In Thirty-first SGAI International Conference on Artificial Intelligence (SGAI-11)

Using a Plan Graph with Interaction Estimates for Probabilistic Planning

Yolanda E-Martín and María D. R-Moreno and David E. Smith

Abstract Many planning and scheduling applications require the ability to deal with uncertainty. Often this uncertainty can be characterized in terms of probability distributions on the initial conditions and on the outcomes of actions. These distributions can be used to guide a planner towards the most likely plan for achieving the goals. This work is focused on developing domain-independent heuristics for probabilistic planning based on this information. The approach is to first search for a low cost deterministic plan using a classical planner. A novel plan graph cost heuristic is used to guide the search towards high probability plans. The resulting plans can be used in a system that handles unexpected outcomes by runtime replanning. The plans can also be incrementally augmented with contingency branches for the most critical action outcomes.

1 Introduction

The success of plan graph heuristics in classical planners like FF [11] or HSP [3], has influenced research on heuristic estimators to deal with probabilistic planning problems. These kind of problems, represented in PPDDL [7], are characterized by full observability and non-deterministic effects of actions that are expressed by probability distribution.

A few probabilistic planners such as FF-Replan [15] or RFF [6] determinize the given probabilistic problem into a classical planning problem, and use heuris-

Yolanda E-Martín
Departamento de Automática, Universidad de Alcalá, e-mail: yolanda@aut.uah.es

María D. R-Moreno
Departamento de Automática, Universidad de Alcalá, e-mail: mdolores@aut.uah.es

David E. Smith
Intelligent Systems Division, NASA Ames Research Center, e-mail: david.smith@nasa.gov

tic functions based on relaxed plans to guide a classical planner in the search for a deterministic plan. However, other probabilistic planners use plan graphs to compute estimates of probability that propositions can be achieved and actions can be performed [1, 8]. This information can be used to guide the probabilistic planner towards the most likely plan for achieving the goals.

The main motivation for this work is to find high probability deterministic seed plans. These plans can be used in a system that handles unexpected outcomes by runtime replanning. The plans can also be incrementally augmented with contingency branches for critical action outcomes. To find high probability seed plans, we use a relaxed-plan heuristic to guide a forward state space planner. Construction of relaxed plans is guided by probability estimates propagated through a plan graph. These probability estimates are more accurate than typical, as they make use of the notion of interaction introduced by Bryce & Smith in [5].

This approach has been implemented in the Parallel Integrated Planning and Scheduling System (PIPSS) [18]. PIPSS is the union of a heuristic search planner and a scheduling system. This paper starts by describing the translation technique from probabilistic planning domains into a deterministic domains. Then, we describe our plan graph cost estimator and the relaxed plan extraction procedure that guides the search. We follow with an empirical study of the techniques within PIPSS and compare with some other probabilistic planners. Finally, future work is discussed.

2 Conversion from PPDDL to PDDL

To convert from PPDDL to PDDL we follow the approach of Jimenez, Coles & Smith [14]. 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 in to a cost equal to the negative logarithm of the probability. This cost will be used to compute the probability of each proposition and action.

More precisely, if A is a probabilistic action with outcomes $O_1,...,O_i$ with probabilities $P_1,...,P_i$ 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 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 A_i is given the cost $C_i = -Ln(P_i)$. Figure 1 shows an example of the conversion strategy . Figure 1(a) shows a probabilistic action that has two effects leading to the two deterministic actions shown in Figures 1(b) and 1(c).

```
(:action pick-up
 :parameters (?b1 ?b2 - block)
 :precondition (and (not(= ?b1 ?b2)) (emptyhand) (clear ?b1) (on ?b1 ?b2))
 :effect (probabilistic
   3/4 (and (holding ?b1) (clear ?b2) (not(emptyhand)) (not(clear ?b1))
            (not(on ?b1 ?b2)))
   1/4 (and (clear ?b2) (on-table ?b1) (not(on ?b1 ?b2)))))
```

(a) Probabilistic action in PPDDL

```
(:action pick-up-ALIAS-0
 :parameters (?b1 ?b2 - block)
 :precondition (and (not(= ?b1 ?b2)) (emptyhand) (clear ?b1) (on ?b1 ?b2))
 :effect (and (holding ?b1) (clear ?b2) (not(emptyhand))
              (not(clear ?b1)) (not(on ?b1 ?b2)) (increase (cost) 0.28)))
```

(b) Deterministic action I

```
(:action pick-up-ALIAS-1
 :parameters (?b1 ?b2 - block)
 :precondition (and (not(= ?b1 ?b2)) (emptyhand) (clear ?b1) (on ?b1 ?b2))
 :effect (and (clear ?b2) (on-table ?b1) (not(on ?b1 ?b2))
              (increase (cost) 1.38)))
```

(c) Deterministic action II

Fig. 1 Example of determinization of a probabilistic action.

