Planning Graph Heuristics for Selecting Objectives in Over-subscription Planning Problems

Romeo Sanchez Nigenda and Subbarao Kambhampati*
Department of Computer Science and Engineering
Ira A. FULTON School of Engineering
Arizona State University, Tempe AZ, 85287-5406
{rsanchez, rao}@asu.edu

Abstract

Partial Satisfaction or Over-subscription Planning problems arise in many real world applications. Applications in which the planning agent does not have enough resources to accomplish all of their given goals, requiring plans that satisfy only a subset of them. Solving such partial satisfaction planning (PSP) problems poses several challenges, from new models for handling plan quality to efficient heuristics for selecting the most beneficial goals. In this paper, we extend planning graph-based reachability heuristics with mutex analysis to overcome complex goal interactions in PSP problems. We start by describing one of the most general PSP problems, the PSP NET BENEFIT problem, where actions have execution costs and goals have utilities. Then, we present *AltWlt*,[1] our heuristic approach augmented with a multiple goal set selection process and mutex analysis. Our empirical studies show that *AltWlt* is able to generate the most beneficial solutions, while incurring only a small fraction of the cost of other PSP approaches.

Introduction

Most planners handle goals of attainment, where the objective is to find a sequence of actions that transforms a given initial state I to some goal state G, where $G = g_1 \wedge g_2 \wedge ... \wedge g_n$ is a conjunctive list of goal fluents. Plan success for these planning problems is measured in terms of whether or not all the conjuncts in G are achieved. However, in many real world scenarios, the agent may only be able to partially satisfy G, because of subgoal interactions, or lacking of resources and time. Effective handling of partial satisfaction planning (PSP) problems poses several challenges, including the problem of designing efficient goal selection heuristics, and an added emphasis on the need to differentiate between feasible and optimal plans. In this paper, we focus on one of the most general PSP problems, called PSP NET BENEFIT. In this problem, each goal conjunct has a fixed utility attached to it, and each ground action has a fixed cost associated with it. The objective is to find a plan with the best "net benefit" (i.e., cumulative utility minus cumulative cost).

Despite the ubiquity of PSP problems, surprisingly little attention has been paid to the development of effective approaches for solving them. Some preliminary work by Smith (2004) proposed a planner for over-subscribed planning problems, in which an abstract planning problem (e.g. Orienteering graph) is built to select the subset of goals and orders to achieve them. Smith (2004) speculated that the heuristic distance estimates derived from a planning graph data structure are not particularly suitable for PSP problems given that they make the assumption that goals are independent, without solving the problem of interactions among them.

In this paper, we show that in fact although planning graph estimates for PSP problems are not very accurate in the presence of complex goal interactions, they can also be extended to overcome such problems. In particular, our approach named *AltWlt* that builds on the *AltAlt*^{ps} planner (van den Briel *et al.* 2004), involves a sophisticated multiple goal set selection process augmented with mutex analysis in order to solve complex PSP problems. The goal set selection process of *AltWlt* considers multiple combinations of goals and assigns penalty costs based on mutex analysis when interactions are found. Once a subset of goal conjuncts is selected, they are solved by a regression search planner with cost sensitive planning graph heuristics.

The rest of this paper is organized as follows. In the next section, we provide a brief description of the PSP NET BENEFIT problem. After that, we review *AltAlt*^{ps}, which forms the basis for our current approach, emphasizing its goals set selection algorithm and cost-sensitive reachability heuristics. Then, in the next part of the paper, we introduce our proposed approach, *AltWlt*, pointing out the limitations of *AltAlt*^{ps} with some clarifying examples, and the extensions to overcome them. We then present a set of complex PSP problems and an empirical study on them that compares the effectiveness of *AltWlt* with respect to its predecessor, and *Sapa*^{ps} (van den Briel *et al.* 2004), another planning-graph based heuristic PSP planner. We also show some illustrative data on an optimal MDP and IP approach to PSP NET BENEFIT to assess the quality of the solutions returned by *AltWlt*. Finally, we end with a discussion of the related

*We thank the ICAPS reviewers for many helpful comments. This research is supported in part by the NSF grant IIS-0308139.

[1] A Little of This and a Whole Lot of That

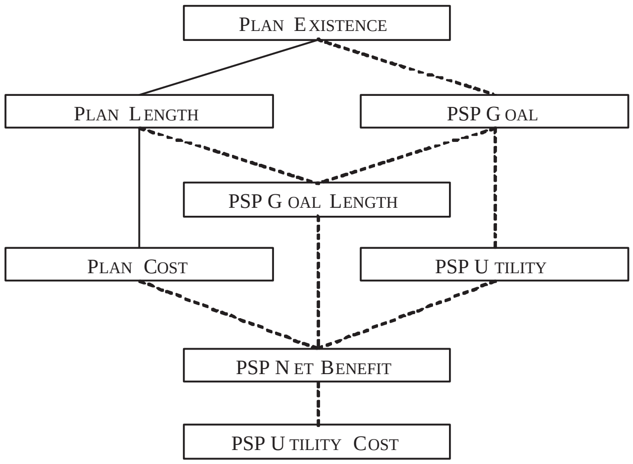


Figure 1: Hierarchical overview of several types of complete and partial satisfaction planning problems.

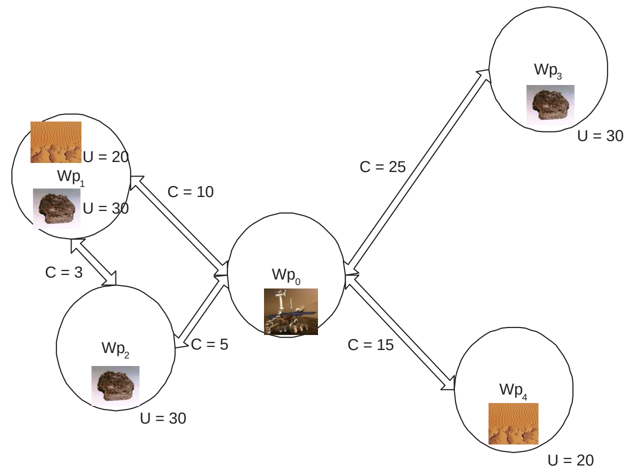


Figure 2: Rover domain problem

work and conclusions.

