Planning graph based heuristics for Partial Satisfaction Problems

Romeo Sanchez
Department of Computer Science and Engineering
Arizona State University, Tempe AZ, 85287-5406
rsanchez@asu.edu

Abstract

In many real world planning scenarios, agents often do not have enough resources to achieve all of their goals. Hence, this requires finding plans that satisfy only a subset of them. Solving such partial satisfaction planning (PSP) problems poses several challenges, including an increased emphasis on modelling and handling plan quality (in terms of action costs and goal utilities). Despite the ubiquity of such PSP problems, very little attention has been paid to them in the planning community. In this extended abstract, we focus on one of the more general PSP problems, termed PSP NET BENEFIT. After describing our problem, we present an initial approach for solving it based on *AltAlt* (Nguyen, Kambhampati, & Sanchez 2002), a regression planner enforced with reachability heuristics. We called our approach *AltAlt*^{ps} . We describe how cost information could be propagated in a planning graph data structure, and how this information could be used to select a subset of the top level goals upfront, and also to generate cost sensitive heuristics to guide the regression search. We also present an evaluation plan of our approach, as well as some possible improvements to *AltAlt*^{ps} .

Introduction

In classical planning problems, we assume a very strict success criterion, either the set of goals is completely satisfied or it is not possible to achieve such set. However, in real world planning scenarios, it is more preferable to achieve some of the goals than none of them, as here the success is measured in terms of an objective function being evaluated. These problems are also known as *Partial Satisfaction Problems* (PSP), since the goals can only be partially satisfied.

One of the most successful algorithms in the last few years for solving classical planning problems is *Graphplan* (Blum & Furst 1997). This algorithm can be seen as a disjunctive version of forward state space planners. The algorithm has two interleaved phases: a forward phase where a polynomial-time data structure called *"planning graph"* is incrementally extended, and a backward phase where that planning graph is searched for a solution.

Part of our past work has shown that the planning graph data structure from *Graphplan* is a rich source for extracting effective and admissible heuristics for controlling search under different planning frameworks. Specifically, we have developed a state-space planner called *AltAlt* (Nguyen, Kambhampati, & Sanchez 2002; Sanchez, Nguyen, & Kambhampati 2000), which uses such heuristics to drive a regression search engine. I have extended my initial work on *AltAlt* to generate parallel plans in state-space search using distance-based heuristics. This variant called *AltAlt*^{p}, further improves the quality of the parallel plans generated using an online plan compression algorithm (Sanchez & Kambhampati 2003). We have presented an extensive empirical evaluation for both approaches and their heuristics.

In this extended abstract, we will address the application of planning graph based heuristics to PSP problems, and we will introduce some ideas to greedily solve this kind of problems. Specifically, we will commit to problems that require action costs and goal utilities, in which our objective is to maximize the NET BENEFIT of the final plan. The net benefit is not more than the difference between the total utility of the goals achieved and the costs of the actions in the solution plan. We will introduce our problem as well as some ideas to handle cost in *AltAlt*^{ps} . The main extensions in *AltAlt*^{ps} involve techniques for propagating cost information on planning graphs (Do & Kambhampati 2002), and a novel approach for heuristically selecting a subset of goal conjuncts upfront. Once a subset of goal conjuncts is selected, it is solved by a regression search that has cost sensitive heuristics. We will also present an evaluation plan for our approach and future work.

Problem description and related work

Current planning frameworks do not address the issue of partial satisfaction of goals, because they strictly try to satisfy a conjunction of them. However, this all or nothing success criterion does not seem to be very realistic. There are problems where satisfying at least some of the goals is better than not satisfying any of them at all. The problem is of course to choose the goals that will be satisfied, subject to the optimization of an objective function. Choosing the right subset of goals is not an easy task. For example, in the context of a regression planner like *AltAlt* (Nguyen, Kambhampati, & Sanchez 2002), if we have n top level goals in our final

state, then we would need to consider 2^n goal subsets to find the optimal plan. Such a method would not be appropriate when we have too many goals, or when some of the goals have complex interactions and are not feasible together.

