Partial Satisfaction (Over-Subscription) Planning as Heuristic Search

Minh B. Do & Subbarao Kambhampati*
Department of Computer Science and Engineering
Arizona State University, Tempe AZ 85287-5406

Abstract

Many planning problems can be characterized as over-subscription problems in that goals have different utilities, actions have different costs and the planning system must choose a subset that will provide the greatest net benefit. This type of problems can not be solved by existing planning systems, where goals are assumed to have uniform utility, and the planner can terminate only when *all of the goals are achieved*. Existing methods for such problems use greedy approaches, which pre-select a subset of goals based on their estimated utility, and solve for those goals. Unfortunately, greedy methods, while efficient, can produce plans of arbitrarily low quality. In this paper, we introduce a more sophisticated heuristic search framework for over-subscription planning problems. In our framework, top-level goals are treated as *soft-constraints* and the search is guided by a relaxed-plan based heuristic that estimates the most beneficial set of goals from a given state. We implement this search framework in the context of *Sapa*, a forward state-space planner. We provide preliminary empirical results that demonstrate the effectiveness of our approach in comparison to a greedy approach.

1 Introduction

Many planning problems can be characterized as over-subscription problems (c.f. Smith(2003; 2004)) in that goals have different values and the planning system must choose a subset that can be achieved within the time and resource limits. Examples of the over-subscription problems include many of NASA planning problems such as planning for telescopes like Hubble[Kramer & Giuliano, 1997], SIRTF[Kramer, 2000], Landsat 7 Satellite[Potter & Gasch, 1998]; and planning science for Mars rover [Smith, 2003]. Often, given the resource and time limits, not all goals can be accomplished, and thus the planner needs to focus on a subset of goals that have highest total value given those constraints. In this paper, we consider a subclass of the over-subscription

*Minh Do's current address is: Palo Alto Research Center, Room 1530; 3333 Coyote Hill Road; Palo Alto CA 94304-1314. We thank Romeo Sanchez, Menkes van den Briel, Will Cushing and David Smith for many helpful comments. This research is supported in part by the NSF grant IIS-0308139 and the NASA grants NCC2-1225 and NAG2-1461.

problem where goals have different utility (or values) and actions incur different execution costs. The objective is to find the best *beneficial* plan, that is the plan with the best tradeoff between the total benefit of achieved goals and the total execution cost of actions in the plan. We refer to this subclass of the over-subscription problems as partial-satisfaction planning (PSP) problems (since not all the goals need to be satisfied by the final plan), and illustrate it with an example:

Example: In Figure 1, we show the travel example that we will use throughout the paper. In this example, a student living in Las Vegas (LV) needs to go to San Jose (SJ) to present a AAAI paper. The cost for traveling is $C(travel(LV, SJ)) = 230$ (airfare + hotel). To simplify the problem, we assume that if the student arrives at San Jose, he automatically achieves the goal $g_1 = Attended\_AAAI$ with utility $U(g_1) = 300$ (equals to the AAAI's student scholarship). The student also wants to go to Disneyland (DL) and San Francisco (SF) to have some fun and to San Diego (SD) to see the zoo. The utilities of having fun in those three places ($U(g_2 = HaveFun(DL))$, $U(g_3 = HaveFun(SF))$, $U(g_4 = SeeZoo(SD))$) and the cost of traveling between different places are shown in Figure 1. The student needs to find a traveling plan that gives him the best tradeoff between the utilities of being at different places and the traveling cost (transportation, hotel, entrance ticket etc.). In this example, the best plan is $P = \{travel(LV, DL), travel(DL, SJ), travel(SJ, SF)\}$ that achieves the first three goals g_1, g_2, g_3 and ignores the last goal $g_4 = SeeZoo(SD)$.

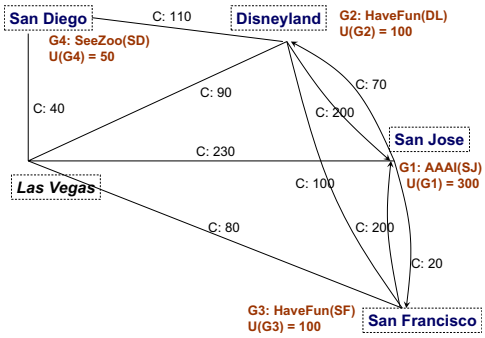

Figure 1: The travel example

Current planning systems are not designed to solve the over-subscription or PSP problems. Most of them expect a set of conjunctive goals of equal value and the planning process does not terminate until all the goals are achieved. Extending the traditional planning techniques to solve the over-subscription problem poses several challenges:

- The termination criteria for the planning process change because there is no fixed set of conjunctive goals to satisfy.
- Heuristic guidance in the traditional planning systems are designed to guide the planners to achieve a fixed set of goals. They can not be used directly in the over-subscription problem.
- Plan quality should take into account not only action costs but also the values of goals achieved by the final plan.

