Expert Systems

Review Article

DOI: 10.1111/exsy.12037

Progressive heuristic search for probabilistic planning based on interaction estimates

Yolanda E-Martín,[1,2] María D. R-Moreno[2] and David E. Smith[3]

(1) Universities Space Research Association (USRA), 615 National Avenue, Suite 220, Mountain View, CA, 94043
E-mail: yolanda@aut.uah.es
(2) Departamento de Automática, Universidad de Alcalá, Carretera Madrid-Barcelona Km 33.6 28871, Alcalá de Henares, Madrid, Spain
E-mail: mdolores@aut.uah.es
(3) Intelligent Systems Division, NASA Ames Research Center, Moffet Field, CA 94035-1000
E-mail: david.smith@nasa.gov

Abstract: *Development of real planning and scheduling applications often requires the ability to handle uncertainty. Often this uncertainty is represented using probability information on the initial conditions and on the outcomes of actions. In this paper, we describe a novel probabilistic plan graph heuristic that computes information about the interaction between actions and between propositions. This information is used to find better relaxed plans and to compute their probability of success. This information guides a forward state space search for high probability, non-branching seed plans. These plans are then used in a planning and scheduling system that handles unexpected outcomes by runtime replanning. We briefly describe the heuristic, the search process, and the results on different domains from recent international planning competitions. We discuss the results of this study and some of the issues involved in advancing this work further.*

Keywords: probabilistic planning, planning graphs, interaction, heuristic search

1. Introduction

Automated planning is the process of choosing and organizing actions into plans to achieve goals. Initially, planning research focused on developing planning systems capable of solving problems that assume perfect worlds. This approach is called deterministic planning (Ghallab *et al.*, 2004) and is based on the assumption of complete knowledge of the initial conditions and the effects of actions. Significant advances have been made in recent years in the size and complexity of deterministic problems that can be solved with these planners. However, in real problems, unexpected events can occur, actions can have unexpected outcomes and the state of the world may not be known with certainty. If a deterministic planner is used to solve such problems, the execution of the plan may fail because the plan does not take into account the possible contingencies. To address this problem, it is important to develop planners capable of handling uncertainty in the domain.

A line of research dealing with planning problems under uncertainty is probabilistic planning. This approach describes the uncertainty using probability distributions and assumes that results of the actions are fully observable. Probabilistic Planning Domain Definition Language (PPDDL) (Younes *et al.*, 2005), an extension of the Planning Domain Definition Language (PDDL), is capable of representing probabilistic planning problems by allowing probabilities on initial conditions and on action outcomes.

In the literature, several paradigms to solve probabilistic planning problems are described. Traditional approaches to probabilistic planning involve standard Markov decision processes that find a policy (Boutilier *et al.*, 1999; Hansen & Zilberstein, 2001; Bonet & Geffner, 2003). Although these methods provide robust plans, they are computationally expensive because the solution includes a contingency branch for every possible outcome that may occur, given the actions in the plan. Other probabilistic planners use plan graphs to compute estimates of probability that propositions can be achieved and actions can be performed (Blum & Langford, 1999; Bryce *et al.*, 2006; Little & Thiébaux, 2006). This information can be used to guide a probabilistic planner towards the most likely plan for achieving the goals. Another recent technique consists of transforming the given probabilistic planning problem into a deterministic planning problem. This method uses deterministic planners to solve the problem and runtime replanning to deal with unexpected states (Yoon *et al.*, 2007; Teichteil-Königsbuch *et al.*, 2010). The FF-Replan planner has been the pioneer in this paradigm, and it has proven very successful. It won the first two probabilistic planning competitions held in 2004 (Younes *et al.*, 2005) and 2006 (Bonet & Given, 2006). (In the 2006 competition, it was not submitted, but the organizers ran it for comparison). Despite the success of FF-Replan, it does not make use of the probabilistic information in the domain description, which may result in frequent replanning. For this reason, the work presented in (Jiménez *et al.* 2006), which follows the same line as FF-Replan, turns the probability information into costs and searches for a lowest cost plan using a deterministic optimizing planner.

Our work involves the use of probability information in a heuristic function to guide a deterministic planning search. Concretely, the approach constructs high probability, non-branching seed plans. These plans can be used in a system that handles unexpected outcomes by runtime replanning (E-Martín *et al.*, 2011). The plans can also be incrementally augmented with contingency branches for critical action outcomes.

This approach has been implemented in the Parallel Integrated Planning and Scheduling System (PIPSS) (Plaza *et al.*, 2008). PIPSS combines the Heuristic Progressive Planner (HPP) (R-Moreno *et al.*, 2005) and the Object-Oriented SCheduling ARquitecture (O-OSCAR) (Cesta *et al.*, 2001). HPP is a forward heuristic planner that uses a relaxed plan heuristic to guide the search process (Hoffmann & Nebel, 2001). To find high probability seed plans, we propagate probability estimates through a plan graph and use this to construct high probability relaxed plans. This information is used to guide the planning process. These probability estimates are more accurate than traditional methods, as they make use of the notion of interaction introduced by Bryce and Smith (2006). Interaction captures the degree of dependence (positive or negative) between pairs of propositions and pairs of actions in a plan graph.

This paper starts by describing the translation technique from probabilistic planning domains into deterministic domains. Then, we describe our probabilistic estimator based on a plan graph and the relaxed plan extraction procedure that guides the search. We follow with an empirical study of these new techniques within PIPSS and compare with some other probabilistic planners. Finally, some related work, conclusions and future work are discussed.

2. Architecture

As mentioned previously, this approach has been implemented in PIPSS. It uses a forward heuristic planner that bases its heuristic function on a relaxed plan. Figure 1 shows the architecture of the planner that highlights the key features of the system: *PPDDL-PDDL Conversion*, *Plan Graph Estimator* and *Heuristic Computation* modules. Given a PPDDL problem, the system initially translates the probabilistic problem into a deterministic one by using the technique explained in the next section. Then, the *Analysis & Processing* module processes and loads all the information given in the domain, and encodes the strings as numbers to decrease the computation time. The system builds the plan graph estimator as described in the subsequent section. After this step, the search process starts. The system performs an A* search when it is looking for a solution. For each state, the plan graph is updated, and a relaxed plan is created to estimate the cost (probability) of achieving the goals from that state. This estimation is called a completion cost estimate (CCE).

3. Conversion from PPDDL to PDDL

To convert from PPDDL to PDDL, we follow the approach of Jiménez, Coles, and Smith (2006). In general, the process consists of generating a deterministic action for each probabilistic effect of a probabilistic action. For each new action created, the probability of its outcomes is transformed into an additive cost equal to the negative logarithm of the probability:

$$C_i \;=\; -Ln(P_i) \tag{1}$$

This cost is used to compute the probability of each proposition and action.

More precisely, if A is a probabilistic action with outcomes $O_1,\ldots,O_i$ with probabilities $P_1,\ldots,P_i$ respectively, we create a new deterministic action for each outcome. Each deterministic action A_i has all the preconditions of A. If the outcome O_i is conditional, then A_i will also have additional preconditions corresponding to the conditions of O_i. The effects of A_i are the effects in the outcome O_i, and the cost of A_i is given by equation (1).

Figure 2 shows an example of the conversion strategy. Figure 2(a) shows the probabilistic action `move-car` that has two outcomes leading to two deterministic actions. The most likely actions outcome implies that the car successfully drives from one position to another. However, in the second outcome, the car achieves the destination but gets a flat tire. The conversion process builds the two deterministic actions (shown in Figure 2(b) and 2(c)), one for each possible outcome, and generates the following additive costs $C_a = -Ln(0.4) \approx 0.916$ and $C_b = -Ln(0.6) \approx 0.511$.

4. Plan graph cost heuristic

In this section, we describe an approach to computing more accurate estimates of probability based on the work by Bryce and Smith (2006), which allows the planner to guide the search towards high probability seed plans.

Given the deterministic actions created by the technique described in the previous section, we build a plan graph and propagate the cost (probability) information through it. We start from the initial conditions and work progressively forward through each successive layer of the plan graph. The cost of performing an action is equal to the cost of achieving its preconditions. The cost of achieving a proposition in the next level of the plan graph is the minimum cost among all the actions of the previous layer that generate the proposition. Typically, the cost of achieving a set of preconditions for an action is taken to be the sum of the costs of achieving the propositions. However, this can be an underestimate if some of the preconditions interfere with each other and can be an overestimate if some of the preconditions are achieved by the same action. For this reason, we introduce the notion of *interaction* (I), which captures the degree of dependence (positive or negative) between pairs of propositions and actions in the plan graph (Bryce & Smith, 2006).