3 Plan Graph Cost Heuristic

In this section we describe the plan graph heuristic, which guides the planner towards the lowest cost (highest probability) plan.

Given the deterministic actions created by the technique described in the previous section, we build a plan graph and propagate 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 will be the cost that its preconditions can be achieved. The cost of achieving a proposition in the next level of the plan graph will be 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* (L), which captures the degree of dependence (positive or negative) between pairs of propositions and actions in the plan graph [5].

3.1 Interaction

Interaction, is a value that determines 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, L, between two propositions or two actions (p and q) is defined as:

$$L(p,q) = Cost(p \wedge q) - (Cost(p) + Cost(q)) \tag{1}$$

It has the following features:

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

That is, L provides information about the degree of interference or synergy between pairs of propositions and pairs of actions in a plan graph. When $L(p,q) < 0$ (synergy) this means that the cost of establishing both p and q is less than the sum of the cost 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 $L(p,q)$ is bounded below by $-min[cost(p), cost(q)]$. Similarly, $0 < L(p,q) < \infty$ 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, $L = \infty$, 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 pair of propositions and all pair of actions at each level. Hence, this terms is taken into account in the cost propagation. In this way, we can establish a better estimation of the cost that two propositions or two action perform at the same time.

3.2 Plan Graph Estimator

This subsection describes the method to create the cost plan graph. The process consists of building a plan graph using a modified GraphPlan [2] algorithm in which the mutex calculation is replaced with interaction calculation.

The cost and interaction computation begins at level zero of the plan graph and sequentially proceeds to higher levels. For level zero we assume 1) the cost of each proposition at this level is $-Ln(Pr)$, where Pr is the probability of the proposition in the current state and 2) the interaction between pair of propositions is 0, that is, the propositions are independent. Neither of these assumptions are essential, but we

adopt them here for simplicity. With these assumptions, we start the propagation by computing the cost of the actions at level zero.

In the following subsections, we give the details of how to do this beginning at the initial proposition layer and working forward to actions, and finally to the next proposition layer.

Computing Action Cost and Interaction

Lets 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:

$$Cost(a) = \sum_{\substack{(x_i,x_j)\in prec_a \\ j>i}} \Big[Cost(x_i) + L(x_i,x_j) \Big] \tag{2}$$

As an example, consider one level of the plan graph shown in Figure 2, where we have three propositions q, r and s with costs .22, .43 and .69 respectively, and interaction values $L(q,r) = -.2$, $L(q,s) = .3$ and $L(r,s) = .6$ at level i. There are also two actions P and W which have outcomes with costs .26 and .3 respectively (these costs are the negative logarithm of the probabilities for those outcomes). The numbers above the propositions and actions are the costs associated with each one (those with * are the costs that will be computed in subsequent sections).

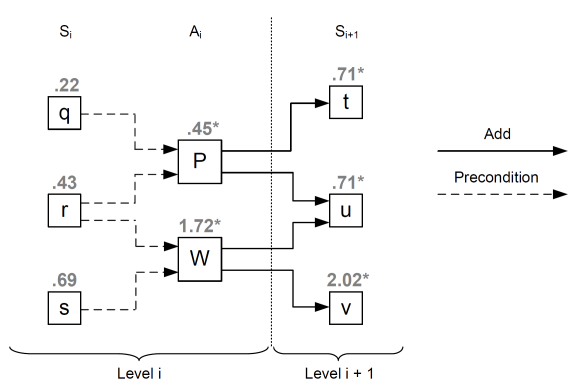

Fig. 2 A Plan Graph with Costs and Interaction Calculation and Propagation

For the example shown in Figure 2, the cost for actions P and W would be:

$$Cost(P) = Cost(q) + Cost(r) + L(q,r) = .22 + .43 - .2 = .45$$

$$Cost(W) = Cost(r) + Cost(s) + L(r,s) = .43 + .69 + .6 = 1.72$$

The next step is to compute the interaction between actions. If the actions are mutex by inconsistent effect, or effects clobbering preconditions, then the cost 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:

$$L(a,b) = \begin{cases} \infty \text{ if } a \text{ and } b \text{ are mutex by inconsistent effects or effects clobbering preconditions} \\ Cost(a \wedge b) - Cost(a) - Cost(b) \text{ otherwise} \end{cases} \tag{3}$$