Preliminaries: PSP NET BENEFIT definition

The following notation will be used: F is a finite set of fluents and A is a finite set of actions, where each action consists of a list of preconditions and a list of add and delete effects. $I \subseteq F$ is the set of fluents describing the initial state and $G \subseteq F$ is the set of goal conjuncts. Hence we define a planning problem as a tuple $P = (F, A, I, G)$. Figure 1 gives a taxonomic overview of several types of complete and partial satisfaction planning problems. The problem of PSP NET BENEFIT is a combination of the problem of finding minimum cost plans (PLAN COST) and the problem of finding plans with maximum utility (PSP UTILITY), as a result is one of the more general PSP problems.[2] In the following, we formally define the problem of finding a plan with maximum net benefit:

Definition 1 (PSP Net Benefit:) *Given a planning problem $P = (F, A, I, G)$ and, for each action a "cost" $C_a \geq 0$ and, for each goal specification $f \in G$ a "utility" $U_f \geq 0$, and a positive number k. Is there a finite sequence of actions $\Delta = \langle a_1, ..., a_n \rangle$ that starting from I leads to a state S that has net benefit $\sum_{f \in (S \cap G)} U_f - \sum_{a \in \Delta} C_a \geq k$?*

Example 1: Figure 2 illustrates a small example from the rover domain (Long & Fox 2003) to motivate the need for partial satisfaction. In this problem, a rover that has landed on Mars needs to collect scientific data from some rock and soil samples. Some waypoints have been identified on the surface. Each waypoint has scientific samples. For example, $waypoint_3$ has a rock sample, while $waypoint_4$ has a soil sample. The rover needs to travel to a corresponding waypoint to collect its samples. Each travel action has a cost associated to it. For example, the cost of traveling from $waypoint_0$ to $waypoint_1$ is given by $C_{travel_{0,1}} = 10$. In addition to the $travel_{x,y}$ actions, we have two more actions $sample$ and $comm$ to collect and communicate the data respectively to the lander. To simplify our problem, these actions have uniform costs independent of the locations where they take place. These costs are specified by $C_{sample_{data,x}} = 5$ and $C_{comm_{data,x,y}} = 4$. Each sample (or subgoal) has a utility attached to it. We have a utility of $U_{rock_3} = 30$ for the rock sample at $waypoint_3$, and a utility $U_{soil_4} = 20$ for a soil sample at $waypoint_4$. The goal of the rover is to find a travel plan that achieves the best cost-utility tradeoff for collecting the samples. In this example, the best plan is $P = \{travel_{0,2}$, $sample_{rock_2,2}$, $comm_{rock_2,2,0}$, $travel_{2,1}$, $sample_{rock_1,1}$, $comm_{rock_1,1,0}$, $sample_{soil_1,1}$, $comm_{soil_1,1,0}\}$ which achieves the goals $rock_2, rock_1$ and $soil_1$, and ignores the rest of the samples at $waypoint_3$ and $waypoint_4$ giving the net benefit 45.

Background: *AltAlt*^{ps} Cost-based Heuristic Search and Goal Selection

In this section, we introduce *AltAlt*^{ps} because it forms the basis for *AltWlt*, the new proposed algorithm that handles complex goal interactions. *AltAlt*^{ps} is a heuristic regression planner that can be seen as a variant of *AltAlt* (Nguyen, Kambhampati, & Sanchez 2002) equipped with cost sensitive heuristics. An obvious, if naive, way of solving the PSP NET BENEFIT problem with such a planner is to consider all plans for the 2^n subsets of an *n-goal* problem, and see which of them will wind up leading to the plan with the highest net benefit. Since this is infeasible, *AltAlt*^{ps} uses a greedy approach to pick a goal subset up front. The greediness of the approach is offset by considering the net benefit of covering a goal not in isolation, but in the context of the potential (relaxed) plan for handling the already selected goals. Once a subset of goal conjuncts is selected, *AltAlt*^{ps} finds a plan

[2] For a more comprehensive study on the complexity and taxonomy of PSP problems see (van den Briel *et al.* 2004).

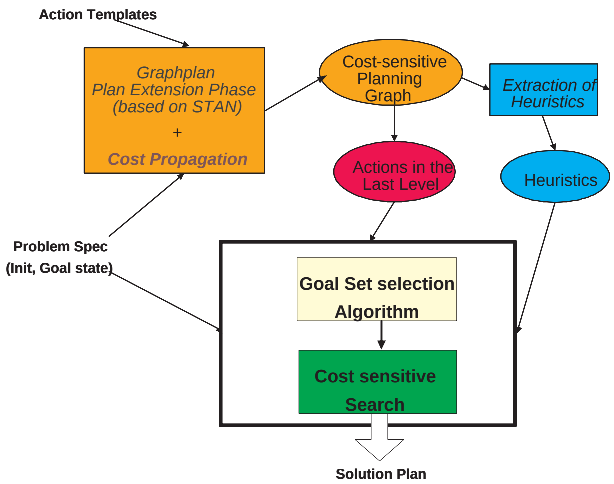


Figure 3: *AltAlt*^{ps} Architecture

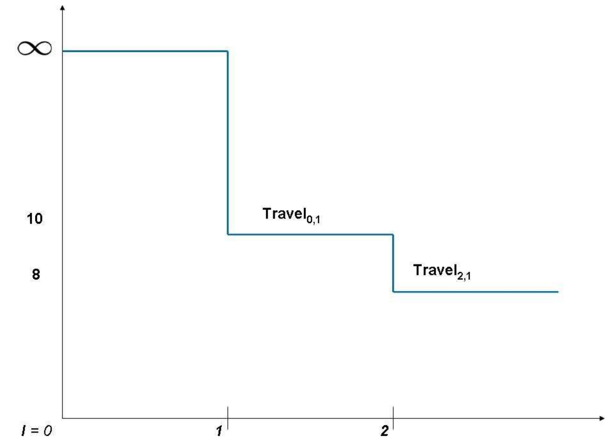


Figure 4: Cost function of $at(waypoint_1)$

that achieves such subset using its regression search engine augmented with cost sensitive heuristics. This description can be seen in the overall architecture of *AltAlt*^{ps} in Figure 3.

Given that the quality of the plans for PSP problems depends on both the utility of the goals achieved and the cost to achieve them, *AltAlt*^{ps} needs heuristic guidance that is sensitive to both action cost and goal utility. Because only the execution costs of the actions and the achievement cost of propositions in the initial state (zero cost) are known, we need to do *cost-propagation* from the initial state through actions to estimate the cost to achieve other propositions, especially the top level goals. We can see in Figure 3 that *AltAlt*^{ps} is using the planning graph structure to compute this cost information. This information over the planning graph is the basis for heuristic estimation in *AltAlt*^{ps}, and is also used to estimate the most beneficial subset of goals upfront and guide the search in the planner. The cost sensitive heuristics, as well as the goal set selection algorithm are described in more detail in the next sections.