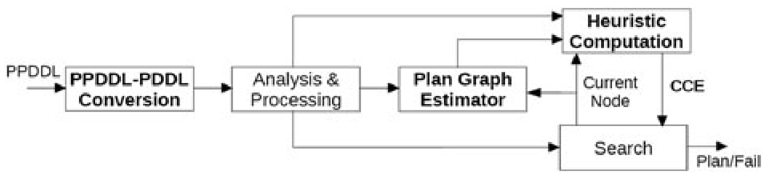


Figure 1: *PIPSS$_I$ system architecture.*

```
(:action move-car
 :parameters (?from - loc ?to - loc)
 :precondition (and (at ?from) (road ?from ?to) (not (flattire)))
 :effect (and (at ?to) (not (at ?from))
           (probabilistic 0.4 (flattire))))
```

(a)

```
(:action move-car-a
 :parameters (?from - loc ?to - loc)
 :precondition (and (at ?from) (road ?from ?to) (not (flattire)))
 :effect (and (at ?to) (not (at ?from)) (flattire)
           (increase (risk) 0.916291)))
```

(b)

```
(:action move-car-b
 :parameters (?from - loc ?to - loc)
 :precondition (and (at ?from) (road ?from ?to) (not (flattire)))
 :effect (and (at ?to) (not (at ?from)) (increase (risk) 0.510826)))
```

(c)

Figure 2: *Determinization of a probabilistic action. (a) Probabilistic action in PPDDL, (b) Deterministic action a, and (c) Deterministic action b.*

This section describes the method used to create the cost plan graph. The process consists of building a plan graph (Blum & Furst, 1997) in which the mutex calculation is replaced with interaction calculation.

The computation of cost and interaction information begins at level zero of the plan graph and sequentially proceeds to higher levels. For level zero, we assume that (1) the cost of each proposition at this level is $-Ln(Pr)$, where Pr has a value of 1 for each proposition if the initial state is fully known and (2) the interaction between each pair of propositions is 0; that is, the propositions are independent. Neither of these assumptions are essential, but we adopt them here for simplicity.

In the following sections, we briefly describe a planning graph structure. Then, we introduce the interaction notion. After that, we give the details of how to do this calculation beginning at the initial proposition layer and working forward to actions, and finally to the next proposition layer.

4.1. Planning graph construction

A planning graph is a structure that provides an efficient method to estimate the set of propositions that are reachable and the actions that can be performed in a planning problem starting in a particular state. It is a directed graph represented by levels, each one composed of two layers. Starting at level 0, the even layers contain proposition nodes that are the reachable propositions. The odd layers contain action nodes whose preconditions are present at the previous level. The arcs represent the relations among facts and actions. An edge from a proposition node to an action node means that the proposition is one of the action's preconditions. An edge from an action node to a proposition node means that the proposition is an effect of the action.

Plan graph expansion extends the planning graph forward until all goals are reached. At level 0, the proposition layer is made up of the set of propositions in the initial state. The action layer contains all actions applicable to the initial state. The union of all those actions' add and delete effects with the propositions that are already in the previous level forms the second proposition layer. The process is repeated until a proposition layer that contains all the goal propositions is reached. As the planning graph represents parallel propositions and actions that can be mutually exclusive, this phase considers the relations that can exist between pairs of propositions and pairs of actions. Specifically, the algorithm defines the following mutex relations (see also Figure 3):

- Between pairs of operators:
  - *Inconsistent effects*: the effect of one operator deletes a proposition that is added by the other.
  - *Interference*: one action's effect deletes a precondition of the other.
  - *Competing needs*: the two operators have mutex preconditions.
- Between pairs of propositions:
  - *Contradiction*: one proposition is the negation of the other.
  - *Inconsistent support*: all the ways for obtaining both propositions are mutex.

While we use this basic plan graph construction algorithm, we replace the mutex calculations by the calculation of *interactions* as described in the next section.

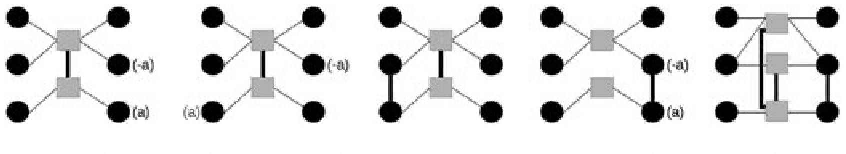


Figure 3: *Graphical description of mutex relations. Circles represent propositions, and squares refer to actions. Thick lines define mutex relations between two elements. From left to right, the different kinds of mutex are inconsistent effects, interference, competing needs, contradiction, and inconsistent support.*

4.2. Interaction

Interaction is a value that represents how more or less costly (probable) it is that two propositions or actions are established together instead of independently. This concept is a generalization of the mutual exclusion concept used in classical plan graphs. Formally, the interaction, I, between two propositions or two actions (p and q) is defined as follows:

$$I(p,\ q) = Cost(p \wedge q) - (Cost(p) + Cost(q)) \tag{2}$$

It has the following features:

$$I(p,\ q)\ is \begin{cases} < 0 & \text{if } p \text{ and } q \text{ are synergistic} \\ = 0 & \text{if } p \text{ and } q \text{ are independent} \\ > 0 & \text{if } p \text{ and } q \text{ interfere} \end{cases}$$

That is, I provides information about the degree of interference or synergy between pairs of propositions and pairs of actions in a plan graph. When $I(p,\ q) < 0$ (synergy), this means that the cost of establishing both p and q is less than the sum of the costs for establishing the two independently. However, this cost cannot be less than the cost of establishing the most difficult of p and q. As a result, $I(p,\ q)$ is bounded below by $-\min[cost(p), cost(q)]$. Similarly, $0 < I(p,\ q) < \infty$ means that there is some interference between the best plans for achieving p and q, so it is harder (more costly) to achieve them both than to achieve them independently. In the extreme case, $I = \infty$, the propositions or actions are mutually exclusive.

Interaction is important because it provides information about the relation (independence, interference, or synergy) between a pair of propositions or a pair of actions at each level of the plan graph. For this reason, instead of computing mutex information in the plan graph, we compute interaction information between all pairs of propositions and all pairs of actions at each level. Interaction is taken into account in the cost propagation. In this way, we can establish a better estimation of the cost for two propositions or two actions performed at the same time.

It is important to note that the costs and interaction values propagated in the plan graph are approximations. The calculation of these values is limited to pairs of propositions and pairs of actions in each level of a plan graph – in other words, binary interaction. For exact calculation, we would need to consider n-ary interaction relationships among propositions and among actions in the plan graph. This would be computationally prohibitive. As a result, we limit the calculation to binary interaction.

4.3. Computing action cost and interaction

Let us suppose that we have the cost and interaction information for propositions at a given level of the plan graph. We use this information to compute the cost and the interaction information for the subsequent action layer. Considering an action a at level l with a set of preconditions $prec_a$, the cost that an action is executed is the cost that all the preconditions are achieved plus the interaction between all pairs of preconditions:

$$Cost(a) = Cost(prec_a) \approx \sum_{x_i \in prec_a} Cost(x_i) + \sum_{\substack{(x_i, x_j) \in prec_a \\ j > i}} I(x_i, x_j) \tag{3}$$

As an example, consider one level of the plan graph shown in Figure 4, where we have three propositions p, q, and r with costs 0.53, 0.19, and 0.8, respectively, and interaction values $I(p,\ q) = -0.1$, $I(p,\ r) = 0.05$, and $I(q,\ r) = 1.2$ at level i. There are also two actions c and d that have outcomes with costs 0.22 and 0.51, respectively (these costs are the negative logarithm of the probabilities given in the domain for those outcomes, which are given in the domain). The numbers above the propositions and actions are the costs associated with each one computed during the cost propagation process (those highlighted are the costs that will be computed in subsequent sections).

For the example shown in Figure 4, the cost for actions c and d would be:

$$\begin{aligned} Cost(c) &\approx Cost(p) + Cost(q) + I(p,\ q) = 0.53 + 0.19 - 0.1 = 0.62 \\ Cost(d) &\approx Cost(q) + Cost(r) + I(q,\ r) = 0.19 + 0.8 + 1.2 = 2.19 \end{aligned}$$

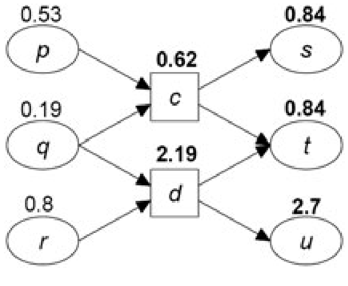


Figure 4: *A plan graph level with costs and interaction calculation and propagation.*

The next step is to compute the interaction between actions. If the actions are mutex by inconsistent effects or interference, then the interaction is ∞. Otherwise, the cost of performing two actions a and b will be the sum of their individual costs plus the cost of the interaction between their preconditions. We can define the interaction between two actions a and b at level l, with sets of preconditions $prec_a$ and $prec_b$, as follows:

$$I(a,\ b) = \begin{cases} \infty & \text{if } a \text{ and } b \text{ are mutex by inconsistent effects or interference} \\ Cost(a \wedge b) - Cost(a) - Cost(b) & \text{otherwise} \end{cases} \tag{4}$$

where in the second case, the interaction can be rewritten as follows:

$$\begin{aligned} I(a,\ b) &= Cost(a \wedge b) - Cost(a) - Cost(b) \\ &= Cost(prec_a \cup prec_b) - Cost(prec_a) - Cost(prec_b) \\ &\approx \sum_{\substack{x_i \in prec_a - prec_b \\ x_j \in prec_b - prec_a}} I(x_i, x_j) - \left[\sum_{x_i \in prec_a \cap prec_b} Cost(x_i) + \sum_{\substack{(x_i, x_j) \in prec_a \cap prec_b \\ j > i}} I(x_i, x_j) \right] \end{aligned}$$

For the example in Figure 4, the interaction between actions c and d would be:

$$I(c,\ d) \approx I(p,\ r) - Cost(q) = 0.05 - 0.19 = -0.14$$

The fact that $I(c,\ d) = -0.14$ means that there is some degree of synergy between actions c and d. This synergy comes from the fact that the two actions have a common precondition, q. However, this synergy is tempered because of the interference between the precondition p of c and the precondition r of d.

