Handling Uncertainty by Interleaving Cost-Aware Classical Planning with Execution

Per Nyblom
Department of Computer and Information Science
Linköping university
perny@ida.liu.se

January 2, 2006

Abstract

This paper presents an algorithm for interleaved planning and execution in uncertain environments by reusing a classical planner capable of plan cost optimization. Probabilistic actions in the initial domain are split into deterministic ones that are presented to the planner. The planner generates a plan that is optimistic in the sense that it is assumed that all possible effects of the original probabilistic action can be achieved deterministically. The cost of the deterministic actions are used to control the output plans from the planner and are dynamically updated by generating and comparing all possible optimistic plans after the first occurence of a probabilistic action.

Appendix B3: Curriculum Vitae (Per Nyblom)

- M.Sc. in Computer Science and Engineering (Civilingenjörsexamen), Linköping University, 2003.
- Current employment: PhD student (doktorand), Linköping University, 2003 – present.

1 Introduction

To be able to handle a dynamic and uncertain universe during planning is essential in many real-world planning domains where some operators have non-deterministic or probabilistic effects. Many attempts to handle these types of domains have been directed towards the use of conditional planners [1] [17] that can generate contingency plans which anticipate all possible action outcomes. Other approaches include decision theoretic planners which model the planning problem as a Markov Decision Process (MDP) [18] or as a Partially Observable MDP (POMDP) [14] and generate a policy that maps all possible states to a suitable action. Although there have been many attempts to limit the search space or increase efficiency for these types of planners, they are only applicable to small problems.

Another approach for dealing with a non-deterministic world is to resolve the uncertainty by sensing and executing actions. Hybrid planning and execution systems like [12] [6] follow this approach where techniques similar to Hierarchical Task Network (HTN) planning is combined with execution. However, these systems are not capable of long-term planning and are often only used as a smart execution system in a larger context [20] [15].

There is a rich set of planning algorithms available that are not capable of dealing with uncertainty and/or probabilities. These algorithms are often well understood and have been used to implement several successful planning systems [4] [11]. Such planning systems can easily be used as a module to provide planning capabilities to an already existing system. It would be beneficial to be able to reuse these planners as components to provide relaxed search capabilities when developing a planning and execution system for domains that demand reasoning about uncertainty.

Since anticipating all possible (modeled) outcomes during planning seems to be unpractical, we want to create a system that is capable of limited probabilistic planning and uses execution of actions to resolve the remaining uncertainty. This paper presents an execution/replanning algorithm that uses a classical planner, capable of optimizing plan costs (which we call a *cost-aware* planner), for generating solutions to relaxed probabilistic planning problems. Our approach is limited in the sense that we are only trying to reach a set of goal states with as low expected cost as possible instead of generating a policy that either maximizes a reward function or a probability of reaching a set of goal states.

The paper is organised as follows. Section 2 contains background information about the planning problem we are trying to solve and introduces the notation used in this paper. Section 3 explains how actions with probabilistic effects can be handled and modelled with cost-aware classical planners by using probabilistic action decomposition, cost estimation for actions, execution and replanning. In section 4, some limitations of this approach are presented.

2 Preliminaries

By a *cost-aware* planner we mean a classical planner that can generate plans that are optimized in some way according to a cost function. A special case of such a planner is one that always generates optimal plans.

2.1 Classical Planning

We follow the definitions in [9] for the set-theoretic representation of a classical planning domain and problem. Let $L = \{p_1, ..., p_n\}$ be a finite set of *proposition* symbols. These propositions are the model of the world that the planner has to work with. A *state* s represents a possible world configuration and will always be a subset of L. The set of all possible world states S is defined as $S \subseteq 2^L$ (note that S is not defined as $S = 2^L$, since some of the states in 2^L might be impossible or unreachable).

A *planning domain* defines the possible transitions between the states in S. The set-theoretic planning domain on L is defined as a restricted state-transition system $\Sigma = \langle S, A, \gamma \rangle$ where A is the set of actions and $\gamma(s, a)$ defines the transitions between a state $s \in S$ when an action a is applied. Each action $a \in A$ is defined as a triple of subsets of L, $a = \langle$ precond(a), effects$^-(a)$, effects$^+(a)\rangle$ with the additional constraint that effects$^-(a) \cap$ effects$^+(a) = \emptyset$. An action a is said to be *applicable* to a state s if precond(a) $\subseteq s$.