Propagating cost as the basis for computing heuristics

Following (Do & Kambhampati 2003), we use cost functions to capture the way cost of achievement changes as the graph gets expanded. In the following, we briefly review the procedure.

The purpose of the cost-propagation process is to build the cost functions $C(f, l_f)$ and $C(a, l_a)$ that estimate the cheapest cost to achieve fluent f at level l_f of the planning graph, and the cost to execute action a at level l_a. At the beginning ($l = 0$), let S_{init} be the initial state and C_a be the cost of action a then[3]: $C(f, 0) = 0$ if $f \in S_{init}$, $C(f, 0) = \infty$ otherwise; $\forall a \in A : C(a, 0) = \infty$. The propagation rules are as follows:

- $C(f, l) = min\{C(a, l) + \mathrm{C}_a) : f \in Eff(a)\}$
- Max-prop: $C(a, l) = max\{C(f, l) : f \in Prec(a)\}$
- Sum-prop: $C(a, l) = \Sigma\{C(f, l) : f \in Prec(a)\}$

The max-propagation rule will lead to an admissible heuristic, while the sum-propagation rule does not. Assume that we want to reach $waypoint_1$ in our rover example. We can reach it directly from $waypoint_0$ within a unit of time, or we can travel through $waypoint_2$ and reach it within two steps. Figure 4 shows the cost function for proposition $p_1 = At(waypoint_1)$, which indicates that the earliest level to achieve p_1 is at $l = 1$ with the lowest cost of 10 (route: $waypoint_0 \rightarrow waypoint_1$). The lowest cost to achieve p_1 reduces to 8 at $l = 2$ (route: $waypoint_0 \rightarrow waypoint_2 \rightarrow waypoint_1$) for the leveled planning graph.

There are many ways to terminate the cost-propagation process (Do & Kambhampati 2003): We can stop when all the goals are achievable, when the cost of all the goals are stabilized (i.e. guaranteed not to decrease anymore), or lookahead several steps after the goals are achieved. For classical planning, we can also stop propagating cost when the graph *levels-off* (Nguyen, Kambhampati, & Sanchez 2002).

Cost-sensitive heuristics

After building the planning graph with cost information, *AltAlt*^{ps} uses variations of the relaxed plan extraction process (Hoffman & Nebel 2001; Nguyen, Kambhampati, & Sanchez 2002) guided by the cost-functions to estimate their heuristic values $h(S)$ (Do & Kambhampati 2003). The basic idea is to compute the cost of the relaxed plans in terms of the costs of the actions comprising them, and use such costs as heuristic estimates. The general relaxed plan extraction process for *AltAlt*^{ps} works as follows:

[3] C_a and $C(a, l)$ are different. If $a = Travel_{0,1}$ then C_a is the travel cost and $C(a, l)$ is the cost to achieve preconditions of a at level l, which is the cost incurred to be at $waypoint_0$ at l.

1. 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 $C(g,l)$) to support g
2. Regress G over action a_g, setting $G = G \cup Prec(a_g) \backslash Eff(a_g)$

The process above 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 extraction procedure indirectly extracts a sequence of actions R_P, 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, in summary we have:

Relax Cost Heuristic 1 $h_{relaxC}(S) = \sum_{a \in R_p} \mathcal{C}_a$

AltAlt^{ps} Goal set selection 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. 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 for supporting $G' + g$, which is biased to (re)use the actions in the relaxed plan R'_P for supporting G'.

Figure 5 gives a description of the goal set selection algorithm. The first block of instructions before the loop initializes our goal subset G',[4] and finds an initial relaxed plan R^*_P for it using the procedure *extractRelaxedPlan(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

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

Figure 5: Goal set selection algorithm.

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'.

In our running example, the original subgoals are $\{g_1 = soil_1,\ g_2 = rock_1,\ g_3 = rock_2,\ g_4 = rock_3,\ g_5 = soil_4\}$, with final costs $C(g,t) = \{17, 17, 14, 34, 24\}$ and utilities vectors U $= \{20, 30, 30, 30, 20\}$ respectively, where $t = leveloff$ in the planning graph. Following our algorithm, our starting goal g would be g_3 because it returns the biggest benefit (e.g. 30 - 14). Then, G' is set to g_3, and its initial relaxed plan R^*_P is computed. Assume that the initial relaxed plan found is $R^*_P = \{travel_{0,2}$, $sample_{rock_2}$, $comm_{rock_2,2,0}\}$. We proceed to compute the best net benefit using R^*_P , which in our example would be $B^*_{MAX} = 30 - (5 + 5 + 4) = 16$. Having found our initial values, we continue iterating on the remaining goals $G = \{g_1, g_2, g_4, g_5\}$. On the first iteration we compute four different set of values, they are: (i) $G_{P_1} = \{g_3 \cup g_1\}$, $R_{P_1} = \{travel_{2,1}$, $sample_{soil_1}$, $comm_{soil_1,2,0}$, $travel_{0,2}$, $sample_{rock_2}$, $comm_{rock_2,2,0}\}$, and $B_{g_{p_1}} = 24$; (ii) $G_{P_2} = \{g_3 \cup g_2\}$, $R_{P_2} = \{travel_{2,1}$, $sample_{rock_1}$, $comm_{rock_1,2,0}$, $travel_{0,2}$, $sample_{rock_2}$, $comm_{rock_2,2,0}\}$, and $B_{g_{p_2}} = 34$; (iii) $G_{P_3} = \{g_3 \cup g_4\}$, $R_{P_3} = \{travel_{0,3}$, $sample_{rock_3}$, $comm_{rock_3,2,0}$, $travel_{0,2}$, $sample_{rock_2}$, $comm_{rock_2,2,0}\}$, and $B_{g_{p_3}} = 12$, and (iv) $G_{P_4} = \{g_3 \cup g_5\}$, $R_{P_4} = \{travel_{0,4}$, $sample_{soil_4}$, $comm_{soil_4,2,0}$, $travel_{0,2}$, $sample_{rock_2}$, $comm_{rock_2,2,0}\}$ with $B_{g_{p_4}} = 12$. Notice then that our net benefit B^*_{MAX} could be improved most if we

[4] $getBestBenefitialGoal(G)$ returns the subgoal with the best benefit, U$_g - C(g,l)$ tradeoff

