O}f{=}v$ iff $\langle f, v\rangle \in obs(\overline{s}_i)$.

The interpretation of $\mathcal{K}\varphi$ is motivated by the fact that in possibility theory, necessity is defined as $Nec(\varphi) = 1 - Pos(\neg\varphi)$, and in probability theory $P(\varphi) = 1 - P(\neg\varphi)$.

In order to efficiently evaluate control formulas, PTLplan incorporates a progression algorithm (similar to the ones of TLplan and ETLplan) that takes as input a formula f and an e-situation and returns a formula f^+ that is "one step ahead", i.e. corresponds to what remains to evaluate of f in subsequent e-situations. It is shown in figure 1. (The goal states G in case 3 are fixed for a given planning problem and need not be passed along.) Note that the algorithm assumes that all quantifiers range over a finite universe U.

Example 5 The following is a control formula stating that a robot should never pick an object up and then drop it immediately again, i.e. hold it only one moment.

$$\Box\neg(\mathcal{K}(\neg\exists x[\texttt{robot-holds}(x)]) \wedge \bigcirc(\mathcal{K}(\exists x[\texttt{robot-holds}(x)]) \wedge \bigcirc\mathcal{K}(\neg\exists x[\texttt{robot-holds}(x)]))) \quad (1)$$

5 The planning algorithm

PTLplan is a progressive planner, which means that it starts from an initial e-situation and then tries to sequentially apply actions until an e-situation where the goal is satisfied is reached. The algorithm is shown in figure 2. It takes as input an e-situation $\overline{s}$, a search control formula f, a goal formula g (with a $\mathcal{K}$), a set of actions A and a success threshold σ (the failure threshold is $1 - \sigma$). It returns a triple

Algorithm PTLplan($\overline{s}, f, g, A, \sigma$)

1. If $\overline{s} \models g$ then return $\langle \texttt{success}, p(\overline{s}), 0\rangle$.
2. Let $f^+ :=$ *Progress*$(f, \overline{s})$;
   if $f^+ = \texttt{false}$ then return $\langle \texttt{fail}, 0, p(\overline{s})\rangle$.
3. For the actions $a_i \in A$ whose preconditions are satisfied in all $s \in \overline{s}$ do:
   a. Let $S^+ = Apply(a_i, \overline{s})$.
   b. For each $\overline{s}_j^+ \in S^+$,
      let $\langle P'_j, succ'_j, fail'_j\rangle :=$ PTLplan($\overline{s}_j^+, f, g, A, \sigma$).
   c. If $(\bigoplus_j fail'_j) \geq 1 - \sigma$ then
      let $cont_i = \langle \texttt{fail}, 0, p(\overline{s})\rangle$.
   d. If $|S^+| = 1$, let $cont_i = \langle a_1 \texttt{ ; } P'_1, succ'_1, fail'_1\rangle$.
      Otherwise let $cont_i = \langle P_i, succ_i, fail_i\rangle$ where
      $P_i = a_i\texttt{;(cond(}c'_1\texttt{:}P'_1\texttt{)...(}c'_n\texttt{:}P'_n\texttt{))}$,
      $succ_i = (\bigoplus_j succ'_j)$, $fail_i = (\bigoplus_j fail'_j)$
      and each $c'_i = \bigwedge\{f{=}v \,|\, \langle f, v\rangle \in obs(\overline{s}_i^+)\}$.
4. Return the $cont_i = \langle P_i, succ_i, fail_i\rangle$ with the lowest $fail_i$, provided $fail_i < (1 - \sigma)$, or otherwise return $\langle \texttt{fail}, 0, p(\overline{s})\rangle$.

Figure 2: The PTLplan planning algorithm.

$\langle plan, succ, fail\rangle$ containing a conditional plan, a degree (possibility/probability) of success, and a degree of failure. It is initially called with the given initial e-situation and a search control formula.

Step 1 checks if the goal is satisfied in $\overline{s}$, if this is the case it returns the possibility/probability $p(\overline{s})$ of $\overline{s}$. Step 2 progresses the search control formula; if it evaluates to `false`, the plan prefix leading to this e-situation is considered unpromising and is not explored further. In step 3, we consider the different applicable actions. For each such action, we obtain a set of new e-situations (a). We then continue to plan from each new e-situation separately (b). For those subplans thus obtained, if the combined possibility/probability of failure is above the failure threshold, we simply return a fail plan (c). If there was a single new e-situation, we return a simple sequence starting with the chosen action (d). Otherwise, we add to the chosen action a conditional plan segment where branches are constructed as follows. The conditions are derived from the different observations of the different e-situations, and the sub-plans are those obtained in (b). The success and failure degrees of this new plan are computed by combining those of the individual sub-plans. In step 4, finally, we return the best of those plans (i.e. the one with the least failure degree) found in step 3, or a fail plan if the degree of failure is too high.