The state-transition function is defined as $\gamma(s, a) = (s -$effects$^-(a)) \cup$ effects$^+(a)$ if $a \in A$ is applicable to $s \in S$. If a is not applicable to s, $\gamma(s, a)$ is undefined.

In this paper it is also assumed that every action is associated with a cost, cost(a), which can be used by a cost-aware planner.

A set-theoretic planning problem is a triple $P = (\Sigma, s_0, g)$ where $s_0 \in S$ is the initial state and $g \subseteq L$ is the set of goal propositions. The set of goal states, S_g, is defined as $\{s \in S \mid g \subseteq s\}$.

A *plan*, which is the output of the planner, is any sequence of actions $\pi = \langle a_1, ..., a_k \rangle$, where $k \geq 0$ (when the plan is empty, then $k = 0$). The state-transition function γ is extended to plans as well:
(1)

$$\gamma(s, \pi) = \begin{cases} s & \text{if } \pi \text{ is empty} \\ \gamma(\gamma(s, a_1), \langle a_2, ..., a_k \rangle) & \text{if } k > 0 \text{ and } a_1 \\ & \text{is applicable to } s \\ \text{undefined} & \text{otherwise} \end{cases}$$

A plan π is a *solution* to a classical planning problem if $g \subseteq \gamma(s_0, \pi)$.

2.2 Probabilistic planning

In a probabilistic planning domain, the actions are no longer assumed to be deterministic. An action has instead a set of *possible effects* which have probabilities associated with them.

The possible effects and the corresponding probabilities can be represented by a probability distribution $p(\cdot \mid s, a)$, which specifies the probabilities and the possible resulting states when applying an action a in a state s. To keep the representation as similar to classical planning as possible, we define a probabilistic action as a tuple $\langle$ precond(a), cost(a), $PE(a)$ $\rangle$ where $PE(a)$ (Possible Effects) is a set of tuples $\langle$ effects$_i^-(a)$, effects$_i^+(a)$, prob$_i(a)$ $\rangle$ which defines the probability distribution $p(\cdot \mid s, a)$. The precond(a), cost(a) and the different effects are used in a similar way to generate new states as defined in section 2.1.

We will differentiate between the *real cost* and the *expected additional* cost of actions a with $|PE(a)| > 1$. The expected additional cost (abbr. eaCost(a)) will be used as an instrument to guide the cost-aware planner towards plans with lower expected cost.

When actions can have several possible effects, a sequential plan is no longer a good solution to such a planning problem. A better plan format is to use a *conditional* plan which is tree-shaped in its simplest form but can also contain loops. The branches in a conditional plan contain *if-then-else* or *while* constructs which makes this plan format much more expressive than sequential plans. Another possible plan format, used for decision-theoretic planners, is a so called *policy* which is a mapping from every state s ($s \in S, S \subseteq 2^L$) to an action $a \in A$. With a policy it is possible to represent an agent's overall behavior instead of just the preferred paths to reach a set of goal states S_g.

We will not generate an explicit policy in our planning/execution algorithm since we are reusing a classical planner to generate plans for reaching *goals*. The overall behavior of our agent can on the other hand be interpreted as the execution of an implicit time-dependent policy that has a special zero-cost **no-op** action which is always applied when the agent is in a goal state.

2.3 Optimistic relaxation

Since we want to use a classical planner to perform probabilistic planning, the actions with many possible effects must be translated to deterministic ones before they are presented to the classical planner. The translation creates a new deterministic action for each possible effect, giving the classical planner the illusion that it can decide exactly what outcome a probabilistic action has. We will call the result of this translation for the *optimistic relaxation* of the probabilistic planning domain. A probabilistic action pa is called the *origin* of a deterministic action a if it represents one of pa's possible effects.

All the optimistic relaxations of a probabilistic action will have the same preconditions and costs as their origin.

The problem with the relaxation is that since the classical planner believes that the world is deterministic, it will generate plans that are optimistic (also called *weak*). An optimistic plan *can* reach the goal if all probabilistic actions have the expected effects, but it can also fail to reach it if any or several unexpected effects occur.