Instead, *AltAlt*^{ps} selects a promising subset of the top level goals upfront avoiding the exponential search, trying to maximize the quality of the solution in terms of its final NET BENEFIT. Here the benefit could be computed using the additive cost of the actions in the solution plan $\mathcal{P}$, and the total utility of the goals composing the subgoal set G'. Specifically:

Definition 1 $NetBenefit\ \beta(\mathcal{P}) : \sum_{g \in G'} utility(g) - \sum_{a \in \mathcal{P}} cost(a)$

which is the remaining utility after the execution costs of the actions have been discounted. Our approach then greedily selects a goal subset to approximate the optimal net benefit. Utilities of the top level goals and execution costs of the actions could be provided to the planner by the user, or could be generated randomly subjected to some constraints during the planning graph construction phase.[1]

Partial satisfaction planning problems could be also solved optimally using Integer Programming techniques. However, current IP planning approaches do not scale up well to bigger sized problems (Kautz & Walser 1999; T. Vossen & Nau 1999). PSP have been most widely studied in the scheduling community under the name of over-subscription scheduling. In general, an over-subscription scheduling problem is one where there are more tasks to be accomplished than there are time and resources available. Typical objectives are to maximize resource usage accommodating as many tasks as possible (Kramer & Smith 2003), or to minimize conflicts in the schedule. Over-subscription scheduling problems are resource driven, while PSP planning problems are more goal driven. Over-subscribed scheduling problems have been solved by iterative repair (Kramer & Smith 2003) methods, constructive approaches (Potter & Gasch 1998), greedy search approaches (Frank *et al.* 2001), and even genetic algorithms (Globus *et al.* 2003).

Planning graph cost propagation

Heuristic functions in our greedy approach need to estimate how costly would be to achieve the set of goals G from the initial state I. However, only the execution costs of the actions are given to the planner, and the propositions $p \in I$ are known to have $zero$ cost. Thus, we need to propagate the cost of every other single proposition in the planning graph to estimate the cost of the propositions in G in order to determine those who are relevant. Let $h_l(p)$ be the cost associated to an individual proposition p at level l of the planning graph. Initially, each proposition p would be assigned a cost of 0 if it is in the initial state, and ∞ otherwise. Thus, for each action a achieving p, the cost of p could be propagated using the iterative planning graph construction procedure of *AltAlt*^{ps} , as follows:

$$h_l(p) := \begin{cases} 0 & \text{if } p \in I \\ min\{h_l(p), cost(a) + \mathcal{C}_l(a)\} & \text{if } l > 0 \\ \infty & \text{otherwise} \end{cases} \quad (1)$$

Where, $cost(a)$ is the execution cost of the action given by the user or generated randomly by the planner, and $\mathcal{C}_l(a)$ is the aggregated cost of the action computed in terms of its preconditions as follows:

1. Max-Propagation: $\mathcal{C}_l(a) = max\{h_{l-1}(q) : q \in Prec(a)\}$, or
2. Sum-Propagation: $\mathcal{C}_l(a) = \sum\{h_{l-1}(q) : q \in Prec(a)\}$

The first alternative is conservative but admissible, while the second one assumes that goals are independent, and in consequence is not admissible but more informative (Bonet, Loerincs, & Geffner 1997).

Notice that we could use the notion of level of the planning graph as a unit of time to propagate the costs of propositions. As a consequence, cost propagation would terminate as soon as the planning graph gets built. We could let the user to decide the cutoff for building the graph. We then let the cost of a proposition be the final cost of that proposition in the final level of the planning graph. Similar cost propagation procedures have been developed in the context of metric temporal planning (Do & Kambhampati 2002).