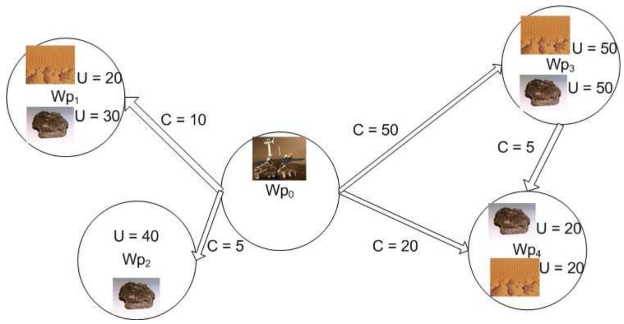


Figure 6: Modified Rover example with goal interactions

consider goal g_2. So, we update $G' = g_3 \cup g_2$, $R^*_P = R_{P_2}$, and $B^*_{MAX} = 34$. The procedure keeps iterating until only g_4 and g_5 remain, which decrease the final net benefit. The procedure returns then $G' = \{g_1, g_2, g_3\}$ as our goal set, which in fact it is the optimal goal set. In this example, there is also a plan that achieves the five goals with a positive benefit, but it is not as good as the plan that achieves the selected G'.

AltWlt: Extending *AltAlt*^{ps} to handle complex goal scenarios

The advantage of *AltAlt*^{ps} for solving PSP problems is that after committing to a subset of goals, the overall problem is simplified to the planning problem of finding the least cost plan to achieve the goal set selected, avoiding the exponential search on 2^n goal subsets. However, the goal set selection algorithm of *AltAlt*^{ps} is greedy, and as a result it is not immune from selecting a bad subset. The main problem with the algorithm is that it does not consider goal interactions. Because of this limitation the algorithm may:

- return a wrong initial subgoal affecting the whole selection process, and
- select a set of subgoals that may not even be achievable due to negative interactions among them.

The first problem corresponds to the selection of the initial subgoal g from where the final goal set will be computed, which is one of the critical decisions of the algorithm. Currently, the algorithm selects only the subgoal g with the highest positive net benefit. Although, this first assumption seems to be reasonable, there may be situations in which starting with the most promising goal may not be the best option. Specifically, when a large action execution cost is required upfront to support a subset of the top level goals, in which each isolated goal component in the subset would have a very low benefit estimate (even negative), precluding the algorithm for considering them initially, but in which the conjunction of them could return a better quality solution. The problem is that we are considering each goal individually in the beginning, without looking ahead into possible combinations of goals in the heuristic computation.

Example 2: Consider the modified Rover example from Figure 6. This time, we have added extra goals, and different cost-utility metrics to our problem. Notice also that the traversal of the paths has changed. For example, we can travel from $waypoint_0$ to $waypoint_1$, but we can not do the reverse. Our top-level goals are $\{g_1 = soil_1,\ g_2 = rock_1,\ g_3 = rock_2,\ g_4 = rock_3,\ g_5 = soil_3,\ g_6 = rock_4,\ g_7 = soil_4\}$, with final costs $C(g, t) = \{19, 19, 14, 59, 59, 29, 29\}$ and utilities U $= \{20, 30, 40, 50, 50, 20, 20\}$ respectively. Following this example, the goal set selection algorithm would choose goal g_3 as its initial subgoal because it returns the highest net benefit (e.g. 40 - 14). Notice this time that considering the most promising subgoal is not the best option. Once the rover reaches $waypoint_2$, it can not achieve any other subgoal. In fact, there is a plan P for this problem with a bigger net benefit that involves going to $waypoint_3$, and then to $waypoint_4$ collecting their samples. Our current goal selection algorithm can not detect P because it ignores the samples on such waypoints given that they do not look individually better than g_3 (e.g. g_4 has a benefit of $50 - 59$). This problem arises because the heuristic estimates derived from our planning graph cost propagation phase assume that the goals are independent, in other words, they may not provide enough information if we want to achieve several consecutive goals.

The second problem about negative interactions among goals is also exhibited in the last example. We already mentioned that if we choose g_3 we can not select any other goal. However, our goal set selection algorithm would also select g_1 and g_2 given that the residual cost returned by the relaxed plan heuristic is lower than the benefit added because it ignores the negative interactions among goals. So, our final goal set would be $G = \{g_3, g_2, g_1\}$, which is not even achievable. Clearly, we need to identify such goal interactions and add some cost metric when they exist. $\square$

We extended our goal selection algorithm in *AltWlt* to overcome these problems. Specifically, we consider multiple groups of subgoals, in which each subgoal from the top level goal set is forced to be true in at least one of the groups, and we also consider adding penalty costs based on mutex analysis to account for complex interactions among goals to overcome the limitations of the relaxed plan heuristic. Although these problems could be solved independently, they can be easily combined and solved together. We discuss these additions in the next section.

Goal set selection with multiple goal groups

The general idea behind the goal set selection with multiple groups procedure is to consider each goal g_i from the top level goal set G as a feasible starting goal, such that we can be able to find what the benefit would be if such goal g_i were to be part of our final goal set selected. The idea is to consider more aggressively multiple combinations of goals in the selection process. Although, we relax the assumption of having a positive net benefit for our starting goals, the approach is still greedy. It modifies the relaxed plan extraction procedure to bias not only towards those actions found in the

```
Procedure MGS(G)
    B*_MAX ← −∞, G* ← ∅
    for (g_i ∈ G)
        GL_i ← nonStaticMutex(g_i, G \ g_i)
        Rp_i ← extractGreedyRelaxedPlan(g_i, ∅)
        G'_i ← greedyGoalSetSelection(g_i, GL_i, Rp_i)
        NB_i ← getUtil(G'_i) − getCost(Rp_i)
        if(NB_i > B*_MAX)
            B*_MAX ← NB_i, G* ← G'_i
    end for
    return G*;
End MGS;
```

Figure 7: Multiple Goal Set Selection Algorithm

relaxed plan of the previous iteration, but also towards those facts that are reflected in the history of partial states of the previous relaxed plan computation to account for more interactions. The algorithm will stop computing a new goal set as soon as the benefit returned decreases. The new algorithm also uses mutex analysis to avoid computing non-achievable goal groups. The output of the algorithm is the goal group that maximizes our net benefit. A more detailed description of the algorithm is shown in Figure 7, and is discussed below.