4.4. *Computing proposition cost and interaction*

The next step consists of calculating the cost of the propositions at the next level. In this calculation, we need to consider all the possible actions at the previous level that achieve each proposition. We make the usual optimistic assumption that we can use the least expensive action, so the cost is the minimum over the costs of all the actions that can achieve the proposition. More formally, for a proposition x at level l, achieved by the actions $Ach(x)$ at the preceding level, the cost is calculated as follows:

$$Cost(x) = \min_{a \in Ach(x)} [Cost(a) + Cost(x|a)] \tag{5}$$

where $Cost(x\,|\,a)$ is the cost of the outcome x for the action a.

In our example, the cost of the proposition t of the graph is

$$\begin{aligned} Cost(t) &= \min[Cost(c) + Cost(t|c),\ Cost(d) + Cost(t|d)] \\ &= \min[0.62 + 0.22,\ 2.19 + 0.51] \\ &= \min[0.84,\ 2.7] = 0.84 \end{aligned}$$

Finally, we consider the interaction between a pair of propositions x and y. To compute the interaction between two propositions at a level l, we need to consider all possible ways of achieving those propositions at the previous level. That is, all the actions that achieve the pair of propositions and the interaction between them. Suppose that $Ach(x)$ and $Ach(y)$ are the sets of actions that achieve the propositions x and y at level l. The interaction between x and y is then:

$$I(x,\ y) \approx \min_{\substack{a \in Ach(x) \\ b \in Ach(y)}} \left[Cost(a) + Cost(b) + I(a,b) + Cost(x|a) + Cost(y|b) \right] - Cost(x) - Cost(y) \tag{6}$$

Note that when $a = b$, the interaction $I(a,b) = -Cost(b)$, so the expression $Cost(a) + Cost(b) + I(a,\ b)$ is just $Cost(a)$.

Returning to our example, consider the calculation of the interaction between s and t where the possible ways to achieve both are performing c or c and d. In this case, the interaction is calculated as follows:

$$\begin{aligned} I(s,t) &\approx \min[Cost(c) + Cost(s\wedge t|c),\ \ Cost(c) + Cost(d) + I(c,d) + Cost(s|c) + Cost(t|d)] - Cost(s) - Cost(t) \\ &= \min[0.84,\ 3.4] - 0.84 - 0.84 = 0.84 - 0.84 - 0.84 = -0.84 \end{aligned}$$

In this case, it is less costly to establish the propositions s and t through action c than c and d. This makes sense because executing a single action should have lower cost than performing two.

Taking the aforementioned calculations into consideration, we build a cost plan graph in the same way that an ordinary plan graph is created, replacing the mutex calculation with interaction calculation. The cost estimates for each action and proposition that appear in the plan graph, and the interaction value of every pair of propositions and every pair of operators are then used to compute the heuristic estimates that guide the planner towards low-cost (high-probability) plans, as explained in the next section.

5. Heuristic search guidance

In this section, we describe the probabilistic plan graph heuristic that guides the planner towards lower-cost (higher probability) plans.

The planner performs an A* search where the function evaluation is the sum of the cost from the starting node to the current node (n) plus an estimation of the cost of going from the current node to the goals, which we call the CCE. That is,

$$f(n) = g(n) + CCE(n) \tag{7}$$

The CCE is computed using the probabilistic plan graph. For each state in the search, the plan graph is updated, and we compute a relaxed plan to estimate the probability of achieving the goals from that state. The relaxed plan is computed by performing regression on the cost plan graph developed in the previous sections. This algorithm makes use of the cost and interaction information to make better choices for actions. In particular, the algorithm orders the goals at each level according to cost and chooses the operator used to achieve each goal based on cost. More precisely,

- Arrange goals: the goals at each level are arranged from the highest cost to the lowest. That is, we begin analyzing the most expensive (least probable) proposition. In other words, we try to solve the more difficult goals as soon as possible.
- Choose operators: if there is more than one operator that achieves a particular goal at a level, we choose the operator with the lowest cost (highest probability) given the other operators that have already been chosen at that level. Suppose that O is the set of operators selected in level l, and $Ach(g)$ is the set of actions that achieve the current goal g at level l. The operator we choose is:

  $$\underset{a\in Ach(g)}{\arg\min}\left[Cost(a) + Cost(g|a) + \sum_{b\in O} I(a,\ b)\right] \tag{8}$$

  As a result of this, we greedily choose a more synergistic and less costly set of operators.

Figure 5 shows the high-level algorithm used in the relaxed plan regression phase.

To illustrate the computation of a CCE, consider a domain problem with operators A, B, and C that have the following costs: $Cost(y\,|\,A) = 0.25$, $Cost(z\,|\,B) = 0.22$, and $Cost(z\,|\,C) = 0.42$. Figure 6 shows a fragment of the plan graph that belongs to a certain node n in the search space. Every operator and action in the fragment has an associated cost value. Suppose that the set of goals is composed of propositions y and z with estimated costs 0.85 and 1.07, respectively. The procedure first deals with z, which has the higher cost goal. This proposition is achieved by actions B and C. The action chosen would be B because it has the lowest cost:

$$\begin{aligned} &\underset{a\in\{B,\ C\}}{\arg\min} && [Cost(a) + Cost(z|a)] \\ & && Cost(B) + Cost(z|B) = 0.85 + 0.22 = 1.07 \\ & && Cost(C) + Cost(z|C) = 1.55 + 0.42 = 1.97 \\ &\text{therefore:} && \\ &\underset{a\in\{B,\ C\}}{\arg\min} && [Cost(a) + Cost(z|a)] = B \end{aligned}$$

Continuing with the next goal, y, it is achieved by operator A and noop-y. To know which is the best choice, the cost of achieving y must be computed for both of these operators assuming operator B. Considering the action cost values shown in Figure 6 and the interactions $I(A,\ B) = 0$ and $I(\text{noop-y},\ B) = -0.85$, the chosen operator would be:

$$\begin{aligned} &\underset{a\in\{\text{noop-y},\ A\}}{\arg\min} && [Cost(a) + Cost(y|a) + I(a,B))] \\ & && Cost(\text{noop-y}) + Cost(y|\text{noop-y}) + I(\text{noop-y},\ B) = 0.85 - 0.85 = 0 \\ & && Cost(A) + Cost(y|A) + I(A,\ B) = 0.6 + 0.25 = 0.85 \\ &\text{therefore :} && \\ &\underset{a\in\{\text{noop-y},\ A\}}{\arg\min} && [Cost(a) + Cost(y|a) + I(a,\ B))] = \text{noop-y} \end{aligned}$$

Function COSTESTIMATE (*G,l*)

G_l	$\equiv$	the set of goals at level l in the relaxed plan graph
g	$\equiv$	a goal proposition
l	$\equiv$	number of levels in the relaxed plan graph
A_l	$\equiv$	the set of actions at level l for a specific goal
a	$\equiv$	an action
O_l	$\equiv$	the set of operators selected at level l
π	$\equiv$	the set of actions selected
CCE	$\equiv$	the completion cost estimate of the current node

```
1.  while $l > 0$
2.      $O_l \leftarrow \emptyset$
3.      $G_{l-1} \leftarrow \emptyset$
4.      while $G_l \neq \emptyset$
5.          $g \leftarrow \underset{g \in G_l}{\text{argmax}}\,(Cost(g))$
6.          $A_l \leftarrow \{a : g \in effect^{+}(a)\}$
7.          $a \leftarrow \underset{a \in A_l}{\text{argmin}} \left[Cost(a) + Cost(g|a) + \sum_{b \in O_l} I(a,b)\right]$
8.          $O_l \leftarrow O_l \cup \{a\}$
9.          $\pi \leftarrow \pi \cup \{a\}$
10.         $G_{l-1} \leftarrow G_{l-1} \cup preconditions(a)$
11.         $G_l \leftarrow G_l - \{g\}$
12.     $l \leftarrow l - 1$
13. CCE(currentNode) $\leftarrow \sum_{i=1..n} Cost(\pi_i)$
14. return CCE
```

Figure 5: *The relaxed plan regression pseudo-algorithm.*

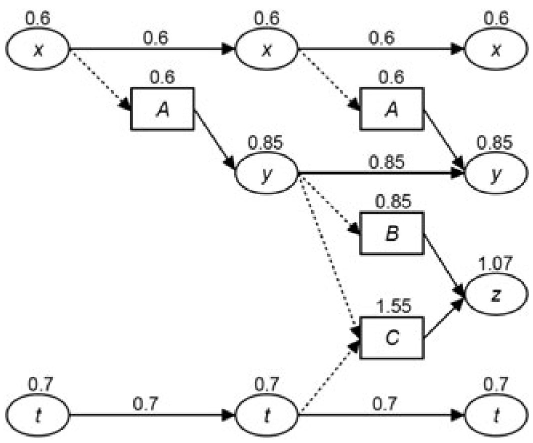


Figure 6: *Fragment of a plan graph.*

In this case, the selected operator is noop-y, because it would have the lowest cost when operator *B* has already been chosen.