Goal set selection procedure and search algorithm

The main idea of the goal set selection procedure in *AltAlt*^{ps} is to incrementally construct a new partial goal set G' from the top level goals G such that the goals considered for inclusion increase the final net benefit, using the goals utilities and costs of achievement. One way to do that is to select an initial subgoal g based on its net benefit, and incrementally add more subgoals to the goal set. If no subgoals can be chosen then we could easily conclude that there is no solution to our problem. The process is complicated by the fact that the net benefit offered by a goal g depends on what other goals have already been selected. Specifically, while the utility of a goal g remains constant, the expected cost of achieving it will depend upon the other selected goals (and the actions that will anyway be needed to support them). To estimate the "residual cost" of a goal g in the context of a set of already selected goals G', we compute a relaxed plan R_P (Hoffmann & Nebel 2001; Nguyen, Kambhampati, & Sanchez 2002) for supporting $G' + g$, which is biased to (re)use the actions in the relaxed plan R'_P for supporting G'. The residual cost of a goal g is the estimated cost introduced by the relaxed plan R_P if we were to consider $g \in G'$. In other words, we include a subgoal g in G' if its residual cost is lower than its utility, which in consequence would increase our final net benefit.

[1] Due to the lack of PSP benchmark problems.

Procedure *partialize*(G)
$g \leftarrow getBestBenefitialGoal(G)$;
if($g = NULL$)
return Failure;
$G' \leftarrow \{g\}$; $G \leftarrow G \setminus g$;
$R^*_P \leftarrow extractRelaxPlan(G', \emptyset)$
$B^*_{MAX} \leftarrow getUtil(G') - getCost(R^*_P)$;
$B_{MAX} \leftarrow B^*_{MAX}$
while($B_{MAX} > 0 \wedge G \neq \emptyset$)
for($g \in G \setminus G'$)
$G_P \leftarrow G' \cup g$;
$R_P \leftarrow ExtractRelaxPlan(G_P, R^*_P)$
$B_g \leftarrow getUtil(G_P) - getCost(R_P)$;
if($B_g > B^*_{MAX}$)
$g^* \leftarrow g$; $B^*_{MAX} \leftarrow B_g$; $R^*_g \leftarrow R_P$;
else
$B_{MAX} \leftarrow B_g - B^*_{MAX}$
end for
if($g^* \neq NULL$)
$G' \leftarrow G' \cup g^*$; $G \leftarrow G \setminus g^*$; $B_{MAX} \leftarrow B^*_{MAX}$;
end while
return G';
End *partialize*;

Figure 1: Goal set selection algorithm.

Figure 1 gives a description of our goal set selection algorithm. The first block of instructions before the loop initializes our goal subset G',[2] and finds an initial relaxed plan R^*_P for it using the procedure *extractRelaxPlan(G',$\emptyset$)*. Notice that two arguments are passed to the function. The first one is the current partial goal set from where the relaxed plan will be computed. The second parameter is the current relaxed plan that will be used as a guidance for computing the new relaxed plan. The idea is that we want to bias the computation of the new relaxed plan to re-use the actions in the relaxed plan from the previous iteration. Having found the initial subset G' and its relaxed plan R^*_P, we compute the current best net benefit B^*_{MAX} by subtracting the costs of the actions in the relaxed plan R^*_P from the total utility of the goals in G'. B^*_{MAX} will work as a threshold for our iterative procedure. In other words, we would continue adding subgoals $g \in G$ to G' only if the overall net benefit B^*_{MAX} increases. We consider one subgoal at a time, always computing the benefit added by the subgoal in terms of the cost of its relaxed plan R_P and goal utility B_g. We then pick the subgoal g that maximizes the net benefit, updating the necessary values for the next iteration. This iterative procedure stops as soon as the net benefit does not increase, or when there are no more subgoals to add, returning the new goal subset G'. Notice that the procedure above is greedy, so we are not immune from selecting a bad subset.

Having found our goal subset G', we proceed to search for a solution using a regression state-space engine. We adapt the search algorithm of *AltAlt* to take into account cost estimates rather than distance estimates. In *AltAlt*^{ps} , the evaluation function used to rank the nodes $f(S) = g(S)+w*h(S)$ represents a different set of estimates. In our cost-based framework, $g(S)$ is the cost of the current partial plan in terms of the additive cost of the actions composing it, and $h(S)$ is the estimated remaining cost given by the heuristics to make the plan a solution. Notice that, if at anytime the value of $g(S)$ is greater than the total utility of the goals chosen, then we know that we can kill the partial plan right away. We now proceed to describe the cost sensitive heuristics that will guide our search framework.