Given the set of top level goals G, the algorithm considers each goal $g_i \in G$ and finds a corresponding goal subset G'_i with positive net benefit. To get such subset, the algorithm uses a modified greedy version of our original *GoalSetSelection* function (from Figure 5), in which the goal g_i has been set as the initial goal for G'_i, and the initial relaxed plan Rp_i for supporting g_i is passed to the function. Furthermore, the procedure only considers those top-level goals left $GL_i \subseteq G$ which are not pair-wise static mutex with g_i. The set GL_i is obtained using the procedure *nonStaticMutex* in the algorithm. By selecting only the non-static mutex goals, we partially solve the problem of negative interactions, and reduce the running time of the algorithm. However, we still need to do additional mutex analysis to overcome complex goal interactions (e.g. dynamic mutexes); and we shall get back to this below. At each iteration, the algorithm will output a selected goal set G'_i given g_i, and a relaxed plan Rp_i supporting it.

As mentioned in the beginning of this section, the modified $extractGreedyRelaxedPlan$ procedure takes into account the relaxed plan from the previous iteration (e.g. Rp^*_i) as well as its partial execution history to compute the new relaxed plan Rp_i for the current subgoal g_i. The idea is to adjust the aggregated cost of the actions $C(a_i, l)$ supporting g_i, to order them for inclusion in Rp_i, when their preconditions have been accomplished by the relaxed plan Rp^*_i from the previous iteration. Remember that $C(a_i, l)$ has been computed using our cost propagation rules, we decrease this cost when $Prec(a_i) \cap_{\exists a_k \in Rp^*_i} Eff(a_k) \neq \emptyset$ is satisfied. In other words, if our previous relaxed plan Rp^*_i supports already some of the preconditions of a_i it better be the case that such preconditions are not being over-counted in the aggregated cost of the action a_i. This greedy modification of our relaxed plan extraction procedure biases even more to our previous relaxed plans, ordering differently the actions that will be used to support our current subgoal g_i. The idea is to try to adjust the heuristic positively to overcome the independence assumption among subgoals.

For example, on Figure 6, assume that our previous relaxed plan Rp^*_i has achieved the subgoals at $waypoint_3$, and we want to achieve subgoal $g_i = soil_4$. In order to collect the sample, we need to be at $waypoint_4$, the cheapest action in terms of its aggregated cost that supports that condition is $a = travel_{0,4}$ with cost of 20 which precludes g_i for being considered (no benefit added). However, notice that there is another action $b = travel_{3,4}$ with original aggregated cost of 50 (due to its precondition), whose cost gets modified by our new relaxed plan extraction procedure since its precondition (at $waypoint_3$) is being supported indirectly by our previous Rp^*_i. By considering action b, the residual cost for supporting $soil_4$ lowers to 5, and as a result it can be considered for inclusion.

Finally, the algorithm will output the goal set G^* that maximizes the net benefit B^*_{MAX} among all the different goal partitions G'_i. Following Example 2 from Figure 6, we would consider 7 goal groups having the following partitions: $g_1 = soil_1$ & $GL_1 = \{g_2\}$, $g_2 = rock_1$ & $GL_2 = \{g_1\}$, $g_3 = rock_2$ & $GL_3 = \emptyset$, $g_4 = rock_3$ & $GL_4 = \{g_5, g_6, g_7\}$, $g_5 = soil_3$ & $GL_5 = \{g_4, g_6, g_7\}$, $g_6 = rock_4$ & $GL_6 = \{g_7\}$, $g_7 = soil_4$ & $GL_7 = \{g_6\}$. The final goal set returned by the algorithm in this example is $G^* = \{g_4, g_5, g_6, g_7\}$, which corresponds to the fourth partition G'_4, with maximum benefit of 49. Running the original algorithm (from Figure 5) in this example would select goal group $G'_3 = \{g_3\}$ with final benefit of 26.

Even though our algorithm may look expensive since it is looking at different goal combinations on multiple groups, it is still a greedy approximation of the full 2^n combinations of an optimal approach. The reduction comes from setting up the initial subgoals at each goal group at each iteration. The worst case scenario of our algorithm would involve to consider problems with no interactions and high goal utility values, in which the whole set of remaining subgoals would need to be considered at each group. Given n top level goals leading to n goal groups, the worst case running time scenario of our approach would be in terms of $n * \sum_{i=1}^{n-1} i$, which is much better than the factor 2^n.

Penalty costs through mutex analysis

Although the *MGS* algorithm considers static mutexes, it still misses negative interactions among goals that could affect the goal selection process. This is mainly due to the optimistic reachability analysis provided by the planning graph. Consider again Example 2, and notice that goals $g_5 = soil_3$ and $g_7 = soil_4$ are not statically interfering, and they require a minimum of three steps (actions) to become true in the planning graph (e.g. *travel* - *sample* - *comm*). However, at level 3 of the planning graph these goals are mutex, implying that there are some negative interactions among them. Having found such interactions, we could assign a *penalty* P_C to our residual cost estimate for ignoring them.

Penalty costs through subgoal interactions A first approach for assigning such a penalty cost P_C, which we call $NEGFM$,[5] follows our work from (Nguyen, Kambhampati, & Sanchez 2002) considering the binary interaction degree δ among a pair of propositions. The idea is that every time a new subgoal g gets considered for inclusion in our goal set G', we compute δ among g and every other subgoal $g' \in G'$. At the end, we output the pair $[g, g']$ with highest interaction degree δ if any. Recalling from (Nguyen, Kambhampati, & Sanchez 2002) δ gets computed using the following equation:

$$\delta(p_1, p_2) = lev(p_1 \wedge p_2) - max\{lev(p_1), lev(p_2)\} \quad (1)$$

Where $lev(S)$ corresponds to the set level heuristic that specifies the earliest level in the planning graph in which the propositions in the set S appear and are not mutex to each other (Nguyen, Kambhampati, & Sanchez 2002). Obviously, if not such level exists then $lev(S) = \infty$, which is the case for static mutex propositions.

The binary degree of interaction δ provide a clean way for assigning a penalty cost to a pair of propositions in the context of heuristics based on number of actions, given that δ is representing the number of extra steps (actions) required to make such pair of propositions mutex free in the planning graph. Following our current example, $lev(g_5 \wedge g_7) = 5$ (due to dummy actions), as a result $\delta(g_5, g_7) = 2$ represents the cost for ignoring the interactions. However, in the context of our PSP problem, where actions have real execution costs and propositions have costs of achievement attached to them, it is not clear how to compute a penalty cost when negative interactions are found.