The same technique is applied for the rest of the layers until the initial state is reached. Once the plan is extracted, we compute the heuristic estimation, which is the sum of the costs of every action outcome selected in the relaxed plan regression algorithm. In this way, we are considering the cost and interaction information that has been computed previously in the relaxed plan extraction.

Continuing with the example, suppose that the plan solution selected is composed of operators *A* and *B*, the CCE value would be:

$$CCE(n) = Cost(y|A) + Cost(z|B) = 0.25 + 0.22 = 0.47$$

This indicates that the estimated cost to reach the goal from that state is 0.47, that is, a probability of $e^{(-0.47)} = 0.62$.

6. Experimental results

In this section, we describe the experiments that demonstrate that the plan graph cost heuristic using interaction guides the search to high probability solutions for probabilistic planning problems.

We have conducted an experimental evaluation on International Probabilistic Planning Competition (IPPC-06) (Bonet & Given, 2006) and IPPC-08 (Buffet & Bryce, 2006) fully observable probabilistic (FOP) planning domains as well as the 'probabilistically interesting' domains introduced by Little and Thiebaux (2007). The test consists of running the planner and using the resulting plan in the MDP Simulator (Younes *et al.*, 2005). The planner and the simulator communicate by exchanging messages. The simulator first sends the planner the initial state. The planner then sends an action to the simulator, which sends back a resulting state. Because the simulator is stochastic, the resulting state can vary from one run to the next, so each plan must be run many times to obtain an expected value. The experiments were conducted on an Intel Core 2 Quad processor running at 2.33 GHz with 8 GB of RAM.

The following section is a brief description of the domains used for the experimental evaluation.

6.1. IPPC-06

For IPPC-06, we are concerned with the probabilistic planning domains having FOP track action outcomes:

- Blocksworld: similar to the classical Blocksworld but with additional actions. A gripper can hold a block or a tower of blocks or be empty. When trying to perform an action, there is a chance of dropping a block on the table.
- Exploding-Blocksworld: a dead-end version of the Blocksworld domain described earlier where additionally the blocks can explode. The explosion may affect the table or other blocks.
- Elevators: this domain consists of a set of coins arranged on different levels. To collect them, the elevators can move among the levels. The movements can fail if the elevator falls down the shaft and finishes on a lower level.
- Random: this domain is randomly generated with random preconditions and effects. A special reset action can lead any state to the initial state, making it dead-end free.
- Tireworld: in this domain, a car has to move between two locations. When the car drives a segment of the route, there is the chance of getting a flat tire. When this occurs, the tire must be replaced. However, spare tires are not available in all locations.
- Zenotravel: this domain has actions to embark and disembark passengers from an aircraft that can fly at two alternative speeds between locations. The actions have a probability of failing without causing any effects. So, actions must sometimes be repeated.

6.2. IPPC-08

For IPPC-08, we are again concerned with the probabilistic planning domains having FOP track action outcomes:

- Blocksworld: similar to the IPPC-06 Blocksworld domain.
- Exploding-Blocksworld: similar to the IPPC-06 Exploding-Blocksworld domain.
- Triangle-tireworld: similar to the IPPC-06 Tire World domain but with slight differences in the definition to permit short but dangerous paths.

- Rectangle-tireworld: this domain is inspired by Triangle-tireworld, except that there are no spare tires available. However, the domain defines an action that when the car gets a flat tire, it can still go everywhere but at low probability.
- Zenotravel: similar to the IPPC-06 Zenotravel domain.

6.3. *Probabilistically interesting domains*

Little and Thiébaux (2007) have created a number of very simple problems that explore the issue of probabilistic planning versus replanning. These problems lead to dead ends. The domains vary by (1) the number of dead-end states where the goal is unreachable, (2) the degree to which the probability of reaching a dead-end state can be reduced through the choice of actions, (3) the number of distinct trajectories from which the goal can be reached from the initial state, and (4) the presence of mutually exclusive choices that prevent alternative courses of actions.

- Climb: this domain consists of a person who is stuck on a roof because the ladder they used to climb up has fallen down. There are two options to get down: climbing down without the ladder but with a certain risk of injury or death, or calling for help from someone below to bring the ladder and then climb down with the ladder, which has no risk.
- River: in this domain, a person on one side of the river needs to cross to the other side. There are three ways to achieve the goal with different chances of success.
- Tire1 & Tire10: domains based on the Tireworld domain, with different levels of difficulty to reach the final location without getting a flat tire.

The following planners have been used for the experimental evaluation:

- FF-Replan (Yoon *et al.*, 2007): an online probabilistic planner that converts the probabilistic domain definitions into a deterministic domain using all-outcomes determinization. It then uses FF to compute a solution plan. During the execution, each time the planner leads to a state that is not expected, the planner searches for a new plan from the current unexpected state to the goal. FF-Replan won the 2004 International Probabilistic Planning Competition.
- FFH (Yoon *et al.*, 2008): an FF-Replan planner that handles unexpected states by an online anticipatory strategy based on hindsight optimization.
- FFH+(Yoon et al., 2010): an improved FFH with helpful actions that allow the planner to reduce its computational cost. These methods detect potentially useful actions and reuse relevant plans.
- FPG (Buffet & Aberdeen, 2006): considers the planning problem as an optimization problem and solves it using stochastic gradient ascent through the OLpomdp Algorithm (Baxter & Bartlett, 1999). The search is performed in policy space. This planner won the 2006 International Probabilistic Planning Competition.
- RFF (Teichteil-Königsbuch *et al.*, 2010): determinizes the PPDDL problem the same as FF-Replan. It then computes a policy by generating successive execution paths leading to the goal from the initial states by using FF. The generated policy has a low probability of failing during execution. This probability is computed by using a Monte Carlo simulation. This planner won the 2008 International Probabilistic Planning Competition.

We compare these with four variants of the PIPSS planner:

- PIPSS$_C$: a PIPSS planner where the propagation of cost information through the plan graph does not consider interaction estimates. During execution, the planner does not perform any further action when an unexpected state occurs.
- PIPSS$_I$: a PIPSS planner where the propagation of cost information through the plan graph considers interaction estimates. Like PIPSS$_C$, during execution, the planner does not perform any further action when an unexpected state occurs.
- PIPSS$_{CR}$: a PIPSS planner where the propagation of cost information through the plan graph does not consider interaction estimates. To deal with unexpected states at execution time, the planner does runtime replanning.
- PIPSS$_{IR}$: a PIPSS planner where the propagation of cost information through the plan graph considers interaction estimates. Like PIPSS$_{CR}$, to deal with unexpected states at execution time, the planner does runtime replanning.

This section is divided into four sections. The first one shows the tests performed with the different versions of our PIPSS planner. We explain the benefits of considering interaction estimates and runtime replanning, and the CPU time for the four variants of PIPSS is introduced. The other three sections correspond to the set of domains used for experimental purposes. For these three, we show a table that represents the number of successful rounds. The results have been compared with those shown by Yoon *et al.* (2010). We could not compare computation time because some of the planners are not available.

6.4. *Test of the PIPSS planner versions*

As mentioned earlier, the tests are performed in a simulation environment. Because PIPSS$_C$ and PIPSS$_I$ do not perform replanning when unexpected outcomes happen, we have evaluated the following strategies to consider a failure during runtime:

```
(define (domain buy-milk)
 ...
 (:action drive
  :parameters (?from - loc ?to - loc)
  :precondition (and (at ?from) (not (flattire)))
  :effect (and (at ?to) (not (at ?from))
            (probabilistic 2/5 (flattire))))

 (:action get-cash
  :precondition (at store)
  :effect (have-cash))

 (:action buy-milk
  :precondition (and (at store) (have-cash))
  :effect (have-milk))
-----------------------------------------------------
(define (problem milk)
 (:domain buy-milk)
 (:objects home store - loc)
 (:init (and (at home) (not (flattire))))
 (:goal (have-milk)))
```

Figure 7: *A probabilistic domain and problem expressed in PPDDL.*

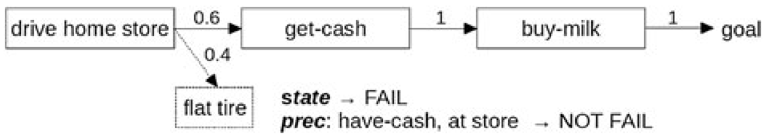


Figure 8: *Example that shows the difference between the state and prec simulation approaches.*

- *state* approach: fails when the state is not as expected. That is, the state given by the simulator is different from the state that the planner predicted during the search process.
- *prec* approach: fails when the state does not satisfy regressed preconditions. That is, the state given by the simulator does not satisfy the necessary preconditions to continue the remaining plan.