[2] $getBestBenefitialGoal(G)$ returns the subgoal with the best benefit, $utility(g) - h_l(g)$ tradeoff.

Cost-based planning graph heuristics

Once that we have selected our partial goal set, we need to search for a cost-based plan in order to maximize our final net benefit. In other words, we need to find not only the most feasible subset of goals but also the less expensive solution, such that our net benefit gets maximized. In this section, we introduce cost-based heuristics that will help us to guide the search towards such solutions.

From the cost propagation procedure described in the last section, we could easily derive the first heuristic for our cost-based planning framework, computed under the assumption that the propositions constituting a state are strictly independent. Such a heuristic is described as follows:

Sum Cost Heuristic 1 $h_{sumC}(S) = \sum_{p \in S} h_l(p)$

The h_{sumC} heuristic suffers from the same problems than the h_{sum} heuristic introduced by Bonet, Loerincs, & Geffner,1997. It will tend to overestimate the cost of a set of propositions. To make the heuristic admissible we could replace the sum function with the maximum of the individual costs of the propositions composing the state. This leads us to our next heuristic:

Max Cost Heuristic 2 $h_{maxC}(S) = \max_{p \in S} h_l(p)$

This heuristic also directly resembles the h_{max} heuristic from (Bonet, Loerincs, & Geffner 1997; Nguyen, Kambhampati, & Sanchez 2002), but in terms of cost. We could try to combine the *differential* power of h_{sumC} and h_{maxC} to get a more effective heuristic for a wider range of problems.

Combo Cost Heuristic 3 $h_{comboC}(S) = h_{sumC}(S) + h_{maxC}(S)$

We could improve further the solution quality of our heuristics if we start taking into account the positive interactions among subgoals while still ignoring the negative ones. We could adapt the idea of the relaxed plan heuristics from (Hoffmann & Nebel 2001; Nguyen & Kambhampati 2001; Nguyen, Kambhampati, & Sanchez 2002) into our framework. The general relaxed plan extraction process for *AltAlt*^{ps} works as follows: (i) start from the goal set G containing the top level goals, remove a goal

g from G and select a lowest cost action a_g (indicated by $h_l(g)$) to support g; (ii) regress G over action a_g, setting $G = G \cup Prec(a_g) \backslash Eff(a_g)$. The process continues recursively until each proposition $q \in G$ is also in the initial state I. This regression accounts for the positive interactions in the state G given that by subtracting the effects of a_g, any other proposition that is co-achieved when g is being supported is not counted in the cost computation. The relaxed plan procedure indirectly extracts a sequence of actions R_P (the actions a_g selected at each reduction), which would have achieved the set G from the initial state I if there were no negative interactions. The summation of the costs of the actions $a_g \in R_P$ can be used to estimate the cost to achieve all goals in G. The procedure described above is implemented through the *extractRelaxPlan(S,∅)* procedure introduced in Figure 1, obtaining the following heuristic:

RelaxCost Heuristic 4 $h_{relaxC}(S) = getCost(extractRelaxPlan(S, \emptyset))$

The h_{relaxC} heuristic does not take into account the negative interactions among subgoals. *AltAlt* provides a way to compute such interactions by considering the cost of ignoring them. We could adapt the ideas from (Nguyen, Kambhampati, & Sanchez 2002) to compute the *interaction degree* Δ among propositions in a particular state S, and include some cost for ignoring such interactions. Such interaction could be computed in the following way:

Definition 2 $\Delta(S) = lev(S) - max_{p \in S} lev(p)$

where $lev(S)$ is the first level in the planning graph in which all propositions in S appear with no subset of S marked mutex (Nguyen, Kambhampati, & Sanchez 2002).