Having found the pair with highest $\delta(g, g')_{g' \in G'}$ value, our first solution $NEGFM$ considers the maximum cost among both subgoals in the final level l_{off} of the planning graph as the penalty cost P_C for ignoring such interaction. This is defined as:

$$P_C(g, G')_{NEGFM} = max \left\{ \begin{array}{c} (C(g, l_{off}), C(g', l_{off})) \\ : g' \in G' \wedge max(\delta(g, g')) \end{array} \right\} \quad (2)$$

$NEGFM$ is greedy in the sense that it only considers the pair of interacting goals with maximum δ value. It is also greedy in considering only the maximum cost among the subgoals in the interacting pair as the minimum amount of extra cost needed to overcome the interactions generated by the subgoal g being evaluated. Although $NEGFM$ is easy to compute, it is not very informative affecting the quality of the solutions returned. The main reason for this is that we have already considered partially the cost of achieving g when its relaxed plan is computed, and we are in some sense blindly over-counting the cost if $C(g, l_{off})$ gets selected as the penalty P_C. Despite these clear problems, $NEGFM$ is able to improve in problems with complex interactions over our original algorithm.

[5] Negative Factor: Max

Procedure *NEGFAM*$(G, Rp_G, g', a_{g'})$
  $cost_1 \leftarrow 0, cost_2 \leftarrow 0$
  $P_C \leftarrow 0, maxCost \leftarrow 0$
  for $(g_i \in G)$
    $a_i \leftarrow getSupportingAction(g_i, Rp_G)$
    $cost_1 \leftarrow competingNeedsCost(g_i, a_i, g', a_{g'})$
    $cost_2 \leftarrow interferenceCost(g_i, a_i, g', a_{g'})$
    $maxCost \leftarrow max(cost_1, cost_2)$
    if$(maxCost > P_C)$
      $P_C \leftarrow maxCost$
  end for
  return P_C;
End *NEGFAM*;

Figure 8: Interactions through actions

Penalty costs through action interactions A better idea for computing the negative interactions among subgoals is to consider the interactions among the actions supporting such subgoals in our relaxed plans, and locate the possible reason for such interactions to penalize them. Interactions could arise because our relaxed plan computation is greedy. It only considers the cheapest action[6] to support a given subgoal g', ignoring any negative interactions of the subgoal. Therefore, the intuition behind this idea is to adjust the residual cost returned by our relaxed plans, by assigning a penalty cost when interactions among their actions are found in order to get better estimates. We called this idea $NEGFAM$.[7]

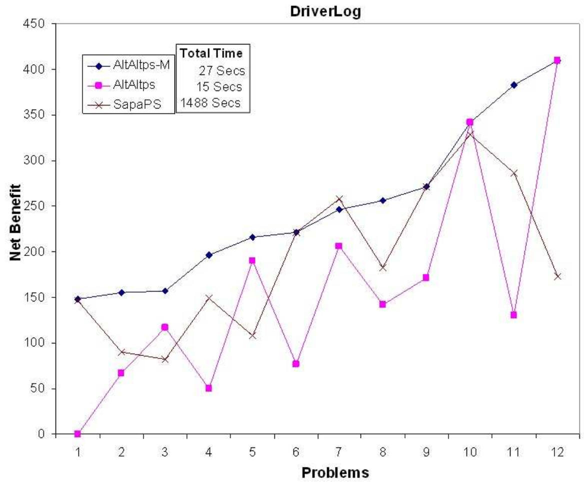
$NEGFAM$ is also greedy because it only considers the actions directly supporting the subgoals in the relaxed plan, and it always keeps the interaction with maximum cost as its penalty cost. In case that there is no supporting action for a given subgoal g' (e.g. if $g' \in I$), the algorithm will take g' itself for comparison. $NEGFAM$ considers the following types of action interaction based on (Weld 1999):

1. Competing Needs: Two actions a and b have preconditions that are *statically* mutually exclusive, or at least one precondition of a is statically mutually exclusive with the subgoal g' given.
2. Interference: Two actions a and b, or one action a and a subgoal g' are interfering if the effect of a deletes b's preconditions, or a deletes g'.

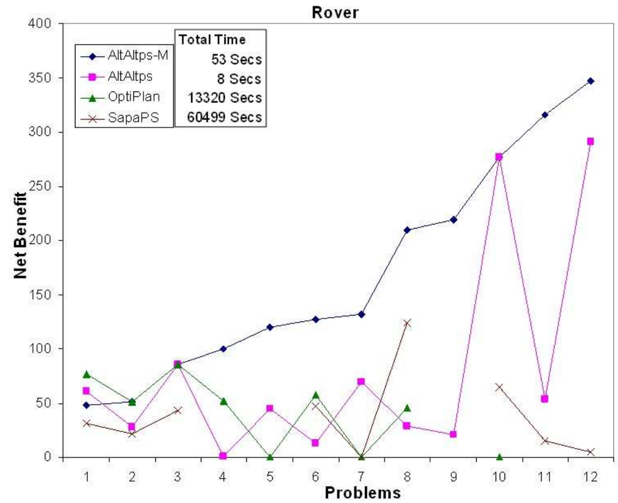
Notice that we are only considering pairs of static mutex propositions when we do the action interaction analysis. The reason for this is that we just want to identify those preconditions that are critically responsible for the actions interactions, and give a penalty cost based on them. Once found a pair of static propositions, we have different ways of penalizing them. We show the description of the $NEGFAM$ technique on Figure 8. The procedure gets the current selected goals G, the relaxed plan Rp_G supporting them, and the subgoal g' being evaluated and action $a_{g'}$ supporting it. Then, it computes two different costs, one based on the com-

[6] With respect to $C(a, l) + \mathcal{C}_a$

[7] Negative Factor By Actions: Max



a) DriverLog Domain



b) Rover Domain

Figure 9: Empirical Evaluation: Solution quality and total running time

peting needs of actions, and the second one based on their interference:

- For competing needs, we identify the proposition with maximum cost in the pair of static preconditions of the actions, and we set P_C to this cost. The idea is to identify what the minimum cost would be in order to support two competing preconditions. Given $p_1 \wedge p_2$, where $p_1 \in Prec(a_i)$ and $p_2 \in Prec(a_{g'})$, or $p_2 = g'$ when $\neg\exists_{a_{g'}}$, the cost is $cost_1$ = $max(C(p_1, leveloff)$,$C(p_2, leveloff))$ if $lev(p_1 \wedge p_2) = \infty$. This penalty gets computed using the procedure $competingNeedsCost(g_i, a_i, g', a_{g'})$ in Figure 8.
- In case of interference, our penalty cost P_C is set to the cheapest alternate way (i.e. action) for supporting a proposition being deleted. The idea is to identify what the additional cost would be in order to restore a critical precondition, which needs to be deleted to achieve another subgoal. Given $p_1 \in Prec(a_i)$, and $\neg p_1 \in Eff(a_{g'})$ or $g' = \neg p_1$, the cost is $cost_2$ = $min\{\mathcal{C}_x : \forall_x\ s.t\ p_1 \in Eff(x)\}$. This cost is computed using the procedure $InterferenceCost(g_i, a_i, g', a_{g'})$.