To illustrate these approaches, consider the problem shown in Figure 7. It consists of a person that drives to the store to buy milk. To accomplish this, the person needs to have cash. A tire can go flat during the trip. If this event happens, however, the drive reaches the store anyway. Figure 8 shows the plan generated by the planner. When the simulation starts, the planner sends the action `drive (home, store)` to the simulator. The performance of this action may give either a flat tire or not. If the next outcome causes a flat tire and the simulation is working using *state*, then the process will stop and consider this as a failure because the state was not what the planner was expecting. However, using *prec*, we can check that even though the car gets a flat tire, the preconditions for the remaining plan, `have-cash` and `at store`, are covered because the flat tire fact does not affect the remaining plan. Thus, the plan can continue without causing a failure in the simulation.

For all the planners, 500 trials per problem were performed with a total time limit of 500 minutes for the 500 trials. There are 15 problems for each domain. So, the maximum number of successful rounds for each domain is $15 \cdot 500 = 7500$. However, Triangle-tireworld domain on IPPC-08 only has 10 problems so that the total rounds in this case is $10 \cdot 500 = 5000$.

Tables 1–3 show the number of successful rounds for the different versions of PIPSS in each domain.

Table 1: *Total number of successful rounds on the IPPC-06 using different versions of the PIPSS system*

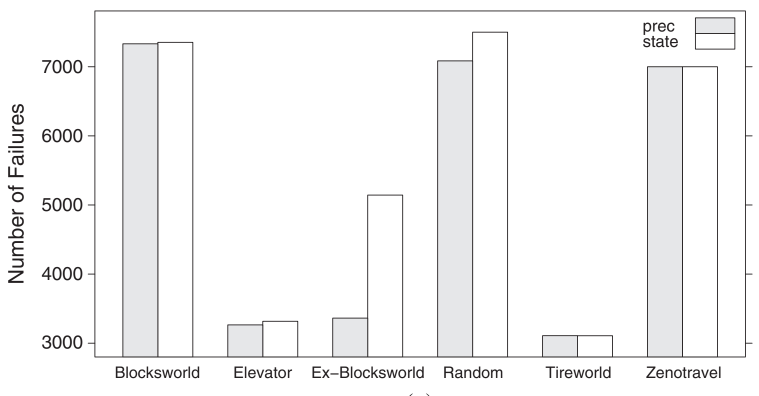
IPPC-06						
Domains	PIPSS$_{Cs}$	PIPSS$_{Cp}$	PIPSS$_{Is}$	PIPSS$_{Ip}$	PIPSS$_{CR}$	PIPSS$_{IR}$
Blocksworld	140	133	147	168	2762	2951
Exploding-Blocksworld	2386	4139	2357	4137	4038	4018
Elevators	4349	4414	4183	4235	6455	6524
Random	0	592	0	415	1508	1511
Tireworld	4393	4372	4392	4391	5548	5614
Zenotravel	500	500	500	500	2285	2207
Total	11768	14150	11579	13846	22596	22825

Table 2: *Total number of successful rounds on the IPPC-08 using different versions of the PIPSS system*

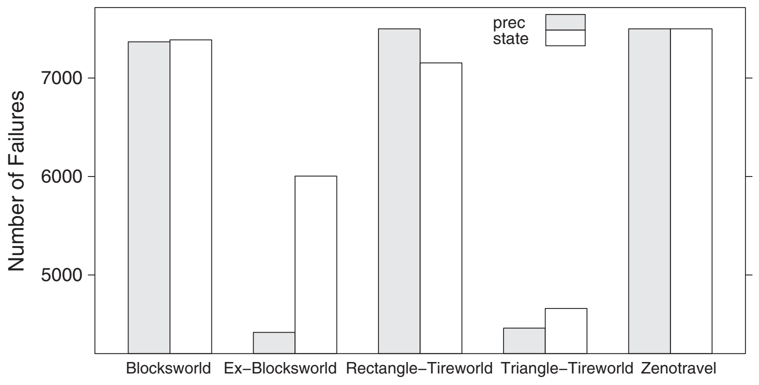
IPPC-08						
Domains	PIPSS$_{Cs}$	PIPSS$_{Cp}$	PIPSS$_{Is}$	PIPSS$_{Ip}$	PIPSS$_{CR}$	PIPSS$_{IR}$
Blocksworld	134	106	112	132	2358	2493
Exploding-Blocksworld	1461	3021	1469	3083	3001	3025
Triangle-tireworld	335	327	363	340	338	343
Rectangle-tireworld	288	0	346	0	314	320
Zenotravel	0	0	0	0	1730	2000
Total	2218	3454	2290	3769	7741	8181

Table 3: *Total number of successful rounds on probabilistically interesting domains using different versions of the PIPSS system*

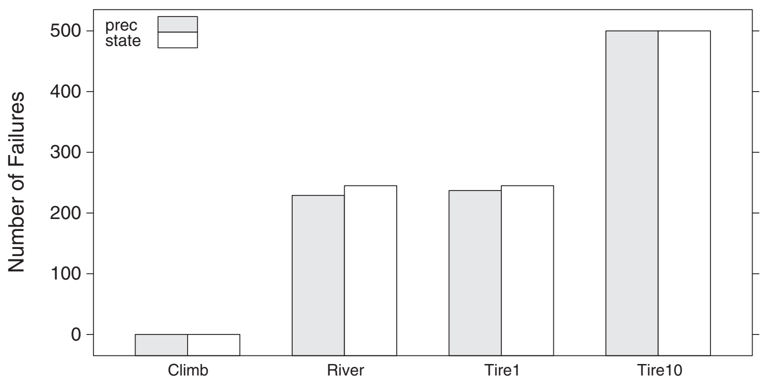
Probabilistically interesting domains						
Domains	PIPSS$_{Cs}$	PIPSS$_{Cp}$	PIPSS$_{Is}$	PIPSS$_{Ip}$	PIPSS$_{CR}$	PIPSS$_{IR}$
Climb	500	500	500	500	500	500
River	257	248	255	271	254	263
Tire1	266	253	255	263	251	266
Tire10	0	0	0	0	0	0
Total	1023	1001	1010	1034	1005	1029



(a)



(b)



(c)

Figure 9: *Comparison between state and prec simulation approaches in a number of failed rounds on (a) IPPC-06, (b) IPPC-08, and (c) probabilistically interesting domains.*

Results show us that the versions with a higher number of successful rounds are those that consider runtime replanning: PIPSS$_{CR}$ and PIPSS$_{IR}$. According to these tables, we can also confirm that failing during execution when the current state does not satisfy regressed preconditions (*prec*) works better than failing if the given state is not in the seed plan (*state*). Figure 9 shows the number of unsuccessful rounds that PIPSS$_{Is}$ and PIPSS$_{Ip}$ found while performing the previous test. We can observe that using *prec*, the number of failures is usually lower than performing *state* in the domains Blocksworld, Exploding-Blocksworld, Elevator, Random and Triangle-tireworld, where the plan's actions have fewer interdependences. This means that, if the system is doing runtime replanning, the replanning times will be lower. Because of this, we have performed PIPSS$_{CR}$ and PIPSS$_{IR}$ versions by applying the approach in which the planner performs replanning only when the state does not satisfy regressed preconditions. By using this strategy, we can deduce in advance if the remaining plan could be completely performed so that it would avoid replanning calls.

When we compare PIPSS$_{CR}$ and PIPSS$_{IR}$, PIPSS$_{IR}$ holds a slight advantage overall. In general, in those domains where there are several mutex or non-mutex ways from which the goal can be reached from the initial state, interaction estimates should find a plan with higher probability. In Figure 10, the generated seed plans for the two approaches have nearly identical probability except in one problem where the plan generated by PIPSS$_{IR}$ has significantly higher probability. For instance, in Figure 10(a), we can see that the probability generated by PIPSS$_{IR}$ in problem six is higher than that found by PIPSS$_{CR}$. In the case of the Rectangle-tireworld domain (Figure 10(c)), PIPSS$_{IR}$ generates a plan with higher probability than PIPSS$_{CR}$ for problem three. In the case of the Elevator domain (Figure 10 (b)), PIPSS$_{IR}$ generates a plan with higher probability than PIPSS$_{CR}$ in problem 10.

For interaction alone, without any replanning, the number of problems with higher probability solutions is not large, but for the combination of interaction estimates and runtime replanning, the system's behaviour improves considerably. In this case, although the advantage of PIPSS$_I$ over PIPSS$_C$ is small for any individual planning episode, it compounds over the course of many replanning cycles, resulting in a more substantial advantage for PIPSS$_{IR}$ over PIPSS$_{CR}$.