3 Handling Probabilistic Effects

A naive approach to interleaved planning and execution is to use a classical planner to solve the optimistic relaxation of the problem and then execute the optimistic plan until a probabilistic action (PA) does not give the expected effects. If this happens, a new optimistic plan could be generated from the current state.

This approach is however very risky for such domains where certain actions may lead to a state from where the goal can not be reached. A better approach is to use some form of *limited conditional planning* by examining a few of the possible failures of the optimistic plan. This can be achieved by simulating the optimistic plan until an optimistic relaxation of a PA occurs in the plan. The different outcomes of the PA can also be simulated, creating a set of possible successor states. For all these possible states, new optimistic plans could be generated. In some of the states it might not even be possible to reach the goal, or it might be very expensive.

With this limitied conditional planning approach, we have a method to *detect* possibly dangerous plans generated by a classical planner, but how do we make the planner *differentiate* between them?

One possible way is to remove the more risky actions from the optimistic relaxation, but this may be a very bad thing to do since the effect of an action with respect to the possibility of reaching the goal, depends on the state that the action is applied to, which means that we might filter out all solution plans by doing so.

We will instead use a cost-aware planner that can be manipulated by modifying action costs. Probabilistic action costs are divided into two parts: a *real* cost (the original cost of the PA) and an *expected additional* cost (eaCost). The eaCost(pa) for a probabilistic action pa is the cost that will be used to modify the behavior of the cost-aware planner. Actions that might lead to dead ends or expensive recovery plans should obviously have a higher expected additional cost to make the cost-aware planner compare it with other actions or plans for reaching the goal. We use the relative cost of the possible resulting optimistic plans to calculate the value the eaCost should have.

For example, consider a probabilistic action pa in a plan that can result in three possible optimistic plans $\langle \pi_1, \pi_2, \pi_3 \rangle$ with probabilities $\langle p_1, p_2, p_3 \rangle$ where π_1 is the plan with lowest cost. The eaCost(pa) is calculated as $p_2 \cdot (\text{cost}(\pi_2) - \text{cost}(\pi_1)) + p_3 \cdot (\text{cost}(\pi_3) - \text{cost}(\pi_1))$ which seems to be a reasonable value for the expected additional cost for the pa action. The cost of the optimistic relaxations of PAs are inherited and calculated by summing the real and the expected additional costs for their origin PAs.

The eaCost for a PA is initially set to zero, which means that when we detect an optimistic relaxation a of a probabilistic action pa in a plan with a non-zero eaCost(pa), we assume that the alternative outcomes of this action have already been considered. We can then proceed to execute the optimistic plan up to a.

A problem with this approach is that an action in classical planning represents a *set* of transitions in the underlying state transition system and this additional expected cost represents all of these state transitions, even if some of these transitions may have more or less impact on the total cost of the plan. This problem will be discussed in section 4.

3.1 Algorithm

The algorithm that performs the planning and execution procedure that was briefly described above, is given in procedure 1. The main procedure *optipep* (Optimistic Planning/Execution Procedure) is invoked by an agent in its estimated initial state s_0 given the probabilistic actions A_{prob} and the goal propositions g. It returns either *Done* (the goal is reached) or *Failure*.

It starts by generating the optimistic relaxation to the original probabilistic planning domain with the *genOptRelax* procedure. Next, it enters a loop where first an optimistic plan is generated by the cost-aware planner by applying the *solve* procedure. This plan is then simulated with the *simulateToPAction* procedure to the first optimistic version of a PA. If the expected additional cost for that action is zero, the optimistic plans are generated from all possible states that can be the result of applying the PA. The costs of all these plans are compared to the cheapest one and the expected additional cost for the PA is updated. If the first PA found has a non-zero eaCost, parts of the plan are executed with the *executeToPAction* procedure which stops after the first PA has been executed.

This process of generating an optimistic plan, calculate expected additional costs and executing plan fragments continues until a goal state is reached (*Done*) or no optimistic plan can be found (*Failure*).

The *genOptRelax* procedure takes every possible effect of each PA and creates a deterministic counterpart that is stored in the A_{opt} set, using the method presented in section 2.3. This procedure also sets up a map (*ActionMap*) from these deterministic actions to their probabilistic origin. The set A_{det} contains the deterministic actions in A_{prop}.