Our algorithm then selects the cost that maximizes our return value P_C given by the two techniques mentioned above. Our P_C is then added to the residual cost of subgoal g'.

Following our example 2 (Figure 6), we already mentioned that if we chose $g_3 = rock_2$ we would also select $g_1 = soil_1$ and $g_2 = rock_1$ in our original algorithm, which is not feasible. However, by taking into account the negative interactions among subgoals with $NEGFAM$ we would discard such unfeasible sets. For example, suppose that $G = \{g_3\}$ and $Rp_G = \{travel_{0,2}$, $sample_{rock_2,2}$, $comm_{rock_2,2,0}\}$, and the goal being considered for inclusion is $g' = g_1$ with residual cost 19, corresponding to its relaxed plan $Rp_{g'} = \{travel_{0,1}$, $sample_{soil_1,1}$, $comm_{rock_1,1,0}\}$. Notice that the supporting actions for g_3 and g_1 are $comm_{rock_2,2,0}$ and $comm_{soil_1,1,0}$ respectively. These actions have competing needs, one action requires the rover to be at $waypoint2$ while the other one assumes the rover is at $waypoint1$. The penalty cost P_C given by $NEGFAM$ for ignoring such interaction is 10, which is the maximum cost among the static mutex preconditions. Adding this value to our residual cost gives us a final cost of 29, which precludes the algorithm for considering g_1 (i.e. benefit = 20 - 29). Although $NEGFAM$ is also greedy since it may over increase the residual cost of a subgoal g', it improves over our original algorithm and $NEGFM$, returning better quality solutions for problems with complex interactions (as will be shown in our next section).

Empirical Evaluation

In the foregoing, we have described with illustrative examples, how complex interactions may affect the goal set selection process of *AltAlt*^{ps} . Our aim in this section is to show that planning graph reachability heuristics augmented with mutex analysis still provide efficient estimates for solving the PSP NET BENEFIT problem in the presence of complex goal interactions.

Since there are no benchmark PSP problems, we used existing STRIPS planning domains from the 2002 International Planning Competition (Long & Fox 2003), and modified them to support explicit declaration of goal utilities and action costs. In particular, our experiments include the domains of DriverLog and Rover. For the DriverLog domain, goal utilities ranged from 40 to 100, while the costs of the actions ranged from 3 to 70. Goal interactions were increased by considering bigger action execution costs, and modified paths in the network that the planning agent has to traverse. The idea was to place the most rewarding goals in the costlier paths of the network in order to increase the complexity of finding the most beneficial subset of goals. For the Rover domain, utilities ranged from 20 to 30, and action execution costs ranged from 4 to 45. In addition to the modifications introduced in the DriverLog domain to increase the level of interactions among goals, the Rover domain also allows for dead-ends, and loops in the network that the rover has to traverse. The idea was to present more options for the planning agent to fail. Consequently, it proved to be much more difficult to solve restricting even more the attainability of multiple goal sets. The design of this domain was inspired by the Rover domain presented by Smith(2004), without considering resources in our domain description.

We compared our new approach *AltWlt* to its predecessor (*AltAlt*^{ps}), and *Sapa*^{ps} (van den Briel *et al.* 2004). Although *Sapa*^{ps} also uses planning graph heuristics to rank their goals, it does not provide mutex analysis and its search algorithm is different. Unlike *AltWlt*, *Sapa*^{ps} does not select a subset of the goals up front, but uses an anytime A* heuristic search framework in which goals are treated as "soft constraints" to select them during planning. Any executable plan is considered a potential solution, with the quality of the plan measured in terms of its net benefit. We considered it pertinent to take into account both planners to see more clearly the impact of the techniques introduced in this paper. We have also included in this section a run of *OptiPlan* (van den Briel *et al.* 2004) in the Rover domain, to demonstrate that our greedy approach is able to return high quality plans. *OptiPlan* is a planner that builds on the work of solving planning problems through IP (Vossen, Ball, & Nau 1999), which generates plans that are optimal for a given plan length.[8] We did not compare to the approach presented by Smith(2004) because his approach was not yet available by the time of this writing.[9] All four planners were run on a P4 2.67Ghz CPU machine with 1.0GB RAM.

Figure 9 shows our results in the DriverLog and Rover domains. We see that *AltWlt* outperforms *AltAlt*^{ps} and *Sapa*^{ps} in most of the problems, returning higher quality solutions. In fact, it can be seen that *AltWlt* returns 13 times as much net benefit on average than *AltAlt*^{ps} in the DriverLog domain (i.e a 1300% benefit increase). A similar scenario occurs with *Sapa*^{ps} , where *AltWlt* returns 1.42 times as much more benefit on average (a 42% benefit increase). A similar situation occurs with the Rover domain in Figure 9 (b), in which *AltWlt* returns 10 and 12 times as much more benefit on average than *AltAlt*^{ps} and *Sapa*^{ps} respectively. This corresponds to a 1000% and 1200% benefit increase over them. Although *OptiPlan* should in theory return optimal solutions for a given length, it is not able to scale up, reporting only upper bounds on most of its solutions. Furthermore, notice also in the plots that the total running time taken by *AltWlt* incurs a very little additional overhead over *AltAlt*^{ps} , and it is completely negligible in comparison to *Sapa*^{ps} or *OptiPlan*.

Looking at the run times, it could appear at first glance that the set of problems are relatively easy to solve given the total accumulated time of *AltAlt*^{ps} . However, remember that for many classes of PSP problems, a trivially feasible, but decidedly non-optimal solution would be the "null" plan, and *AltAlt*^{ps} is in fact returning faster but much lower quality solutions. We can see that the techniques introduced in *AltWlt* are helping the approach to select better goal sets by accounting for interactions. This is not happening in *AltAlt*^{ps} , where the goal sets returned are very small and easier to solve.

We also tested the performance of *AltWlt* in problems with less interactions. Specifically, we solved the suite of random problems from (van den Briel *et al.* 2004), including the ZenoTravel and Satellite planning domains (Long & Fox 2003). Although the gains there were less impressive, *AltWlt* was able to produce in general better quality solutions than the other approaches, returning bigger total net benefits.