Where in the second case the interaction can be simplified as follows:

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

For the example in Figure 2, the interaction between actions P and W would be:

$$L(P,W) = L(q,s) - Cost(r) = .3 - .43 = -.13$$

The fact that $L(P,W) = -.13$ means that there is some degree of synergy between the actions P and W. This synergy comes from the fact that the two actions have a common precondition, r. However, this synergy is tempered due to the interference between the precondition q of P and the precondition s of W.

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:

$$Cost(x) = \min_{a \in Ach(x)} [Cost(a) + Cost(x|a)] \tag{4}$$

Where $Cost(x|a)$ is the cost of the outcome x for the action a.

In our example, the cost of the proposition u of the graph is:

$$\begin{aligned} Cost(u) &= \min[Cost(P) + Cost(u|P), Cost(W) + Cost(u|W)] \\ &= \min[.45 + .26, 1.72 + .3] \\ &= \min[.71, 2.02] = .71 \end{aligned}$$

Finally, we consider the interaction between a pair of propositions x and y. In order 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:

$$L(x,y) = \min_{\substack{a \in Ach(x) \\ b \in Ach(y)}} [Cost(a) + Cost(b) + L(a,b) + Cost(x|a) + Cost(y|b)] - Cost(x) - Cost(y) \tag{5}$$

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

$$\begin{aligned} I(t,u) &= \min[Cost(P) + Cost(u|P), Cost(P) + Cost(W) + L(P,W) + Cost(t|P) + Cost(u|W)] \\ &\quad - Cost(t) - Cost(u) \\ &= \min[.45 + .26, .45 + 1.72 - .13 + .26 + .3] - .71 - .71 \\ &= .71 - .71 - .71 = -.71 \end{aligned}$$

In this case, it is less costly to establish the propositions t and u through action P than P and W. This makes sense because executing a single action always has lower cost than performing two.

Taking the above 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 appears in the graph, and the interaction value of every pair of propositions and every pair of operators are then used to compute the heuristic estimation that guides the planner towards low cost plans.

3.3 Heuristic Computation

As mentioned above, the plan graph and cost estimates are used to help guide a forward state-space planning search. Thus, initially the system builds a cost plan graph. For each state, the plan graph is updated and a relaxed plan is created to estimate the cost of achieving the goals from that state. The relaxed plan regression 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.
- 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:

$$\underset{a \in Ach(g)}{\operatorname{argmin}} \left[Cost(a) + Cost(g|a) + \sum_{b \in O} L(a,b) \right] \tag{6}$$

Figure 3 shows the algorithm used in the relaxed plan regression phase.

Function COSTESTIMATE (G,l)

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

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

Fig. 3 The relaxed plan regression pseudo-algorithm.

To illustrate, consider the relaxed plan graph shown in Figure 4, where every operator and action has associated a cost value. Suppose that the set of goals is composed of the propositions r and p with cost .53 and .67 respectively, we start analyzing goal p and later r. In this way, we first deal with the highest cost goal. The proposition p is achieved by actions C and D and, we choose C because it has the lowest cost. Goal r is achieved by operator B and *noop-r*. In order to know which is the best choice, the cost of achieving r must be computed for both of these operators assuming operator C. Considering the action cost values shown in Figure 4, and the interactions and additive costs: $L(B,C) = 0$, $L(\text{noop-r},C) = 0$, $Cost(r|\text{noop-r}) = 0$ and $Cost(r|B) = 0.25$, the chosen operator would be:

$$\underset{a \in Ach(g)}{\text{argmin}} \quad [Cost(\text{noop-r}) + L(\text{noop-r},C) + Cost(r|\text{noop-r}), Cost(B) + L(B,C) + Cost(r|B)]$$