The *simulateToPAction* procedure (procedure 2) simulates the prefix of the plan generated by the planner until an action that has been created by the *genOptRelax* procedure is encountered. The result is a tuple containing the first PA (pa) found together with a set of $\langle$ action, result state, probability $\rangle$ tuples (ASP_{poss}). The result can also be *simulationDone*, which means that no PAs at all was encountered in the plan.

optipep keeps two different costs for each action. The first cost, rCost (the real cost), is the original cost in the probabilistic planning domain. The second cost, eaCost, is the expected additional cost. The cost for an optimistic classical action a is the sum of rCost(pa) and eaCost(pa) for its probabilistic action pa = ActionMap(a).

Procedure 1 optipep(A_{prop}, s_0, g)

```
$\langle A_{opt}, A_{det}$, ActionMap $\rangle \leftarrow$ genOptRelax$(A_{prop})$
$s \leftarrow s_0$
**loop**
  $\pi \leftarrow$ solve$(A_{opt} \cup A_{det}, s, g)$
  **if** $\pi$ = *NoPlan* **then**
    return *Failure*
  **end if**
  result $\leftarrow$simulateToPAction$(s, \pi)$
  **if** result = *simulationDone* **then**
    executePlan$(\pi)$
    return *Done*
  **else**
    $\langle pa, ASP_{poss} \rangle \leftarrow$ result
    **if** eaCost$(pa) = 0$ **then**
      minCost $\leftarrow \infty$
      **for all** $\langle a, t, p \rangle \in ASP_{poss}$ **do**
        $\pi_{poss} \leftarrow$ solve$(A_{opt} \cup A_{det}, t, g)$
        **if** minCost $>$ Cost$(\pi_{poss})$ **then**
          minCost $\leftarrow$ Cost$(\pi_{poss})$
        **end if**
        PP $\leftarrow$ PP $\cup \langle \pi_{poss}, p \rangle$
      **end for**
      {PP contains all possible plans and their probabilities}
      **for all** $\langle \pi, p \rangle \in$ PP **do**
        diffCost $\leftarrow$ cost($\pi$) $-$ minCost
        eaCost($pa$) $\leftarrow$ eaCost($pa$) $+p\cdot$ diffCost
      **end for**
      **for all** $\{a \mid$ ActionMap$(a) = pa\}$ **do**
        cost($a$) $\leftarrow$ rCost($pa$) + eaCost($pa$)
      **end for**
    **else**
      $s \leftarrow$ executeToPAction$(\pi)$
      **for all** $pa \in A_{prop}$ **do**
        eaCost($pa$) $\leftarrow 0$
      **end for**
    **end if**
  **end if**
**end loop**
```

Procedure 2 simulateToPAction(s, π)

```
$\pi_{current} \leftarrow \pi$
$a \leftarrow$ first$(\pi_{current})$
**while** $a \notin A_{opt}$ **do**
  $\pi_{current} \leftarrow$ rest$(\pi_{current})$
  **if** $\pi_{current} = \langle\rangle$ **then**
    return *simulationDone*
  **end if**
  $s \leftarrow \gamma(s, a)$
  $a \leftarrow$ first$(\pi_{current})$
**end while**{$a$ is now a member of $A_{opt}$}
$ASP_{poss} \leftarrow \emptyset$
$pa \leftarrow$ ActionMap$(a)$
**for all** $\{a \mid$ ActionMap$(a) = pa\}$ **do**
  $ASP_{poss} \leftarrow ASP_{poss} \cup \langle a, \gamma(s, a), prob(a) \rangle$
**end for**
return $\langle pa, ASP_{poss} \rangle$
```

3.2 Example

The replanning/execution algorithm presented in section 3.1 is probably best understood by an example. Suppose that a mobile robot can move between two locations **loc1** and **loc2** by executing the PA **move(robot, loc1, loc2)**. The possible outcomes for this action are the following. Either the robot reaches the location **loc2**, is stuck at **loc1** or breaks down (and needs to be repaired).

The robot has also some deterministic actions, **get-unstuck** and **call-repair-service**, which can be applied when it is stuck or broken respectively. These actions can be represented in Probabilistic PDDL (PPDDL) [16] (see figure 1).