Figure 11 shows the distribution of the time required to find the seed plan for PIPSS$_C$ and PIPSS$_I$ over all of the problems. The curves are very similar with a distinct large peak and two other smaller but noticeable peaks. Most problems are solved in less than 5 s; however, the second

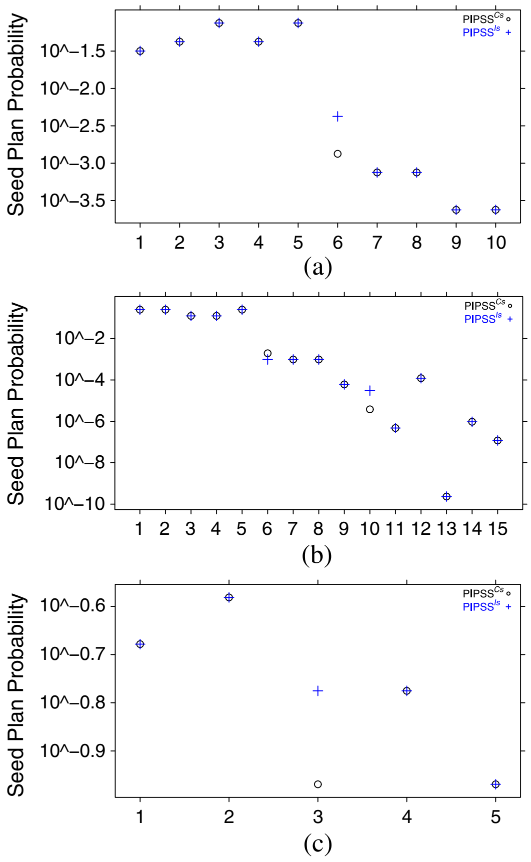


Figure 10: *Comparison between the seed plan probability obtained by PIPSS$_C$ and PIPSS$_I$ in different domains: (a) Blocksworld IPPC-06, (b) Elevator IPPC-06, and (c) Rectangle-tireworld IPPC-08 domains.*

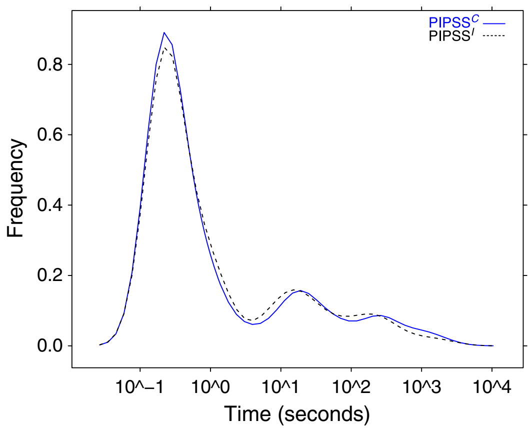


Figure 11: *Distribution of time for generating the seed plans.*

peak shows that there are cases in which the planner requires 6–100 s. These cases correspond to the most difficult problems in Blocksworld-08, a single case in Blocksworld IPPC-06, and a few cases in Exploding-Blocksworld IPPC-06, Elevator and Rectangle-tireworld. The third peak corresponds with the hardest problems in Exploding-Blocksworld IPCC-08 and Elevator, and with a single case in Random, Rectangle-tireworld and Zenotravel IPPC-08. Problems whose time exceeds 1800 s are instances in which the planner does not find a solution within the allotted search time.

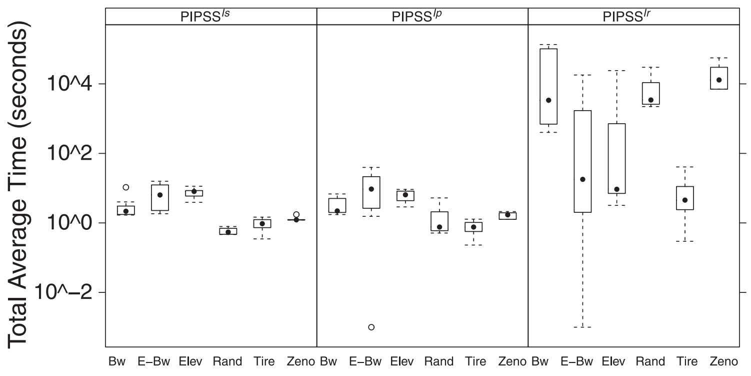
Figure 12 shows the total average time for simulation per domain while performing the previous test with the variants PIPSS$_{Is}$, PIPSS$_{Ip}$, and PIPSS$_{IR}$. In general, we can observe that using the approach *prec*, the time is slightly higher than applying *state* in some of the domains. However, as we have shown in Tables 1–3, by performing *prec*, the planner has a higher number of successful rounds. According to the graph, we also see a significant increase in the total time for PIPSS$_{IR}$. This is largely due to the runtime replanning that this variant performs. However, the difference in the number of successful rounds between the variants that do not use runtime replanning and the ones that do is extremely large. So we think that the additional time is a worthwhile penalty.

6.5. The 2006 IPPC

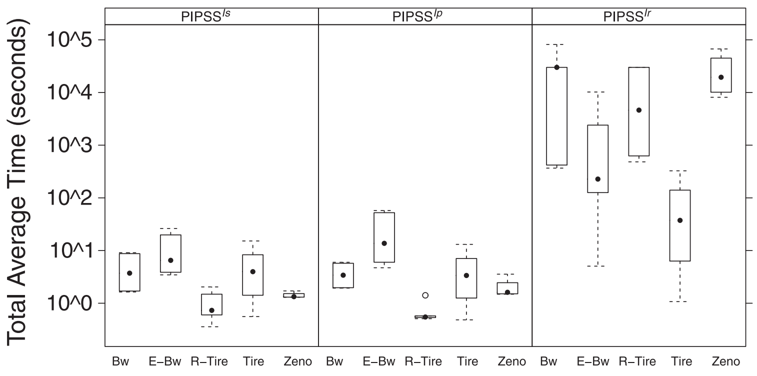
For the IPPC-06, we are concerned with the FOP domains, which were described at the beginning of the section.

For all the planners, 30 trials per problem were performed (as in the competition) with a total time limit of 30 minutes for the 30 trials. There are 15 problems for each domain. So, the maximum number of successful rounds for each domain is $15 \cdot 30 = 450$.

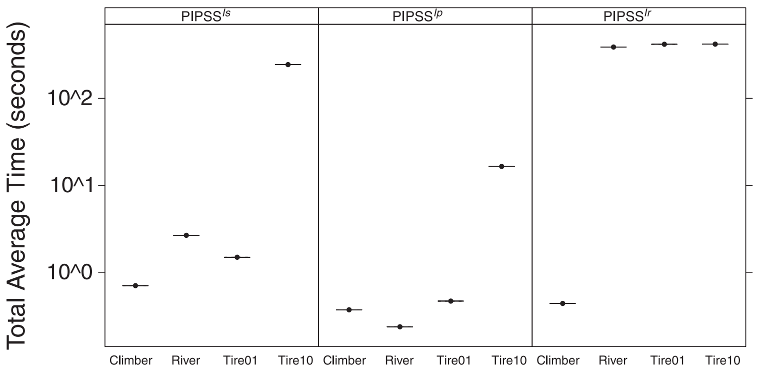
Table 4 shows the number of successful rounds for FFH, FFH+, FPG, PIPSS$_{CR}$, and PIPSS$_{IR}$ planners in each domain. PIPSS$_{IR}$ obtains good results in three of the five domains. The highest success rates are obtained in the domains Exploding-Blocksworld, Elevators, and Tireworld that have dead-end states even though PIPSS cannot recover from dead-end states. In fact, PIPSS$_{IR}$ is the planner that achieves the highest rate in the Elevator domain. This is evidence that we are generating relatively low-cost (high-probability) plans. However, in domains such as Blocksworld or Zenotravel, PIPSS performs poorly. Although we obtain good results in problems with dead-end states, we were somewhat surprised that in problems with no dead ends, we obtain worse results, because these domains are well suited to the replanning technique. Table 5 shows information about the behaviour of our planner in the Blocksworld domain. For the smaller problems in this domain, PIPSS works perfectly by achieving all the rounds successfully. However, as the complexity of the problem increases, the performance of PIPSS is worse. We note that the number of replanning calls during a simulation is related to the length of the seed plan generated. The greater the length of the plan is, the greater is the chance of failure. According to the data shown in Table 5, for the most difficult problems, the replanning technique requires a significant amount of time to have a new plan. One reason for the large amount of time consumed for PIPSS is that it is being overwhelmed by the number of determinized ground actions in the domain. To corroborate this, Figure 13 shows the number of instantiated actions for



(a)



(b)



(c)

Figure 12: *Comparison between the total time required by PIPSS$_{Is}$, PIPSS$_{Ip}$ and PIPSS$_{IR}$ on (a) IPPC-06, (b) IPPC-08, and (c) probabilistically interesting domains.*

Table 4: *Total number of successful rounds on the IPPC-06*

	Planners				
Domains	FFH	FFH+	FPG	PIPSS$_{CR}$	PIPSS$_{IR}$
Blocksworld	256	**335**	283	160	162
Exploding-Blocksworld	205	**265**	193	233	239
Elevators	214	292	342	379	**396**
Random	301	**357**	292	83	62
Tireworld	343	**364**	337	342	360
Zenotravel	0	**310**	121	115	123
Total	1319	**1923**	1568	1312	1342

each problem in each domain. The non-dead-end domains such as Blocksworld, Random or Zenotravel are those with the highest number of instantiated actions, which corresponded to the domains with low rates. In the Blocksworld domain, PIPSS only finds solutions for the first 10 problems, because for the rest of the problems the planner does not find a solution within the given time limit. We found additional evidence for this by performing the test for problems six and nine with the time limit increased by a factor of 10. This resulted in 100% successful rounds. We performed the same test for problems seven and eight obtaining 10 and 14 successful rounds. The total amount of time needed to achieve all the successful rounds for these two problems was 824 and 549 minutes, respectively. We have found the same behaviour in Zenotravel and Random domains; however, in these cases, the planner only finds a solution for the first five problems within the given time limit.