$$\underset{a \in Ach(g)}{\text{argmin}} \quad [0+0+0, 0+0+0.25] = 0$$

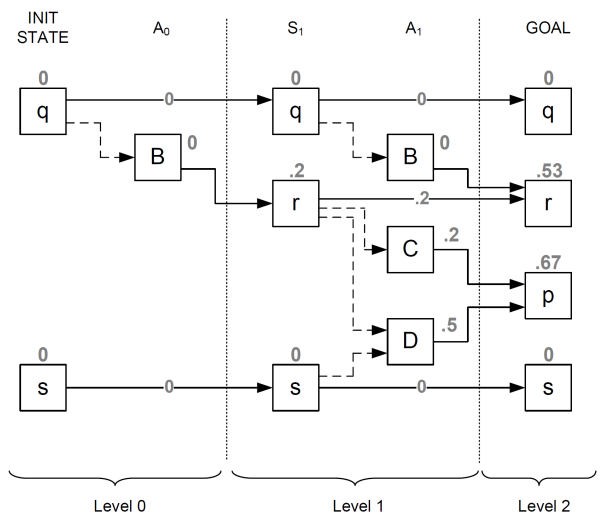

Fig. 4 A Relaxed Plan Graph

In this case the selected operator is *noop-r* because it would have the lowest cost.

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. That would be the sum of the cost of every action selected in the search plan process. In this way, we are considering the interaction information.

Supposing that the plan solution selected is composed of operators B and C, the heuristic would be 0 + .2 = .2. In this way, the planner would guide the search process towards the low cost path thus achieving the low cost plan.

4 Experimental Results

In this section, we describe the experiments that demonstrate that the plan graph cost heuristic guides the search to high probability solutions for probabilistic planning problems. We have conducted an experimental evaluation on IPPC-06 [4] and IPPC-08 [12] fully-observable-probabilistic planning track (FOP) as well as the "probabilistically interesting" domains introduced by Little and Thiebaux [9]. For all the planners 30 trials per problem were performed with a total time limit of 30 minutes for the 30 trials. The test consists of running the planner and using the resulting plan in the MDP Simulator [7]. Unexpected outcomes are handled by runtime replanning. The planner and simulator communicate by exchanging messages: the simulator sends a message to the planner with the current state; the executive sends a message with the next action of the plan. When the planner receives an unexpected state it treats this by replanning.

Four planners have been used for the experimental evaluation:

- FFH [16]: an FF-Replan planner that converts the probabilistic domain definitions into a deterministic domain using all-outcomes determinization. It then uses FF to compute a solution plan. To handle unexpected states it generates contingency branches on the outcomes that are expected to be in the plan. The original FF-Replan [15] won the 2004 International Probabilistic Planning Competition.
- FFH+ [17]: an improved FFH with helpful methods that allow the planner to reduce its computational cost. These methods detect potentially useful actions and reuse relevant plans.
- FPG [13]: considers the planning problem as an optimization problem, and solves it using stochastic gradient ascent through the OLpomdp Algorithm [10]. This planner won the 2006 International Planning Competition.
- RFF [6]: determinizes the PPDDL problem into a classical planning problem and then produces solution plans that are treated using sequential Monte-Carlo simulations to assess the probability of replanning during execution. This planner won the 2008 International Planning Competition.

This section is divided into three subsections, each one corresponding to the set of domains used for experimental purposes. For each subsection, we show a table that represents the number of successful rounds. The results have been compared with those shown in [17]. We cannot compare computational time because some of the planners are not available.

4.1 The 2006 IPPC Domains

The Second International Probabilistic Planning Competition consisted of two tracks, one for conformant planning characterized by non-observability and non-deterministic effects (NOND track), and the other for probabilistic planning with

fully observable domains and probabilistic action outcomes (FOP track). We are concerned with the FOP track, which contains the following domains:

- Blocks World: this domain is similar to the classical BlocksWorld with additional actions. A gripper can hold a block or a tower of them or be empty. When trying to perform an action, there is a chance of dropping a block on the table.
- Exploding Blocks World: this is 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 in 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.
- Tire World: in this domain a car has to move between two locations. When the car drives a part of the route, there is the possibility of 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.