Obviously, when there are negative interactions in the state S the value of $\Delta(S) > 0$. If that is the case we could add some cost that could reflect the penalty for ignoring the interactions. A conservative idea would be to add the h_{maxC} estimate of the state S to the h_{relaxC} heuristic when negative interactions are detected, implying that much cost is being left out by the considering only positive interactions. This could be formulated as follows:

Adjusted Relax Cost Heuristic 5

$$h_{adjRelC}(S) = \begin{cases} h_{relaxC}(S) if \Delta(S) = 0 \\ h_{relaxC}(S) + h_{maxC}(S) otherwise \end{cases} \quad (2)$$

Even though that this heuristic may improve the cost estimates during the planning phase, there is still one drawback in its computation. Notice that the penalizing cost factor would be based on the $h_{maxC}(S)$ estimate, which returns the maximum proposition cost from S. If the proposition p with the maximum cost does not belong to any mutex relation, then it should not be part of the penalizing cost factor, given that it is not part of the negative interactions of the state S. A way to improve the penalty estimate would be to consider every pair of interacting propositions in the state S and pick the pair with the biggest interaction degree. This idea has been explored in *AltAlt* too, and could be also used to find a better cost penalty factor. The binary interaction degree among two propositions has been defined in (Nguyen, Kambhampati, & Sanchez 2002), and it could be computed using:

$$\Delta_{max}(S) = max_{p,q \in S} \Delta(\{p, q\}) \quad (3)$$

This estimate then could be used to determine which propositions are more related to the negative interactions of the state, such that we could consider them to get the penalty cost factor. The following heuristic would take into account these issues.

Adjusted Relax2 Cost Heuristic 6

$$h_{adjRel2C}(S) = \begin{cases} h_{relaxC}(S,) if \Delta_{max}(S) = 0 \\ else\ h_{relaxC}(S) + h_{maxC}(\{g_1, g_2\}) \end{cases} \quad (4)$$

where $\Delta_{max}(S) = \Delta(\{g_1, g_2\})$.

Evaluation plan

It is hard to evaluate our approach because there are no benchmark PSP problems and alternative planning approaches. Therefore, we have decided to use existing STRIPS planning domains from the last International Planning Competition (Long & Fox 2002), and randomly generate costs and utilities for each particular planning problem. We have run an initial set of experiments on the DriverLog domain from AIPS 2002, in which we compare our approach *AltAlt*^{ps} with a variant of it that selects every single goal instead of finding a subset of them. The aim of this initial set of experiments is to demonstrate the impact of partialization in the quality of the solutions generated. We can see these initial results in Figure 2. Notice that in fact *AltAlt*^{ps} returns better quality solutions than its counterpart. The other approach is not able to find solutions for most of the problems (e.g. there is no benefit in conjunctive solutions).

Additional empirical evaluation is required to compare the effectiveness of our greedy algorithm with respect to optimal solutions generated by an IP or MDP based approach.

Conclusions and future work

In this paper, we introduced a generalization of the classical planning problem that allows partial satisfaction of goal conjuncts. We motivated the need for such partial satisfaction planning (PSP), and developed a heuristic (*AltAlt*^{ps}) planner for it. Our greedy approach is enriched with a family of cost-based heuristics extracted from a planning graph data structure. Our preliminary empirical results show that the heuristic approach is able to generate plans whose quality is much better than the ones generated by a normal planner that avoids partialization.

Our future work will extend our greedy framework[3] to more complex planning problems. Specifically, we plan to

[3] An initial version of our work has been submitted to the Workshop of Integrating Planning Into Scheduling at ICAPS-2004.