The initial value (5) for the cost represents the estimated time it takes to execute the action when it is successful.

After applying the *genOptRelax* procedure to the **move** action, three new deterministic actions are created (see figure 2).

The cost-aware planner is then applied through the *solve* procedure. The result is an optimistic plan which contains a possible prefix of deterministic actions and a first relaxed probabilistic action (see figure 3). In this particular example we assume that the first such action is **move-1**.

The optimistic plan is then simulated with the *simulate-ToPAction* procedure which returns the first PA and a set of possible states that can be the result when this action is executed. In the example the outcomes are that the robot reaches the target, gets stuck or breaks down. For all these possible states a new optimistic plan is generated by the *solve* procedure. The planning and execution system now has access to a limited conditional plan (see figure 4) which anticipates just the possible outcomes of one PA. The question that needs to be answered now is if it is a good idea to apply the **move** operator at all? The algorithm tackles this problem by comparing the costs of all the possible plans to the cheapest one (the left branch in figure 4). These relative costs (*diffCost* in the algorithm) are then used to calculate

```
(:action move
  :precondition
   (and (not (broken rob1)) (not (stuck rob1))
        (at rob1 loc1) (path loc1 loc2))
  :effect
    (probabilistic
      0.80 (and (at rob1 loc2) (not (at rob1 loc1)))
      0.19 (stuck rob1)
      0.01 (broken rob1))
  :cost 5)

(:action get-unstuck
  :effect
   (not (stuck rob1))
  :cost 10)

(:action call-repair-service
  :effect
   (not (broken rob1))
  :cost 200)
```

Figure 1: Some actions specified in PPDDL

```
(:action move-1
  :precondition ;; Same as for move
  :effect (at rob1 loc2)
  :cost 5)

(:action move-2
  :precondition ;; Same as for move
  :effect (stuck rob1)
  :cost 5)

(:action move-3
  :precondition ;; Same as for move
  :effect (broken rob1)
  :cost 5)
```

Figure 2: The actions created by the *genOptRelax* procedure

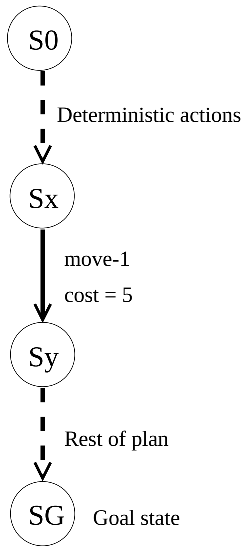

Figure 3: Optimistic plan generated by the planner

the eaCost for the **move** PA. In this example the expected additional cost becomes $15 \cdot 0.19 + 205 \cdot 0.01 = 4.9$ which is then added to the real cost for the **move** action which gives a total cost of 9.9.

The system now has a more realistic value for the cost of the **move** action and given this new information, a different plan might be generated by the planner. Therefore, a new optimistic plan is generated by the *solve* procedure which is again simulated by the *simulateToPAction* procedure. If the first PA (possibly **move** again) has a value for eaCost that is greater than zero, it is assumed that this action has a reasonable cost estimate and the optimistic result plan can be executed up to the **move** action. However, if the first PA has a zero eaCost, this cost has to be determined before any action can be executed (because the plan might look totally different before and after a cost update of an action).

4 Limitations

Since an action in classical planning represents a set of state transitions, there can be cases where applying the *optipep* procedure results in a bad execution behavior. Actions may for example have different effects on the resulting plan cost depending on the state they are invoked in. An example where this problem arises is in the *tire-world* domain (see figure 5) used in the probabilistic part of the International Planning Competition (IPC) in 2004 [5].

The actions in the planning domain consists of moving cars between different locations, loading tires on cars and changing tires. The only probabilistic actions are the instantiations of the **move-car** operator which can have possible effect with probability 0.15 of giving the car a flat tire. A flat tire can be changed with the **change-tire** action but only if the car has a spare tire loaded. Spare tires can be loaded