Over-subscription problems have received some attention in scheduling, using the "greedy" approaches. Straightforward adaptation to planning would involve considering goals in the decreasing order of their values. However, goals with higher values may also be more expensive to achieve and thus may give the cost vs. benefit tradeoff that is far from optimal. Moreover, interactions between goals may make the cost/benefit to achieve a set of goals quite different from the value of individual goals. Recently, Smith (2004) and van den Briel et. al (2004) proposed two approaches for solving partial satisfaction planning problems by heuristically selecting a subset S of goals that appear to be the most beneficial and then use the normal planning search to find the least cost solution that achieve all goals in S. If there are n goals then this approach would essentially involves selecting the most beneficial subset of goals among a total of 2^n choices. While this type of approach

	Classical Planning	PSP
Potential solutions	Feasible plans: achieve all goals	Beneficial plans: +ve net benefit
Termination	Any plan achieving all goals	Plan with the highest net benefit.
g value	Num. actions in the current plan	Net benefit of the current plan
h value	Num. additional actions needed	Additional net benefit that can be achieved

Figure 2: Comparing the A* models of normal and PSP planning problems

is less susceptible to inoptimal plans compared to the naive greedy approach, it too is overly dependent on the informedness of the initial goal-set selection heuristic. If the heuristic is not informed, then there can be beneficial goals left out and S may contain unbeneficial goals.

In this paper, we discuss a heuristic search framework for solving the PSP problems that involves treating the top-level goals as *soft-constraints*. The search framework does not concentrate on achieving a particular subset of goals, but rather decides what is the best solution for each node in a search tree. The evaluations of the g and h values of nodes in the A* search tree are based on the subset of goals achieved in the current search node and the best potential beneficial plan from the current state to achieve a subset of the remaining goals. We implement this technique over the *Sapa* planner [Do & Kambhampati, 2003] and show that it can produce high quality plans compared to the greedy approach.

The rest of the paper is organized as follows: in the next two sections, we discuss a general A* search model partial satisfaction planning problem and how heuristic guidance measures are calculated. We then provide some empirical results and conclude the paper with discussions on the related and future work.

2 Handling Partial Satisfaction Planning Problems with A* Search

Most of the current successful heuristic planners [Bonet & Geffner, 1997; Hoffmann & Nebel, 2001; Do & Kambhampati, 2003; Nguyen et. al., 2001; Edelkamp, 2003] use weighted A* search with heuristic guidances extracted from solving the relaxed problem ignoring the delete list. In the heuristic formula $f = g + w * h$ guiding the weighted A* search, the g value is measured by the total cost of actions leading to the current state and the h value is the estimated total action cost to achieve all goals from the current state (forward planner) or to reach the current state from the initial state (backward planner).

Compared to the traditional planning problem, PSP problem has several new requirements: (i) goals have different utility values; (ii) not all the goals need to be accomplished in the final plan; (iii) the plan quality is not measured by the total action cost but the tradeoff between the total achieved goals' utilities and the total action cost. Therefore, in order to employ the A* search, we need to modify the way each search node is evaluated as well as the criteria used to terminate the search process (see Figure 2). To keep the discussion simple, in this paper we will consider search in the context of a forward planner; the model can however be easily extended to regression planning.