Results

There are 15 problems for each domain. So, the maximum number of successful rounds for each domain is $15 \cdot 30 = 450$. Table 1 shows the number of successful rounds for FFH, FFH+, FPG and PIPSS$_I$ planners in each domain. PIPSS$_I$ gets good results in three of the five domains. Concretely, the higher successful rates are obtained in those domains like Exploding BlocksWorld or TireWorld that have dead-end states despite PIPSS$_I$ does not deal with dead-end states yet. This is evidence that we are generating relatively low cost (high probability) plans. However, in classical domains like BlocksWorld or Zeno, PIPSS$_I$ perfoms poorly. Although we get good results in problems with dead end states, we are surprised that in classical problems we obtain worse results. Then, we need to analyze how the calculation of the interactions affects this type of problems.

Table 1 Successful Rounds on the IPPC-06

	PLANNERS			
DOMAINS	FFH	FFH+	FPG	PIPSS$_I$
Blocks World	256	335	283	113
Exploding Blocks World	205	265	193	180
Elevators	214	292	342	342
Tire World	343	364	337	351
Zeno Travel	0	310	121	50
TOTAL	1018	1566	1276	1003

4.2 The 2008 IPPC Domains

The Uncertainty Part of the 6th International Planning Competition had three different tracks: fully observable probabilistic (FOP), non-observable non-deterministic (NOND) and fully observable non-deterministic (FOND). Again we have used the FOP track, which contains the following domains:

- Blocks World: Similar to IPPC-06 Blocks World Domain.
- Exploding Blocks World: Similar to IPPC-06 Exploding-BlocksWorld Domain.
- 2-Tire World: Similar to IPPC-06 Tire World Domain but with slight differences in the definition to permit short but dangerous paths.
- Search and Rescue: in this domain there is a helicopter on a rescue mission. To achieve its mission, it has to explore several areas to find one that is landable and close to the human to rescue.
- SysAdmin-SLP: this domain consists of a system administrator that has to manage a network of servers that fail with a higher probability when a neighbouring server is down. The objective is to restart servers. However, there is the possibility that the network is completely down.

Results

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 2 shown that again PIPSS$_I$ has a low successful rate in those domains with no dead end states like BlocksWorld. In the Exploding BlocksWorld domain, PIPSS$_I$ obtains better results than the winner planner of the IPPC, the RFF planner. However, it gets low number of successful rounds compare to the other two planners. In the 2-TireWorld domain, it gets a poorly rate. That is because, this domain leads on a high number of dead end states that are not supported by PIPSS$_I$. In the SysAdmin-SLP and Search and Rescue domains PIPSS$_I$ was unable to solve any problem because there are some PDDL expressions (i.e. exists, imply) that PIPSS$_I$ cannot yet handle.

Table 2 Successful Rounds on the IPPC-08

	PLANNERS			
DOMAINS	FFH	FFH+	RFF	PIPSS$_I$
Blocks World	185	270	364	120
Exploding Blocks World	131	214	58	85
2-Tire World	420	420	382	42
Search and Rescue	450	450	0	0
SysAdmin-SLP	0	0	117	0
TOTAL	1186	1354	921	247

4.3 Probabilistically Interesting Domains

Little and Thiebaux have created a number of very simple problems that explore the issue of probabilistic planning vs replanning. These problems lead to dead-ends. These may show the true behavior of the planner because several reasons as mentioned in [9]. First, the presence of dead-end states where the goal is unreachable. Second, the degree to which the probability of reaching a dead-end state can be reduced through the choice of actions. Third, the number of distinct trajectories from which the goal can be reached from the initial state. Finally, the presence of mutual exclusion of choices that exclude other courses of action later.

Results

There is one problem for each domain so, the maximum number of successful rounds for each domain is 30. Table 3 shows that in this test, PIPSS$_I$ completes all the rounds for the *Climb* domain and gets the highest successful rate for the *River* domain. For the *Tire* domain, when the number of tires is low, PIPSS$_I$ gets a good rate, but when this number increases it does not solve any round.