Example 6 To give a feel for the operation of PTLplan, we go through the tiger scenario. The actions are defined as in example 1, and the initial e-situation is as in example 2. There is one search control formula:[2]

$$f = \Box(\mathcal{K}(\neg \texttt{dead})) \tag{2}$$

Finally, the goal is:

$$g = \mathcal{K}(\texttt{rew} \wedge \neg \texttt{dead}) \tag{3}$$

This goal should be achieved with a probability of at least $\sigma = 0.8$, giving a failure threshold of 0.2.

1. From the initial e-situation, the applicable actions are `open(ld)`, `open(rd)` and `listen()`.
2. We first apply `open(ld)`, resulting in the e-situations described in the first half of example 3.

   (a) Continuing planning from the first of these e-situations, the condition $\Box(\mathcal{K}(\neg \texttt{dead}))$ after progression evaluates to false, so this branch yields $\langle \texttt{fail}, 0, 0.5\rangle$ (plan, success degree,failure degree); see step 2 in the algorithm.

   (b) In the second one, the goal is achieved, yielding $\langle \texttt{success}, 0.5, 0\rangle$.

   Thus, the combined probability of failure is $0 \oplus 0.5 = 0.5$, which is above the failure threshold (step 3.c).
3. Applying `open(rd)` gives a similar outcome.
4. Applying `listen()` to the initial e-situation yields the two e-situations from the second half of example 3.

   (a) Continuing from the first new e-situation, we eventually find that choosing `open(rd)` leads to success with a degree of 0.425: $\langle$`open(rd); cond(rew=T : success)(dead=T : fail)`, $0.425, 0.075\rangle$

   (b) Continuing from the second one, we find that choosing `open(ld)` leads to success with a probability of 0.425: $\langle$`open(ld); cond(rew=T : success)(dead=T : fail)`$, 0.425, 0.075\rangle$.

   Combining the branches from (a) and (b) and appending them to `listen()`, finally, yields the plan shown in example 4, with 0.85 and 0.15 as probabilities of success/failure.

[2]The same effect could be achieved by including $\neg$`dead` in the preconditions of the actions, but we encode it as a search control formula to have the opportunity to see one in action.

Problem	SC	No SC	Other planners	
Tiger 0.5	0.002	0.008	0.01	ZANDER
Tiger 0.85	0.008	0.05	0.02	d:o
Tiger 0.939	0.07	0.20	0.08	d:o
Coffee	0.05	11.82	2.73	MAHINUR
Ask-coffee	0.16	13.06	769.20	ZANDER
Cold-room	0.56	556.80	11.78	Guéré-Alami

Figure 3: Some solution times for different scenarios, in CPU seconds. SC = "search control formulae were used".

6 Implementation and experiments

PTLplan has been implemented in Allegro CommonLisp, and uses a breath first search method involving detection of previously visited situations. The performance of PTLplan has been tested on some scenarios encountered in the literature. All tests were performed on a 300 MHz Pentium II. Table 3 presents the results of the experiments. Note that the use of search control formulae has a considerable impact, in particular in the three last scenarios. We also include some results documented for some other planners [Majercik and Littman, 1999; Onder and Pollack, 1999; Guéré and Alami, 1999] in order to give some rough indication about the relative merits of PTLplan.[3]

The *tiger scenario* was solved for success probabilities $\sigma = 0.5, 0.85, 0.939$. Only the marginally helpful control formula in example 6 was used. The *coffee robot scenario* [Boutilier and Poole, 1996; Onder and Pollack, 1999] involves a robot that is to go to a cafe in order to buy coffee with cream and sugar, and then bring the coffee back to the office. If it is raining (50% chance) the robot has to bring an umbrella along, otherwise not. A conjunction of 6 search control formulae were used in this scenario. The solution plan had two branches; the longest one had 8 steps. In the *asking coffee robot scenario* [Majercik and Littman, 1999], the robot does not have to bother about cream and sugar, but has to ask the user whether he wants coffee (50% chance). The solution had 4 branches. The *cold-room scenario* is a more complex, possibilistic scenario from [Guéré and Alami, 1999] involving a poorly illuminated table with two test tubes. A robot

[3]Unfortunately, the lack of agreed-upon, standard scenarios for conditional probabilistic/possibilistic planning makes systematic comparisons difficult. Also note that differences in hardware, small variations in how the scenarios were encoded etc, could account for some of the differences.

should move the red tube to a second table in a cold-room. A conjunction of 8 search control formulae were used in this scenario, and one of them is shown in example 5. The solution plan involved going to a third well-illuminated table to ensure that the correct test tube was picked up. It had two branches; the longest one had 14 steps.