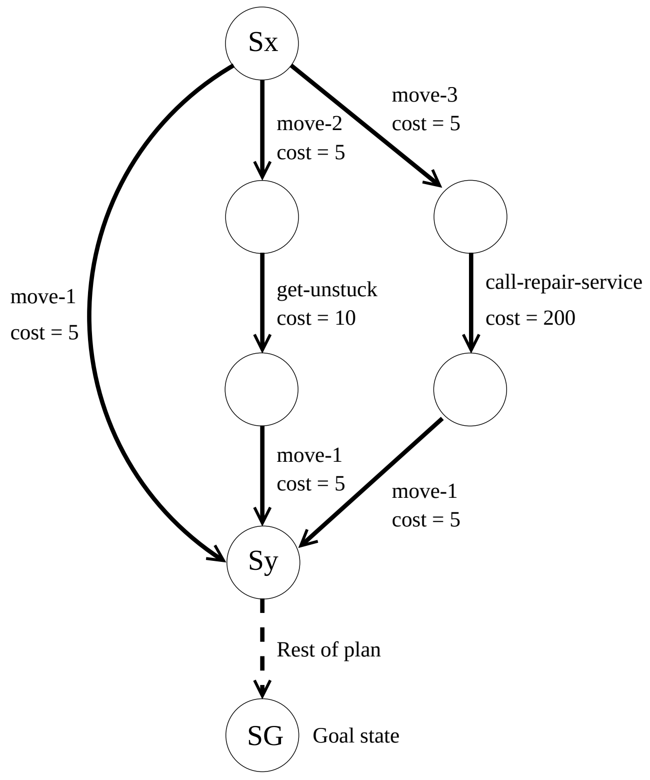

Figure 4: Optimistic plans for all possible outcomes from state **Sx**

on cars with **load-tire** actions but only at locations where such tires exists. The domain is interesting because even an optimal policy for a specific tire-world problem may fail because of bad luck while moving cars.

When *optipep* is applied to this domain, the action cost estimations for every **move-car** action instance might be very high except for those instances where the car ends up in a location where it can apply a **change-tire** action. The reason for this is that the different plans resulting after a **move-car** action depends heavily on the **has-spare-vehicle** and **has-spare-location** instantiated atoms.

These types of problems can be solved by preprocessing the domain and splitting the **move-car** actions into **move-car-with-tire** and **move-car-without-tire** but it is not a satisfactory solution in the general case.

A more general solution might be to automatically try to find the dependencies between parts of a state (propositions) and the resulting plan cost and create new actions when there is a difference above some pre-specified threshold.

5 Related Work

Interleaved planning and execution has sometimes been considered a necessary solution when dealing with the real world where execution not only resolves modelled uncer-

```
(:action move-car
  :parameters
    (?from - location ?to - location)
  :precondition
    (and (vehicle-at ?from) (road ?from ?to)
         (not (flat-tire)))
  :effect
    (and (vehicle-at ?to) (not (vehicle-at ?from))
 (probabilistic 0.15 (flat-tire))))

(:action load-tire
  :parameters (?loc - location)
  :precondition
    (and (vehicle-at ?loc) (has-spare-location ?loc)
 (not (has-spare-vehicle)))
  :effect
    (and (has-spare-vehicle)
         (not (has-spare-location ?loc))))

(:action change-tire
  :precondition (and (has-spare-vehicle) (flat-tire)
  :effect (and (not (has-spare-vehicle))
               (not (flat-tire))))
```

Figure 5: The operators in the *tire-world* domain

tainty but also deals with a limited model of the world. In [7], planning and execution is interleaved to reduce the search space for a conditional planner. A similar approach is taken in [2] where a conditional planner, MBP [1], is modified and reused in a planning/execution algorithm. The main differences between our approach and [2] are that we use optimistic plans to guide the search instead of information gain and we reuse a cost-aware planner instead of a conditional planner. Another difference is that we take probabilities and action costs into account.

Our approach to limited conditional planning is somewhat similar to N-fault tolerant planning [13] where conditional plans are generated that can handle at most N faults, making it a flexible method to control the search space/plan quality ratio. Instead of generating the conditional plan explicitly, we investigate the outcomes of probabilistic actions and modify the expected additional costs. N-fault tolerant planning does not consider the probabilities of the different action failures at all.

6 Summary and Future Work

We have presented an interleaved planning and execution procedure (*optipep*) that performs probabilistic planning by reusing a classical cost-aware planner. The probabilistic actions' costs are updated by comparing the possible cost of the plans for all outcomes of the first occurrence of a probabilistic action. The method has a limitation when the relative costs of the possible plans after applying a probabilistic action are very different depending on the state the action was applied in.