In the forward planners, applicable actions are applied to the current state to generate new states. Generated nodes are sorted in the queue by their f values. The search stops when the first node taken from the queue satisfies all the pre-defined conjunctive goals. In our ongoing example, at the initial state $S_{init} = \{at(LV)\}$, there are four applicable actions $A_1 = travel(LV, DL)$, $A_2 = travel(LV, SJ)$, $A_3 = travel(LV, SF)$ and $A_4 = travel(LV, SD)$ that lead to four states $S_1 = \{at(DL), g_2\}$, $S_2 = \{at(SJ), g_1\}$, $S_3 = \{at(SF), g_3\}$, and $S_4 = \{at(SD), g_4\}$. Assume that we pick state S_1 from the queue, then applying action $A_5 = travel(DL, SF)$

to S_1 would lead to state $S_5 = \{at(SF), g_2, g_3\}$. For a given state S, let partial plan $P_P(S)$ and goal set $G(S)$ be the plan leading from the initial state S_{init} to S and the set of goals accomplished in S. The g value in the formula $f = g + w * h$ guiding the weighted A* search for solving PSP problem is calculated as the difference between the cumulative utility of the goals covered, and the cumulative cost of the actions used:

g value: $g(S) = U(G(S)) - C(P_P(S))$

Thus, for state S_5, the total accumulated utility and cost values are: $U(G(S_5)) = U(g_2)+U(g_3) = 100+100 = 200$, and $C(P_P(S_5)) = C(A_1) + C(A_5) = 90 + 100 = 190$.

For a given state S, let P_R be a plan segment that is applicable in S (i.e., it can be executed starting at S), and $S' = Apply(P_R, S)$ is a state resulting from applying P_R to S. Like $P_P(S)$, the cost of P_R is the sum of the costs of all actions in P_R. The utility of the (partial) plan P_R according to state S is defined as follows: $U(Apply(P_R, S)) = U(G(S')) - U(G(S))$.

Best beneficial remaining plan: *For a given state S, the best beneficial remaining plan P_S^B is a plan applicable in S and there is no other plan P applicable in S such that: $U(Apply(P, S)) - C(P) > U(Apply(P_S^B, S)) - C(P_S^B)$*

The optimal *utility-to-go* of a given state is calculated as follows:

h^* **value**: $h^*(S) = U(Apply(P_S^B, S)) - C(P_S^B)$

Notice that since the empty plan $P_\emptyset$ containing no actions is applicable in all states, and $U(Apply(P_\emptyset, S)) = C(P_\emptyset) = 0$, $U(Apply(P_S^B, S)) - C(P_S^B) \geq 0$ for any state S.

In our ongoing example, from state S_1, the most beneficial plan is $P_{S_1}^B = \{travel(DL, SJ), travel(SJ, SF)\}$, and $U(Apply(P_{S_1}^B, S_1)) = U(\{g_1, g_2, g_3\}) - U(\{g_2\}) = 300+100+100-100 = 400$, $C(P_{S_1}^B) = 200+20 = 220$, and thus $h^*(S_1) = 400 - 220 = 180$. Computing $h^*(S)$ value directly is impractical as we need to search for P_S^B in the space of all potential plans from S'. In the next section, we will discuss a heuristic approach to approximate the h^* value of a given search node S using an h function that involves approximating P_S^B using a relaxed plan for going from S to G.

The general search algorithm, which uses the value $f = g + w * h$ to sort the nodes in the queue in a decreasing order, is described in Figure 3. In this algorithm, search nodes are categorized as follows:

Beneficial Node: *S is a beneficial node if $g(S) > 0$.*

Thus, beneficial nodes S are nodes that give positive net benefit even if no more actions are applied to S. In our ongoing example, all five nodes S_1, S_2, S_3, S_4, S_5 are beneficial nodes. If we decide to extend S_1 by applying the action $A_6 = travel(DL, LV)$ then we will get state $S_6 = \{at(LV), HaveFun(DL)\}$, which is not a beneficial node ($g(S_6) = 100 - 180 = -80$).

Termination Node: *S_T is a termination node if: (i) $h(S_T) = 0$, (ii) $g(S_T) > 0$, and (iii) $\forall S : g(S_T) > f(S)$.*

Termination node S_T is the *best* beneficial node in the queue. Moreover, because $h(S_T) = 0$, there is no benefit of extending S_T and therefore we can terminate the search at S_T. Notice that if the heuristic is *admissible* and we used $f = g + h$,[1] then the set of actions leading to S_T is guaranteed to be an *optimal solution* for the PSP

[1] In this case, a heuristic is admissible if $h(S)$ over-estimates (higher than or equal to) the $U(Apply(P_S^B, S)) - C(P_S^B)$

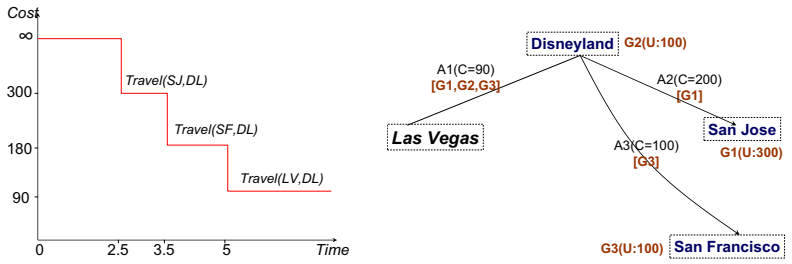

Figure 4: Samples of a cost function (of goal $At(DL)$) and the relaxed plan.

problem.