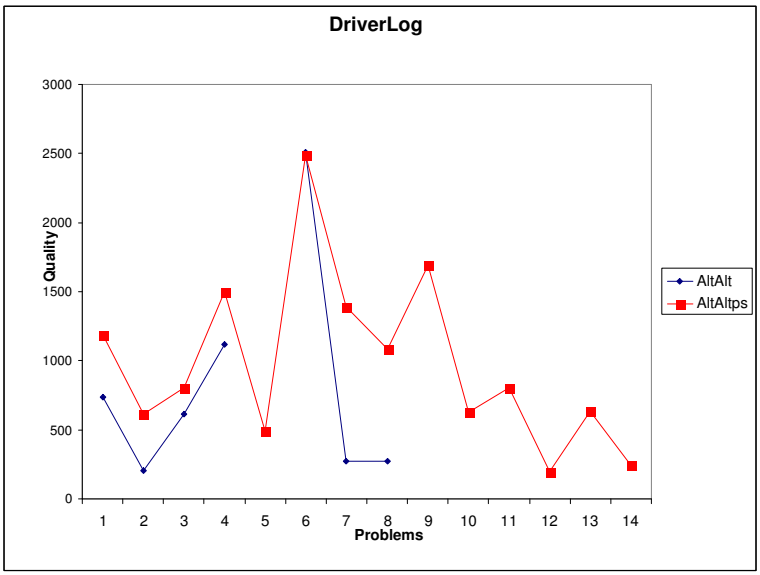

Figure 2: Impact of Partialization

consider problems in which goals may be interacting (e.g. they are mutexes), requiring a more aggressive account for interactions. We also plan to extend our approach to handle more sophisticated metrics (e.g. resource usage, multi-objective estimations). We also plan to evaluate more formally *AltAlt*^{ps} with respect to optimal approaches (e.g.IP based, etc). Finally, we would like to further investigate the techniques used to solve over-subscription scheduling problems to improve our partialization approach.

References

Blum, A., and Furst, M. 1997. Fast Planning Through Planning Graph Analysis. *Artificial Intelligence* 90:281–300.

Bonet, B.; Loerincs, G.; and Geffner, H. 1997. A Robust and Fast Action Selection Mechanism for Planning. In *Proceedings of AAAI-97*, 714–719. AAAI Press.

Do, M. B., and Kambhampati, S. 2002. Planning Graph-based Heuristics for Cost-sensitive Temporal Planning. In *Proceedings of AIPS-02*.

Frank, J.; Jonsson, A.; Morris, R.; and Smith, D. 2001. Planning and scheduling for fleets of earth observing satellites. In *Sixth International Symposium on Artificial Intelligence, Robotics, Automation and Space*.

Globus, A.; Crawford, J.; Lohn, J.; and Pryor, A. 2003. Scheduling earth observing sateliites with evolutionary algorithms. In *Proc. International Conference on Space Mission Challenges for Information Technology*.

Hoffmann, J., and Nebel, B. 2001. The FF Planning System: Fast Plan Generation Through Heuristic Search. *Journal of Artificial Intelligence Research* 14:253–302.

Kautz, H., and Walser, J. 1999. State-space planning by integer optimization. In *AAAI/IAAI*, 526–533.

Kramer, L. A., and Smith, S. 2003. Maximizing flexibility: A retraction heuristic for oversubscribed scheduling problems. In *IJCAI-03*.

Long, D., and Fox, M. 2002. Results of the AIPS 2002 Planning Competition. Toulouse, France.

Nguyen, X., and Kambhampati, S. 2001. Reviving partial order planning. In *Proc. of IJCAI*, 459–466.

Nguyen, X.; Kambhampati, S.; and Sanchez, R. 2002. Planning Graph as the Basis for Deriving Heuristics for Plan Synthesis by State Space and CSP Search. *Artificial Intelligence* 135(1-2):73–123.

Potter, W., and Gasch, J. 1998. A photo album of earth: Scheduling landsat 7 mission daily activities. In *Proc. International Symposium on Space Mission Operations and Ground Data Systems*.

Sanchez, R., and Kambhampati, S. 2003. Altaltp: Online Parallelization of Plans with Heuristic State Search. *Journal of Artificial Intelligence* 19:631–657.

Sanchez, R.; Nguyen, X.; and Kambhampati, S. 2000. AltAlt: Combining the Advantages of Graphplan and Heuristics State Search. In *Proceedings of KBCS-00*. Bombay, India.

T. Vossen, M. Ball, A. L., and Nau, D. 1999. On the use of integer programming models in AI planning. In *IJCAI*, 304–309.