We have not performed any experiments with *optipep*, but we will test it on the Power Supply Restoration domain [19] and on several planning problems involving uncertain reasoning for our multi UAV system [3]. We also want to test our system for the domains and problems that were used in the probabilistic planning competition [5] and compare the result with the planners that participated.

Since there are several cost-aware planner *implementations* freely available [8] [10] [11], we will try to use them to implement the *solve* procedure in *optipep*.

7 Acknowledgement

This paper is partially funded by the Wallenberg Foundation under the WITAS UAV Project and the NFFP03-539 COMPAS Project.

References

[1] P. Bertoli, A. Cimatti, M. Pistore, M. Roveri, and P. Traverso. MBP: a model based planner. In *IJCAI'01 Workshop on Planning under Uncertainty and Incomplete Information*, 2001.

[2] P. Bertoli, A. Cimatti, and P. Traverso. Interleaving execution and planning for nondeterministic partially observable domains. In *Proceedings of ECAI-04*, 2004.

[3] P. Doherty, P. Haslum, F. Heintz, T. Merz, P. Nyblom, T. Persson, and B. Wingman. A distributed architecture for autonomous unmanned aerial vehicle experimentation. *7th International Symposium on Distributed Autonomous Robotic Systems (DARS)*, 2004.

[4] P. Doherty and J. Kvarnström. TALplanner: A temporal logic-based planner. *The AI Magazine*, 22(1):95–102, 2001.

[5] S. Edelkamp, J. Hoffmann, M. Littman, H. Younes, F. Bacchus, D. McDermott, M. Fox, D. Long, J. Rintanen, D. Smith, S. Thiebaux, and D. Weld. The international planning competition, 2004. See http://ls5-www.cs.uni-dortmund.de/∼edelkamp/ipc-4/.

[6] R. Firby. *Adaptive Execution in Complex Dynamic Worlds*. PhD thesis, Yale University, 1989.

[7] M. Genereseth and I. Nourbakhsh. Time-saving tips for problem solving with incomplete information. In *Proceedings of AAAI-93*, 1993.

[8] A. Gerevini and I. Serina. LPG: a planner based on local search for planning graphs. In *Proceedings of AIPS-02*, 2002.

[9] M. Ghallab, D. Nau, and P. Traverso. *Automated Planning, theory and practice.* Morgan Kaufmann Publishers, 2004.

[10] P. Haslum and H. Geffner. Heuristic planning with time and resources. In *IJCAI-01 Workshop on Planning with Resources*, 2001.

[11] J. Hoffmann. Extending FF to numerical state variables. In *Proceedings of ECAI-02*, 2002.

[12] F. Ingrand, M. Georgeff, and A. Rao. An architecture for real-time reasoning and system control. *IEEE Expert*, 7:34–44, 1992.

[13] R. Jensen, M. Veloso, and R. Bryant. Fault tolerant planning: Toward probabilistic uncertainty models in symbolic non-deterministic planning. In *Procedings of ICAPS-04*, 2004.

[14] L. Kaelbling, M. Littman, and A. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101, 1998.

[15] S. Lemai and F. Ingrand. Interleaving temporal planning and execution in robotics domains. In *Proceedings of AAAI-04*, 2004.

[16] M. Littman and H. Younes. PPDDL1.0 - an extension to PDDL for expressing planning domains with probabilistic effects. See http://www.cs.rutgers.edu/∼mlittman/topics/ipc04-pt/.

[17] R. Petrick and F. Bacchus. A knowledge-based approach to planning with incomplete information and sensing. In *Proceedings of AIPS-02*, 2002.

[18] M. Puterman. *Markov Decision Processes, Discrete Stochastic Dynamic Programming.* Wiley Interscience, 1994.

[19] S. Thiebaux and M. Cordie. Supply restoration in power distribution systems – a benchmark for planning under uncertainty. In *Proceedings of ECP-01*, 2001.

[20] D. E. Wilkins, K. L. Myers, J. D. Lowrance, and L. P. Wesley. Planning and reacting in uncertain and dynamic environments. *Journal of Experimental and Theoretical AI*, 7(1):197–227, 1995.