Unpromising Node: *S is a unpromising node if $f(S) \leq 0$.*

Unpromising nodes are not only not beneficial currently (i.e., $g(S) \leq 0$), but none of their descendant nodes are expected to have positive utility. For example, if we take state S_2 and extend it by applying action $A_7 = travel(SJ, LV)$ (going back to LV) then we get to state $S_7 = \{at(LV), Attended\_AAAI\}$ with $U(S_7) = 300$, $C(S_7) = 460$. $h^*(S_7) = 20$ ($P^B_{S_7} = travel(LV, SF)$) is positive but not enough to balance the total cost and thus S_7 is a unpromising node.[2]

01. State Queue: *SQ*=$\{S_{init}\}$
02. Best beneficial node: $N_B = \emptyset$
03. Best benefit: $B_B = 0$
04. **while** *SQ*$\neq\{\}$
05. *S*:= *Dequeue*(*SQ*)
06. **if** $(g(S) > 0) \wedge (h(S) = 0)$ **then**
07. *Terminate Search;*
08. **Nondeterministically select** *A* applicable in *S*
09. *S'* := *Apply*(*A*,*S*)
10. **if** $g(S') > B_B$ **then**
11. $PrintBestBeneficialNode(S')$
12. $N_B \leftarrow S'$; $B_B \leftarrow g(S')$
13. **if** $f(S) \leq 0$ **then** /**unpromising* */
14. $Discard(S)$
15. **else** *Enqueue*(*S'*,*SQ*)
16. **end while**;

Figure 3: Anytime A* search algorithm for PSP problems.

As described in Figure 3, the search algorithm starts with the initial state S_{init} and keeps dequeuing the best promising node S (i.e. highest f value). If S is a termination node, then we stop the search. If S is not, then we extend S by applying applicable actions A to S. If the newly generated node $S' = Apply(A, S)$ is a beneficial node and has a better $g(S')$ value than the best beneficial node visited so far, then we print the plan leading from S_{init} to S'. Finally, if S' is not a unpromising node, then we will put it in the search queue SQ sorted in the decreasing order of f values.

Notice that because we keep outputting the best beneficial nodes while conducting search, this is an anytime algorithm. Thus, we can impose the time or search node limit on the algorithm and try to find the best plan within those limits. To demonstrate the anytime search framework, let's assume that the nodes S_1, S_2, S_3, S_4

value.

[2]Note that if a given node is not a *beneficial node* then it does not mean that it's a *unpromising node*. A given node S can have value $g(S) < 0$ but $f(S) = g(S) + h(S) > 0$ and is still promising to be extended to reach a beneficial node later.

are generated in that order when we extend S_{init}. Starting with the best benefit value of $B_B = 0$, the planner will first output the plan $P_1 = P_P(S_1) = A_1$ leading to S_1 and change the value of B_B to $B_B = g(S_1) = 100 - 90 = 10$. Then, when S_2 is generated, because $g(S_2) = 70 > B_B = 10$, the planner outputs $P_2 = P_P(S_2) = A_2$ as the new best plan. Nodes S_3 and S_4 are less beneficial and are just put back in SQ. Assuming a perfect heuristic (the optimal plan for this example is given in the first section), the planner will pick S_1 to extend. By applying action $A_7 = travel(DL, SJ)$ to S_1, we get to state S_7 that is more beneficial than the best state visited ($g(S_7) = 400 - 290 = 110 > B_B = 70$) and thus we will output $P_3 = P_P(S_7) = \{travel(LV, DL), travel(DL, SJ)\}$. The algorithm continues until we reach the termination node $S_8 = Apply(travel(SJ, SF), S_7)$.