All of these suggest that the performance of PIPSS could be improved by paying greater attention to minimizing the number of ground actions and improving the efficiency of plan graph construction and calculations.

Table 5: *Execution of the Information Blocksworld IPPC-06 domain*

Blocksworld					
Problem	Length seed plan	Time seed plan (s)	Replanning calls	Average replanning time (s)	Successful rounds
1	16	0.198	188	0.282	30
2	14	0.197	137	0.254	30
3	10	0.194	110	0.270	30
4	14	0.24	117	0.281	30
5	10	0.222	129	0.256	30
6	24	1.344	82	22.18	7
7	32	10.45	11	176.7	0
8	24	0.776	16	113.8	0
9	20	0.398	78	23.84	5
10	28	1.476	6	365.2	0

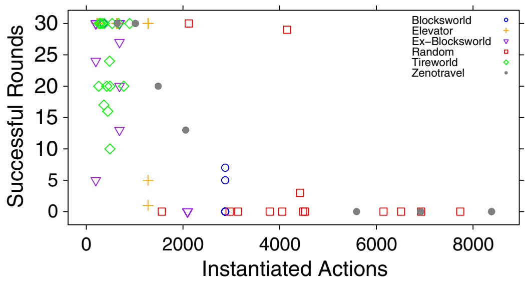


Figure 13: *Number of instantiated actions in each domain of IPPC-06.*

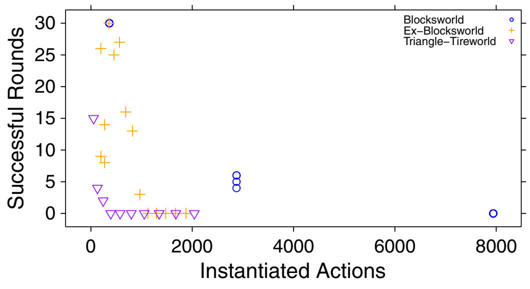


Figure 14: *Number of instantiated actions in each domain of IPPC-08.*

Table 6: *Total number of successful rounds on the IPPC-08*

Planners					
Domains	FFH	FFH+	RFF	PIPSS$_{CR}$	PIPSS$_{IR}$
Blocksworld	185	270	**364**	101	141
Exploding-Blocksworld	131	**214**	58	174	171
2-Tireworld	**420**	**420**	382	17	21
Total	736	**904**	804	292	333

6.6. The 2008 IPPC

For the IPPC-08, we are again concerned with the FOP domains, which were described at the beginning of the section. For all the planners, 30 trials per problem were performed with a total time limit of 30 minutes for the 30 trials. There are 15 problems for each domain. So, the maximum number of successful rounds for each domain is $15 \cdot 30 = 450$. The results in Table 6 show that again PIPSS has a low success rate in those domains without dead-end states such as Blocksworld. As is shown in Table 7, the behaviour of our planner in this Blocksworld domain is the same as the previous section. The time consumed by replanning in the most difficult problems exhausted the given time. We can also confirm by Figure 14 that the number of determinized ground actions for this domain is large, and this fact influences the runtime replanning. In the Exploding-Blocksworld domain, PIPSS obtains better results than the winning RFF planner, although it has a low number of successful rounds compared

Table 7: *Execution of the Information Blocksworld IPPC-08 domain*

Blocksworld					
Problem	Length seed plan	Time seed plan (s)	Replanning calls	Average replanning time (s)	Successful rounds
1	12	0.213	115	0.245	30
2	12	0.208	123	0.247	30
3	12	0.216	151	0.234	30
4	12	0.198	0	0	30
5	26	19.66	65	28.45	6
6	26	19.15	55	33.93	5
7	26	19.33	51	35.35	6
8	26	22.05	61	29.70	4
9	28	1.307	4	442.9	0
10	28	1.276	4	437.5	0

Table 8: *Total number of successful rounds on probabilistically interesting benchmarks*

	Planners					
Domains	FF-Replan	FFH	FFH+	FPG	PIPSS$_{CR}$	PIPSS$_{IR}$
Climb	19	**30**	**30**	**30**	**30**	**30**
River	20	20	20	20	16	**23**
Tire1	15	**30**	**30**	**30**	16	21
Tire10	0	6	**30**	0	0	0
Total	54	86	**110**	80	62	74

with FFH+. In the 2-Tireworld domain, PIPSS does poorly because this domain leads to a high number of dead-end states that it cannot recover from. PIPSS is unable to solve problems in SysAdmin-SLP and Search and Rescue domains from the 2008 IPPC, because it cannot parse PDDL Probabilistically `imply` and `exists` expressions.

6.7. *Probabilistically interesting domains*

For all the planners, 30 trials per problem were performed with a total time limit of 30 minutes for the 30 trials. There is one problem for each domain, so the maximum number of successful rounds for each domain is 30. Runtime replanning does not result in any improvement in these domains because of the way that they are designed. However, we can see in Table 8 that both PIPSS$_{CR}$ and PIPSS$_{IR}$ obtain good results in almost all problems. In the Climb domain, both variants of PIPSS achieve the maximum number of successful rounds. Compared with FF-Replan, which only performs replanning as PIPSS does, we are performing better because of the higher probability seed plans. In the River domain, PIPSS$_{IR}$ achieves the highest rate. However, for the Tireworld domains as the number of tires increases, PIPSS does not work as well because of the inability to deal with dead-end states in advance. Nevertheless, the result achieved by PIPSS in the other domains demonstrates the power of the heuristic.

7. Related work

Probabilistic planning is an active research field. Several different techniques are currently being used to solve probabilistic planning problems. We classify these techniques in four main groups:

1. Solving Markov decision processes. These approaches involve the construction of a policy using some form of value or policy iteration (Putterman, 1994; Boutilier *et al*., 1999) or heuristically guided forward state space search such as LAO* (Hansen & Zilberstein, 2001) or LRTDP (Bonet & Geffner, 2003). One feature that almost all approaches share is that all potential action outcomes in the plan are taken into account. That is, the plan includes a contingency branch for every possible outcome that may occur, given the actions in the plan. These approaches generate robust plans but can be computationally expensive.
2. Using planning graph models. These approaches use planning graphs to compute estimates of probability that propositions can be achieved and actions can be performed. This information can be used to guide a probabilistic planner towards the most likely plan for achieving the goals using a goal regression search (Blum & Langford, 1999; Little & Thiébaux, 2006). Bryce, Kambhampati, and Smith (2006) also use a plan graph to compute estimates of probability. However, they perform an AO* algorithm in seeking a plan solution.
3. Translation into deterministic planning problems. These approaches translate the given planning problem into a classical planning problem that is then solved by a classical planner (Palacios & Geffner, 2009). Other approaches combine the translation-based technique with replanning (Albore *et al*., 2009), or replanning and sampling (Brafman & Shani, 2012), obtaining better results. It is important to note that these planners only deal with uncertainty in the initial state, not with uncertainty in the actions' outcomes. Also, for the most part, these planners only deal with disjunctive uncertainty not probabilistic uncertainty.
4. Determinization and replanning. These approaches, such as FF-Replan (Yoon *et al*., 2007) and RFF (Teichteil-Königsbuch *et al*., 2010), transform the given probabilistic planning problem into a deterministic planning problem and use heuristic functions based on relaxed plans to guide a deterministic planner in the search for a deterministic plan. As a result, these planners generate a solution that is executed until the observed state differs from what is expected according to the next step in the solution. If this happens, then replanning is performed to seek another plan solution. Jiménez, Coles, and Smith (2006) also determinize the planning problem but turn the probability information into costs, and search for an optimal plan using a numeric deterministic planner that minimizes cost.

Our approach involves the conversion of the probability information into cost, the use of relaxed plans to guide a deterministic planner, and runtime replanning. The main difference with our planner is the novel planning graph heuristic function, which considers probability and interaction information to guide the planning search towards high probability (low-cost) plans.