7 Conclusions and future work

In this paper, we have presented work on the planner PTLplan. It is a progressive planner which utilizes strategic knowledge to reduce its search space. We have described PTLplan's fairly rich representation: actions with context-dependent and uncertain effects and observations, and plans with conditional branches. The semantics of this representation is based on associating different possible situations with degrees of possibility or probability. We have also described the planning algorithm and how we utilize a temporal logic to encode search control knowledge that can be used to prune unpromising branches in the search tree. PTLplan is based on TLplan [Bacchus and Kabanza, 2000] and ETLplan [Karlsson, 2001]. The novel contribution of PTLplan is the introduction of uncertainty into the context of progressive planning with strategic knowledge.

In this paper, we have also given some brief but quite promising performance data on PTLplan, indicating that it compares favorably to other probabilistic/possibilistic conditional planners. Yet, more remains to be done, both in order to improve the performance of PTLplan, and to evalute it more systematically, in particular on larger scenarios. Finally, we are now working on integrating PTLplan with a robotic system capable of action and perception [Coradeschi and Saffiotti, 2001].

Acknowledgements

This work was funded by the Swedish KK foundation. Many thanks to Silvia Coradeschi and Alessandro Saffiotti for helpful suggestions, and to Stephen Majercik for answering questions about ZANDER.

References

[AAAI99, 1999] *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, Orlando, Florida, 1999. AAAI Press, Menlo Park, California.

[Bacchus and Kabanza, 1996] F. Bacchus and F. Kabanza. Using temporal logic to control search in a forward chaining planner. In M. Ghallab and A. Milani, editors, *New Directions in Planning*, pages 141–153. IOS Press, Amsterdam, 1996.

[Bacchus and Kabanza, 2000] F. Bacchus and F. Kabanza. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, 116:123–191, 2000.

[Blum and Furst, 1995] A. Blum and M.L. Furst. Fast planning through planning graph analysis. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, Montreal, 1995. Morgan Kaufmann.

[Blythe and Veloso, 1997] J. Blythe and M. Veloso. Analogical replay for efficient conditional planning. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, Providence, Rhode Island, 1997. AAAI Press, Menlo Park, California.

[Boutilier and Poole, 1996] C. Boutilier and D. Poole. Computing optimal policies for partially observable decision processes using compact representations. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, Portland, Oregon, 1996. AAAI Press, Menlo Park, California.

[Coradeschi and Saffiotti, 2001] S. Coradeschi and A. Saffiotti. Perceptual anchoring of symbols for action. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, Seattle, 2001. Morgan Kaufmann.

[Draper *et al.*, 1994] D. Draper, S. Hanks, and D. Weld. Probabilistic planning with information gathering and contingent execution. In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems*. AAAI Press, Menlo Park, Calif., 1994.

[Dubois and Prade, 1988] D. Dubois and H. Prade. *Possibility theory — an approach to computerized processing of uncertainty*. Plenum Press, 1988.

[Emerson, 1990] E.A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B*, chapter 16, pages 997–1072. MIT Press, 1990.

[Guéré and Alami, 1999] E. Guéré and R. Alami. A possibilistic planner that deals with nondeterminism and contingency. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, Stockholm, 1999. Morgan Kaufmann.

[Kaebling *et al.*, 1998] L.P. Kaebling, M.L. Littman, and A.R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1–2):99–134, 1998.

[Karlsson, 2001] Lars Karlsson. Conditional progressive planning: a preliminary report. In Brian Mayoh, John Perram, and Henrik Hautop Lund, editors, *Proceedings of the Scandinavian Conference on Artificial Intelligence 2001*, 2001. To appear.

[Majercik and Littman, 1999] Stephen M. Majercik and Michael L. Littman. Contingent planning under uncertainty via stochastic satisfiability. In AAAI99 [1999], pages 549–556.

[Onder and Pollack, 1999] N. Onder and M.E. Pollack. Conditional, probabilistic planning: A unifying algorithm and effective search control mechanisms. In AAAI99 [1999], pages 577–584.

[Weld *et al.*, 1998] D.S. Weld, C.R. Anderson, and D.E. Smith. Extending graphplan to handle uncertainty and sensing actions. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, Madison, Wisconsin, 1998. AAAI Press, Menlo Park, California.