3 Heuristic Estimation (Estimating P_S^B using cost-sensitive relaxed plans)

For a given state S, while calculating the $g(S)$ value is trivial, estimating the $h^*(S)$ value is not an easy task as we have to first guess the plan P_S^B. If all the goals are reachable, the actions have uniform cost and the goals have substantially higher utilities than action costs, then finding the least cost solution for a traditional planning problem is a special case of finding $P_{S_{init}}^B$ for the PSP problem. To get around this, we approximate P_S^B in terms of a relaxed plan for going from S to the goal state. There are two challenges even with this (1) we need to make relaxed plans sensitive to action costs and (2) we need to take into account the fact that not all goals in the goal state need be achieved (in other words, the relaxed plan may need only focus on a subset of the goals).

To make the relaxed plan extraction sensitive to costs, we use the cost-sensitive temporal planning graph used in *Sapa*[Do & Kambhampati, 2003]. We first build the time-sensitive cost functions for facts in the planning graph until the estimated costs to achieve the individual goals are stabilized. Assume that the student can only go to SJ and SF from LV by airplane, which take respectively 1.0 and 1.5 hour. He can also travel by car from LV, SJ, and SF to DL in 5.0, 1.5 and 2.0 hours, respectively. On the left side of Figure 4, we show the cost function for goal $g_2 = At(DL)$, which indicates that the earliest time to achieve g_2 is at 2.5 (hour) with the lowest cost of 300 (route: $LV \rightarrow SJ \rightarrow DL$). The lowest cost to achieve g_2 reduces to 180 at $t = 3.5$ (route: $LV \rightarrow SF \rightarrow DL$) and again at $t = 5.0$ to 90 (direct path: $LV \rightarrow DL$).

To handle the fact that not all goals need be supported by the relaxed plan, we start with a relaxed plan supporting all the goals and "whittle" it down by not supporting goals which are "expensive" (in that the actions needed to support the goal cost more than the utility provided by the goal).[3]

Specifically, using the cost functions for goals and other facts in the problem, we first heuristically extract the least cost plan that achieves the remaining goals. For example, if we need to find a supporting action to support goal g_2 at time $t = 4$, then the least cost action is $A = travel(SF, DL)$ (Figure 4). Starting with the top level goals, if we choose an action A to satisfy goal g, then we add the preconditions of A to the set of goals to be achieved. In our ongoing example, if we select $A = travel(DL, SF)$ to satisfy $g_3 = HaveFun(SF)$, then the goal $g = at(DL)$ is added to the current set of subgoals. The process terminates when all the subgoals are satisfied by the initial state. We then use a second scan through the extracted relaxed plan to remove goals that do not offset the cost of the actions supporting them (along with the actions that contribute solely to the achievement of those goals). To do this purpose, we build the *supported-goals* list GS for each action A and fact P starting from the top level goals as follows: $GS(A) = \bigcup GS(P) : P \in Effect(A)$ and $GS(P) = \bigcup GS(A) : P \in Precond(A)$.

[3]Notice that another way of handling this would have been to interleave the goal selection and relaxed plan construction process. This is what is done by $AltAlt^{ps}$ [van den Briel et. al., 2004].

Assume that our algorithm extracts the relaxed plan $P = RP(S_{init}) = \{A1 : travel(LV, DL), A2 : travel(DL, SJ), A3 : travel(DL, SF)\}$[4]. On the right side of Figure 4, we show this relaxed plan along with goals each action supports. We build the supported-goals set for each action by going backward from the top level goals. For example, action A_2 and A_3 support only g_1 and g_3 so the goal support list for those two actions will be $GS(A_2) = \{g_1\}$ and $GS(A_3) = \{g_3\}$. The precondition of those two actions, $At(DL)$, would in turn contribute to both these goals $GS(At(DL)) = \{g_1, g_3\}$. Finally, because A_1 supports both g_2 and $At(DL)$, $GS(A_1) = GS(g_2) \cup GS(At(DL)) = \{g_1, g_2, g_3\}$.

Using the supported-goals sets, for each subset S_G of goals, we can identify the subset $SA(S)$ of actions that contribute only to the goals in S_G. If the cost of those actions exceeds the sum of utilities of goals in S_G, then we can remove S_G and $SA(S_G)$ from the relaxed plan. In our example, action A_3 is the only one that solely contributes to the achievement of g_3. Since $C(A_3) \geq U(g_3)$, we can remove A_3 and g_3 from consideration. The other two actions A_1, A_2 and goals g_1, g_2 all appear beneficial. In our current implementation, we consider all subsets of goals of size 1 or 2 for possible removal. After removing unbeneficial goals and actions (solely) supporting them, the cost of the remaining relaxed plan and the utility of the goals that it achieves will be used to compute the h value. Thus, in our ongoing example, $h(S) = (U(g_1) + U(g_2)) - (C(A_1) + C(A_2)) = (100 + 300) - (90 + 200) = 110$.