Table 3 Successful Rounds on Probabilistically Interesting Benchmarks

	PLANNERS				
DOMAINS	FF-Replan	FFH	FFH+	FPG	PIPSS$_I$
Climb	19	30	30	30	30
River	20	20	20	20	23
Tire1	15	30	30	30	21
Tire10	0	6	30	0	0
TOTAL	54	86	110	80	74

5 Conclusions and Future Work

In this paper we have presented a novel plan graph heuristic. This heuristic estimator is used to guide the search towards high probability plans. The resulting plans will be 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 be the best. Although our planner does not deal with dead-end outcomes, 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. However, replanning is not enough to dealing with those cases in which the dead-end states cannot achieve a solution plan. For this reason, our future work involves analyzing the generated seed plans to find potential points of failure which will 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, an attempt will be made 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 is funded by the Junta de Comunidades de Castilla-La Mancha project PEII11-0079-8929. We want to thank Bonifacio Castaño for his help during the development of this work.

References

1. A. Blum and J. Langford. Probabilistic Planning in the Graphplan Framework. In Proceedings of The 5th European Conference on Planning. Durham, UK, 1999.
2. A. Blum and M. Furst. Fast Planning Through Planning Graph Analysis. Artificial Intelligence, vol. 90, pp: 281-300, 1997.
3. B. Bonet and H. Geffner. Planning as Heuristic Search. Artificial Intelligence, vol. 129, pp: 5-33, 2001.
4. B. Bonet and R. Givan. International Probabilistic Planning Competition.`http://www.ldc.usb.ve/~bonet/ipc5`, 2006.
5. D. Bryce and D. E. Smith. Using Interaction to Compute Better Probability Estimates in Plan Graphs. In Proceedings of The ICAPS-06 Workshop on Planning Under Uncertainty and Execution Control for Autonomous Systems. The English Lake District, Cumbria, UK, 2006.
6. F. Teichteil-Königsbuch and U. Kuter and G. Infantes. Incremental Plan Aggregation for Generating Policies in MDPs. In Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems. Toronto, Canada, 2010.
7. H. L. S. Younes, M. L. Littman, D. Weissman and J. Asmuth. The First Probabilistic Track of the International Planning Competition. Journal of Artificial Intelligence Research, 24, pp: 841-887, 2005.
8. I. Little and S. Thiébaux. Concurrent Probabilistic Planning in the Graphplan Framework. In Proceedings of ICAPS-06 Workshop on Planning Under Uncertainty and Execution Control for Autonomous Systems. The English Lake District, Cumbria, UK, 2006.
9. I. Little and S. Thiébaux. Probabilistic Planning vs Replanning. In Proceedings of ICAPS-07 Workshop on Planning Competitions. Providence, Rhode Island, USA, 2007.
10. J. Baxter and P. L. Bartlett. Direct Gradient-Based Reinforcement Learning: I. Gradient Estimation Algorithms. Technical Report. Australian National University, 1999.
11. J. Hoffmann and B. Nebel. The FF Planning System: Fast Plan Generation Through Heuristic Search. Journal of Artificial Intelligence Research, 14, pp: 253-302, 2001.
12. O. Buffet and D. Bryce. International Probabilistic Planning Competition. `http://ippc-2008.loria.fr/wiki/index.php/Main_Page`, 2008.
13. O. Buffet and D. Aberdeen. The Factored Policy Gradient Planner. In Proceedings of the 5th International Planning Competition. The English Lake District, Cumbria, UK, 2006.
14. S. Jimenez, A. Coles and A. Smith. Planning in Probabilistic Domains using a Deterministic Numeric Planner. In Proceedings of the 25th Workshop of the UK Planning and Scheduling Special Interest Group. Nottingham, UK, 2006.
15. S. Yoon, A. Fern and R. Givan. FF-Replan: A Baseline for Probabilistic Planning. In Proceedings of the 17th International Conference on Automated Planning and Scheduling. Providence, Rhode Island, USA, 2007.
16. S. Yoon, A. Fern, R. Givan and S. Kambhampati. Probabilistic Planning via Determinization in Hindsight. In Proceedings of the 23rd AAAI Conference on Artificial Intelligence. Chicago, Illinois, USA, 2008.
17. S. Yoon, W. Ruml, J. Benton and M. B. Do. Improving Determinization in Hindsight for Online Probabilistic Planning. In Proceedings of the 20th International Conference on Automated Planning and Scheduling. Toronto, Canada, 2010.
18. Y. E-Martín, M. D. R-Moreno and B. Castaño. PIPSS*: a System based on Temporal Estimates. In Proceedings of the 30th Annual International Conference of the British Computer Society's Specialist Group on Artificial Intelligence. Cambridge, UK, 2010.