8. Conclusions and future work

In this paper, we have presented a novel plan graph heuristic for probabilistic planning with full observability. The heuristic function we developed makes use of the probability information given in the domain definition to build a probabilistic plan graph estimator. This heuristic estimator is used to guide the search towards high probability plans. The resulting plans are used in a system that handles unexpected outcomes by runtime replanning.

According to the results of the 2006 IPPC and 2008 IPPC, the combination of deterministic planning and replanning seems to work well. Although our planner does not deal with dead-end states in advance, the results dealing with probabilistic planning problems have high success rates in several cases. This is evidence that we are generating relatively high probability seed plans. Unfortunately, the system does not obtain good results under some large domains with no dead-end states. For this reason, one avenue of future work involves optimizing plan graph generation and calculation. Another important enhancement would be to extend the capabilities of our planner to be able to handle some PDDL expressions that are currently not supported.

Of course, runtime replanning is not enough to deal with cases where there are dead-end states. That is why the main thrust of our future work involves analyzing the generated seed plans to find potential points of failure that can be identified as recoverable or unrecoverable. Recoverable failures will be left in the plan and will be repaired through replanning at execution time. For each unrecoverable failure, we attempt to improve the chances of recovery, by adding precautionary steps such as taking along extra supplies or tools that would allow recovery if the failure occurs.

Acknowledgements

This work has been partially funded by the Junta de Comunidades de Castilla-La Mancha project PEII11-0079-8929, the Universities Space Research Association (USRA), and the NASA Ames Research Center. The second author is funded by the Spanish CDTI project COLSUVH.

References

ALBORE, A., H. PALACIOS and H. GEFFNER (2009) A translation-based approach to contingent planning, in *International Joint Conference on Artificial Intelligence (IJCAI'09)*, Pasadena, CA, USA.

BAXTER, J. and P.L. BARTLETT (1999) Direct gradient-based reinforcement learning: I. gradient estimation algorithms. *Technical report*, Australian National University.

BLUM, A. and M. FURST (1997) Fast planning through planning graph analysis, *Artificial Intelligence*, **90**, 281—300.

BLUM, A. and J. LANGFORD (1999) Probabilistic Planning in the Graphplan Framework, in *European Conference on Planning (ECP'99)*, Durham, United Kingdom.

BONET, B. and H. GEFFNER (2003) Labeled RTDP: Improving the Convergence of real-time Dynamic Programming, in *International Conference of Automated Planning and Scheduling (ICAPS'03)*, Trento, Italy.

BONET, B. and R. GIVEN (2006) International Probabilistic Planning Competition, http://www.ldc.usb.ve/~bonet/ipc5

BOUTILIER, C., T. DEAN and S. HANKS (1999) Decision theoretic planning: structural assumptions and computation leverage, *Journal of Artificial Intelligence Research*, **11**, 1—94.

BRAFMAN, R.I. and G. SHANI (2012) A multi-path compilation approach to contingent planning, in *AAAI Conference on Artificial Intelligence (AAAI'12)*, Toronto, Canada.

BRYCE, D. and D.E. SMITH (2006) Using Interaction to Compute Better Probability Estimates in Plan Graphs, in *ICAPS'06 Workshop on Planning Under Uncertainty and Execution Control for Autonomous Systems*, The English Lake District, Cumbria, United Kingdom.

BRYCE, D., S. KAMBHAMPATI and D.E. SMITH (2006) Planning graph heuristics for belief space search, *Journal of Artificial Intelligence Research*, **26**, 35—99.

BUFFET, O. and D. ABERDEEN (2006) The Factored Policy Gradient Planner, in *International Planning Competition (IPC'06)*, The English Lake District, Cumbria, United Kingdom.

BUFFET, O. and D. BRYCE (2006) International Probabilistic Planning Competition, http://ippc-2008.loria.fr/wiki/index.php/Main_Page.

CESTA, A., G. CORTELLESSA, A. ODDI, N. POLICELLA and A. SUSI (2001) A Constraint-Based Architecture for Flexible Support to Activity Scheduling, in *Italian Association for Artificial Intelligence (AI*IA'01)*, Pisa, Italy.

E-MARTÍN Y., M.D. R-MORENO and D.E. SMITH (2011) Using a Plan Graph with Interaction Estimates for Probabilistic Planning, in *International Conference of the British Computer Society's Specialist Group on Artificial Intelligence (SGAI'11)*, Cambridge, United Kingdom.

GHALLAB et al. (2004) Automated Planning: Theory and Practice. Morgan Kaufmann Publishers: San Francisco, CA, Putterman: New York, NY.

HANSEN, E.A. and S. ZILBERSTEIN (2001) Lao*: a heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, **129**, 35—62.

HOFFMANN, J. and B. NEBEL (2001) The FF planning system: fast plan generation through heuristic search, *Journal of Artificial Intelligence Research*, **14**, 253—302.

JIMÉNEZ, S., A. COLES and A. SMITH (2006) Planning in Probabilistic Domains using a Deterministic Numeric Planner, in *Workshop of the UK Planning and Scheduling Special Interest Group (PLANSIG'06)*, Nottingham, United Kingdom.

LITTLE, I. and S. THIÉBAUX (2006) Concurrent Probabilistic Planning in the Graphplan Framework, in *ICAPS'06 Workshop on Planning Under Uncertainty and Execution Control for Autonomous Systems*, The English Lake District, United Kingdom.

LITTLE, I. and S. THIÉBAUX (2007) Probabilistic Planning vs Replanning, in *ICAPS'07 Workshop on Planning Competitions*, Providence, Rhode Island, USA.

PALACIOS, H. and H. GEFFNER (2009) Compiling uncertainty away in conformant planning problems with bounded width, *Journal of Artificial Intelligence Research*, **35**, 623—675.

PLAZA, J., M.D. R-MORENO, B. CASTAÑO, M. CARBAJO and A. MORENO (2008) PIPSS: parallel integrated planning and scheduling system, in *Workshop of the UK Planning and Scheduling Special Interest Group* (*PLANSIG'08*).

PUTTERMAN, M.L. (1994) Markov Decision Processes: Discrete Stochastic Dynamic Programming. John Wiley and Sons: New York, NY.

R-MORENO, M.D., D. CAMACHO and A. MORENO (2005) HPP: a heuristic progressive planner, in *Workshop of the UK Planning and Scheduling Special Interest Group* (*PLANSIG'05*).

TEICHTEIL-KÖNIGSBUCH, F., U. KUTER and G. INFANTES (2010) Incremental Plan Aggregation for Generating Policies in MDPs, in *Autonomous Agents and Multiagent Systems (AAMAS'10)*, Toronto, Canada.

YOON, S., A. FERN and B. GIVAN (2007) FF-Replan: A Baseline for Probabilistic Planning, in *International Conference on Automated Planning and Scheduling (ICAPS'07)*, Providence, Rodhe Island, USA.

YOON, S., A. FERN, R. GIVAN and S. KAMBHAMPATI (2008) Probabilistic Planning via Determinization in Hindsight, in *AAAI Conference on Artificial Intelligence (AAAI'08)*, Chicago, Illinois, USA.

YOON, S., W. RUML, J. BENTON and M.B. DO (2010) Probabilistic Planning via Determinization in Hindsight, in *International Conference on Automated Planning and Scheduling (ICAPS'10)*, Toronto, Ontario, Canada.

YOUNES, H.L.S., M.L. LITTMAN, D. WEISSMAN and J. ASMUTH (2005) The first probabilistic track of the international planning competition, *Journal of Artificial Intelligence Research*, **24**, 841—887.

The authors

Yolanda E-Martín

Yolanda E-Martín is a PhD candidate at Universidad de Alcalá (Spain) and an intern at NASA Ames Research Center. Her current research focuses on incremental contingency planning for temporal and numeric planning problems that involve actions with uncertain outcomes, uncertain durations, and uncertain resource usage.

María D. R-Moreno

María D. R-Moreno received an MS degree in physics from the Universidad Complutense de Madrid (Spain) and her PhD in computer science from Universidad de Alcalá (Spain). Her research interest focuses on planning and scheduling, robotics, and evolutionary computation. She spent one year at NASA Ames Research Center as a postdoc and nine weeks at ESA's European Space Research and Technology Centre (ESTEC) as a research visitor. She is currently an associate professor in the Computer Engineering Department at the Universidad de Alcalá. Her current collaborations include projects with ESA and research with NASA Jet Propulsion Laboratory and NASA Ames Research Center.

David E. Smith

David E. Smith is a senior researcher in the Intelligent Systems Division at NASA Ames Research Center. He received his PhD in computer science from Stanford University in 1985. He served as a research associate at Stanford, as a scientist at the Rockwell Palo Alto Science Center, and as a visiting scholar at the University of Washington before joining NASA in 1997. His research interests include temporal planning, planning under uncertainty, oversubscription planning, and the interplay between motion planning and task planning. He has recently been focusing on applications of planning and scheduling to aviation safety and air traffic management. He has served as an associate editor for the *Journal of Artificial Intelligence Research* and as guest editor for two special issues of the journal. He was recognized as a fellow of the Association for the Advancement of Artificial Intelligence (AAAI) in 2005.