We note that the current heuristic used in $Sapa^{PS}$ is not admissible, this is because: (i) a pruned unpromising node may actually be promising (i.e. extendible to reach node S with $g(S) > B_B$); and (ii) a termination node may not be the best beneficial node. In our implementation, even though weight $w = 1$ is used in equation $f = g + w * h$ to sort nodes in the queue, a different value $w = 2$ is used for pruning (unpromising) nodes with $f = g + w * h \leq B_B$. Thus, only nodes S with estimated heuristic value $h(S) \leq \frac{1}{w} \times h^*(S)$ are pruned. For the second issue, we can continue the search for a better beneficial nodes after a termination node is found until some resource limits are reached (e.g. reached certain number of search node limit).

4 Empirical Evaluation

We have implemented the heuristic search algorithm for PSP problems discussed in this paper on top of the *Sapa* planner. We call the new planner $Sapa^{PS}$. We tested our planner on a set of randomly-generated problems for the two set of problems (*ZenoTravel* and *Satellite*) used in the Third International Planning Competition (IPC3) and compared it with a greedy algorithm representative of other existing approaches (see below). All tests were run in the Cygwin environment using a Pentium IV 1.4GHz with 768MB RAM machine with 1200 seconds time limit. In this section, we first discuss the benchmark problems and then the experimental results.

Generating Test Problems: Given that in general, a given action cost is decided by the amount of metric resources consumed and/or the time spent by that action, we decided to generate the PSP Net Benefit problems from the set of metric temporal planning problems used in IPC3. In particular, we generated the PSP versions of the problems in the ZenoTravel and Satellite domains as follows:

Domain File: We modify the domain files by adding the cost field to each action description. Each action cost is represented as a formula involving metric functions that exist in the original problem descriptions and also new functions that we add to the domain description file. The cost field utilizes both the functions representing metric quantities in the original problem descriptions, and the newly introduced cost-related functions used to

[4]This is not a legal plan because A_2 and A_3 are mutually exclusive. However, we relaxed negative interactions, and thus it appears valid.

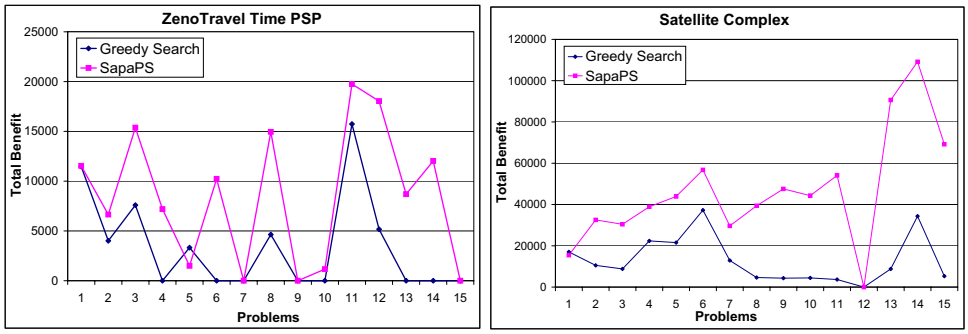

Figure 5: Comparing solution quality between $Sapa^{PS}$ and greedy search.

convert the temporal and resource consumption aspects of each action into a uniform plan benefit represented by the amount of money spent (e.g. ZenoTravel domain) or energy consumed (e.g. Satellite domain).[5]

Problem File: For each domain, we write a Java program that parses the problem files used in the IPC3 in each domain and generates the PSP version of that problem with cost-related function values randomly generated within appropriate upper and lower bounds. Like the metric functions related to the action costs, the goal utilities are also randomly generated within different upper/lower bounds.[6]

Greedy Search: To evaluate the $Sapa^{PS}$ algorithm, we compare it with a greedy search strategy on the two domains discussed above. The greedy search algorithm that we implemented is called "static greedy" because it decides a static order of goals to achieve according to the decreasing order of goal utilities. Starting from the initial state, we try to first find a lowest cost plan that one goal at a time and try to find lowest cost plan that satisfies that single goal *and* does not delete any of the achieved goals. If there is no plan found that has less cost than the utility of the goal that it achieves, then we will skip that goal and move on to the next highest utility goal. This process continues until there is no goal left.