Related Work

As we mentioned earlier, there has been very little work on PSP in planning. One possible exception is the PYRRHUS planning system (Williamson & Hanks 1994) which considers an interesting variant of the partial satisfaction planning problem. In PYRRHUS, the quality of the plans is measured by the utilities of the goals and the amount of resource consumed. Utilities of goals decrease if they are achieved later than the goals' deadlines. Unlike the PSP problem discussed in this paper, all the logical goals still need to be achieved by PYRRHUS for the plan to be valid. It would be interesting to extend the PSP model to consider degree of satisfaction of individual goals.

More recently, Smith (2003) motivated oversubscription problems in terms of their applicability to the NASA planning problems. Smith (2004) also proposed a planner for oversubscription in which the solution of the abstracted planning problem is used to select the subset of goals and the orders to achieve them. The abstract planning problem is built by propagating the cost on the planning graph and constructing the *orienteering* problem. The goals and their orderings are then used to guide a POCL planner. In this sense, this approach is similar to *AltAlt*^{ps} ; however, the orienteering problem needs to be constructed using domain-knowledge for different planning domains. Smith (2004) also speculated that planning-graph based heuristics are not particularly suitable for PSP problems where goals are highly interacting. His main argument is that heuristic estimates derived from planning graphs implicitly make the assumption that goals are independent. However, as shown in this paper, reachability estimates can be improved using the mutex information also contained in planning graphs, allowing us to solve problems with complex goal interactions.

Probably the most obvious way to optimally solve the PSP NET BENEFIT problem is by modeling it as a fully-observable Markov Decision Process (MDP) (Boutilier,

[8]For a more comprehensive description on *OptiPlan* see (van den Briel *et al.* 2004).

[9]Smith's approach takes as input a non-standard PDDL language, without the explicit representation of the operators descriptions.

Dean, & Hanks 1999) with a finite set of states. MDPs naturally permit action cost and goal utilities, but we found in our studies that an MDP based approach for the PSP NET BENEFIT problem appears to be impractical, even the very small problems generate too many states. To prove our assumption, we modeled a set of test problems as MDPs and solved them using SPUDD (Hoey *et al.* 1999).[10] SPUDD is an MDP solver that uses value iteration on algebraic decision diagrams, which provides an efficient representation of the planning problem. Unfortunately, SPUDD was not able to scale up, solving only the smallest problems.[11]

Over-subscription issues have received relatively more attention in the scheduling community. Earlier work in scheduling over-subscription used greedy approaches, in which tasks of higher priorities are scheduled first (Kramer & Giuliano 1997; Potter & Gasch 1998). The approach used by *AltWlt* is more sophisticated in that it considers the residual cost of a subgoal in the context of an existing partial plan for achieving other selected goals, taking into account complex interactions among the goals. More recent efforts have used stochastic greedy search algorithms on constraint-based intervals (Frank *et al.* 2001), genetic algorithms (Globus *et al.* 2003), and iterative repairing technique (Kramer & Smith 2003) to solve this problem more effectively.

Conclusions

Motivated by the observations in (Smith 2004) that Planning Graph based heuristics may not be able to handle complex subgoals interactions in PSP problems, we extended our previous work on *AltAlt*^{ps} (van den Briel *et al.* 2004) to overcome such problems. In this paper, we have introduced *AltWlt*, a greedy approach based on *AltAlt*^{ps} that augments its goal set selection procedure by considering multiple goal groups and mutex analysis.

The general idea behind our new approach is to consider more aggressively multiple combinations of subgoals during the selection process. *AltWlt* is still greedy since it modifies the original relaxed plan extraction procedure to better account for positive interactions among subgoals, adding also penalty costs for ignoring negative interactions among the actions supporting them.

Our empirical results show that *AltWlt* is able to generate plans with better quality than the ones generated by its predecessor and *Sapa*^{ps}, while incurring only a fraction of the running time. We demonstrated that the techniques employed in *AltWlt* really pay off in problems with highly interacting goals. This shows that selection of objectives in over-subscription problems could be handled using planning-graph based heuristics.

[10]We thank Will Cushing and Menkes van den Briel who first suggested the MDP modeling idea.

[11]Details on the MDP model and results can be found in (van den Briel *et al.* 2005).

References

Boutilier, C.; Dean, T.; and Hanks, S. 1999. Decision-theoretic planning: Structural assumptions and computational leverage. *JAIR* 11:1–94.

Do, M., and Kambhampati, S. 2003. Sapa: a multi-objective metric temporal planner. *JAIR* 20:155–194.

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

Globus, A.; Crawford, J.; Lohn, J.; and Pryor, A. 2003. Scheduling earth observing sateliites with evolutionary algorithms. In *Proceedings Int. Conf. on Space Mission Challenges for Infor. Tech.*

Hoey, J.; St-Aubin, R.; Hu, A.; and Boutilier, C. 1999. Spudd: Stochastic planning using decision diagrams. In *Proceedings of the 15th Annual Conference on Uncertainty in Artificial Intelligence (UAI-99)*, 279–288.

Hoffman, J., and Nebel, B. 2001. The ff planning system: Fast plan generation through heuristic search. *JAIR* 14:253–302.

Kramer, L., and Giuliano, M. 1997. Reasoning about and scheduling linked hst observations with spike. In *Proceedings of Int. Workshop on Planning and Scheduling for Space*.

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

Long, D., and Fox, M. 2003. The 3rd international planning competition: results and analysis. *JAIR* 20:1–59.

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 *Proceedings of SpaceOps*.

Smith, D. 2003. The mystery talk. Plannet Summer School.

Smith, D. 2004. Choosing objectives in over-subscription planning. In *Proceedings of ICAPS-04*.

van den Briel, M.; Sanchez, R.; Do, M.; and Kambhampati, S. 2004. Effective approaches for partial satisfation (over-subscription) planning. In *Proceedings of AAAI-04*.

van den Briel, M.; Sanchez, R.; Do, M.; and Kambhampati, S. 2005. Planning for over-subscription problems. Arizona State University, Technical Report.

Vossen, T.; Ball, M.; and Nau, D. 1999. On the use of integer programming models in ai planning. In *Proceedings of IJCAI-99*.

Weld, D. 1999. Recent advances in ai planning. *AI Magazine* 20(2):93–123.

Williamson, M., and Hanks, S. 1994. Optimal planning with a goal-directed utility model. In *Proceedings of AIPS-94*.