Figure 5 shows the comparisons between the $Sapa^{PS}$ algorithm discussed in this chapter and the greedy search algorithm discussed above for the first 15 problems of the two *ZenoTravel* (*Time* setting) and *Satellite* (*Complex* setting) domains. All results were collected using a Pentium IV 1.4GHz machine with running time limit of 1200 seconds. Both domains and problem sets are taken from the third IPC, and the procedure to make the PSP variations out of them was described above. The results show that in most cases, $Sapa^{PS}$ produces plans which have significantly higher total benefit values than the greedy search. In the ZenoTravel domain, the average quality improvement over problems that both approaches can find beneficial solutions is 1.8x. Moreover, there are 5 instances for which $Sapa^{PS}$ can find beneficial solutions while the greedy approach can not find any. Thus, in those problems, there seem to be no beneficial plan that achieves a single goal but only plans that sharing actions to achieve multiple goals can be beneficial. For the *Satellite* domain, the average improvement is 6.1x. The size of the plans in the two domains are quite large, with the largest plans found in the *Satellite* domain containing around 300 actions. Figure 6 shows the running time comparison (in milliseconds) between

[5] For example, the extended cost field in PDDL2.1 for the "BOARD" action in the ZenoTravel domain could be: `(+ (* (boarding-time ?c) (labor-cost ?c)) (airport-fee ?c))` (where ?C is the city).

[6] Since many benchmark problems start out with very few goals, we increase their difficulty by augmenting them with additional goals. For example, if there is a person that is declared in the initial state of ZenoTravel problem but his/her location is not specified as one of the goals, then we will add a goal with the person's final location randomly chosen.

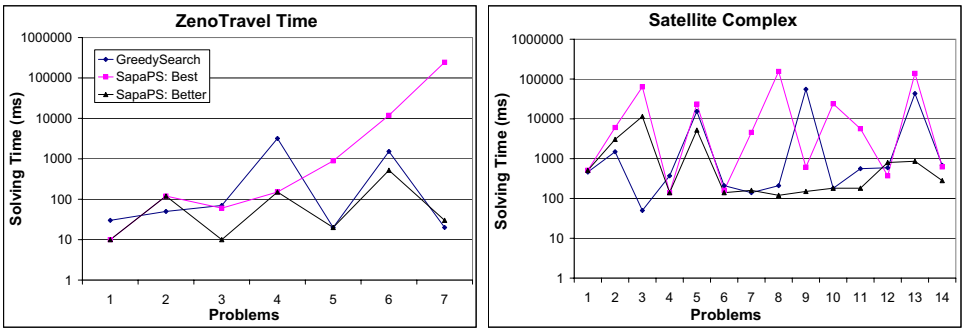

Figure 6: Running time comparison between $Sapa^{PS}$ and greedy search.

$Sapa^{PS}$ and the greedy search algorithm for the problems solved by both planners. Given that $Sapa^{PS}$ runs in *anytime* fashion and returns multiple solutions, we show both the time needed for $Sapa^{PS}$ to find the best solution and the time it needs to find the solution with equal or higher quality than the solution returned by the greedy search algorithm. The results show that while $Sapa^{PS}$ in general takes more time than the greedy search algorithm to find its best solution, it takes less time than the greedy search to find a solution of equal or better quality. This happens because the greedy search algorithm orders goals according to the decreasing order of utility values and thus can be forced to find a plan for a hard to satisfy goal, while $Sapa^{PS}$ is more flexible on the goal set and may find beneficial solutions for a set of easy to achieve goals very fast.

Although space restrictions preclude their inclusion, we have also compared $Sapa^{PS}$ to two other recent over-subscription planners *OptiPlan* and $AltAlt^{PS}$. Since these other planners are unable to handle temporal planning problems, our comparison was done in terms of the classical problem sets generated by van den Briel et. al. [van den Briel et. al., 2004]. Results reported there show that $Sapa^{PS}$, while slightly slower than $AltAlt^{PS}$ in general, is able to generate a higher number of better quality solutions.

5 Related Work

Two recent approaches to PSP planning problems work by selecting a subset of goals and using normal planning techniques to support them [Smith, 2004; van den Briel et. al., 2004]. The specific techniques used by these planners to select the subgoals differ. The advantage of these approaches 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 all the goals. The disadvantage of this type of approach is that if the heuristics do not select the right set of goals, then we can not switch to another subset during search. Moreover, if there is an unachievable goal selected, then the planner will return failure. In contrast $Sapa^{PS}$ avoids relying on any pre-defined subset of goals. It lets the A* framework decide which goals are the most beneficial for a given node during search. Therefore, it can partially correct the mistakes in heuristically selecting subset of goals at each search node as we go deeper in the search tree. $Sapa^{PS}$ also works in an *anytime* fashion and keeps on improving its solution quality given more search time. Nevertheless, the two types of planners can complement each other. Specifically, as mentioned earlier, we could use the $AltAlt^{PS}$ technique of interleaving subgoal selection and relaxed plan computation for our heuristic estimation (rather than a 2-phase construction we use).

One way of solving the PSP problems is to model them directly as deterministic MDPs [Boutilier et. al., 1999], where actions have different costs. The optimal solution to the PSP problem can then be extracted from the

optimal policy of this MDP. In fact, $Sapa^{PS}$ can be seen as an efficient way of directly computing the plan without computing the entire policy. Our preliminary experiments with a state of the art MDP solver show that while direct MDP approaches can guarantee optimality, they scale very poorly in practice and are unable to solve even small problems in our test suites.

6 Conclusion and Future work

In this paper, we described a way of modeling the partial-satisfaction planning problem as heuristic search. We discussed effective techniques for estimating the heuristic values by refining the relaxed plan extracted from the cost-sensitive planning graph. We implemented this approach on top of the *Sapa* planner and showed that the resulting planner, $Sapa^{PS}$, can effectively find high-quality solutions in reasonable time. Although we focused on the case where all goals are "soft constraints," it is straightforward to extend our model to handle a mix of hard and soft goals[7]. We are also working on improving the informedness of $Sapa^{PS}$ heuristics, and extending the framework to handle numeric and deadline goals in the PSP framework.

References

[Bonet & Geffner, 1997] Bonet, B., Loerincs, G., and Geffner, H. 1997. A robust and fast action selection mechanism for planning. *Proc AAAI-97*

[Boutilier et. al., 1999] Boutilier, C., Dean, T., and Hanks, S. 1999. Decision-Theoretic Planning: Structural Assumptions and Computational Leverage. In *Journal of Artificial Intelligence Research (JAIR), 11 (p.1-94)*

[Do & Kambhampati, 2003] Do, M. and Kambhampati, S. 2003. Sapa: a multi-objective metric temporal planer. In *JAIR 20 (p.155-194)*

[Edelkamp, 2003] Edelkamp, S. 2003. Taming numbers and durations in the model checking integrated planning system. In *JAIR 20 (p.195-238)*

[Hoffmann & Nebel, 2001] Hoffmann, J. and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. In *JAIR 14 (p.253-302)*.

[Kramer & Giuliano, 1997] Kramer, L. and Giuliano, M. 1997. Reasoning About and Scheduling Linked HST Observations with Spike. In *Proc. of Int. Workshop on Planning and Scheduling for Space*.

[Kramer, 2000] Kramer, L. 2000. Generating a Long Range Plan for a new Class of Astronomical Observatories. In *Proc. of 2nd NASA Workshop on Planning and Scheduling for Space*.

[Long & Fox, 2003] Long, D. and Fox, M. 2003. The 3rd International Planning Competition: Results and Analysis. In *JAIR 20 (p.1-59)*.

[Nguyen et. al., 2001] Nguyen, X., Kambhampati, S., and Nigenda, R. 2001. Planning Graph as the Basis for deriving Heuristics for Plan Synthesis by State Space and CSP Search. In *AIJ 135 (p.73-123)*.

[Potter & Gasch, 1998] Potter, W. and Gasch, J. 1998. A Photo Album of Earth: Scheduling Landsat 7 Mission Daily Activities. In *Proc. of SpaceOps.*

[Smith & Weld, 1999] Smith, D. and Weld, D. 1999. Temporal planning with mutual exclusion reasoning. In *Proc. of IJCAI-99*.

[Smith, 2003] Smith, D. 2003. The Mystery Talk. *Plannet Summer School*

[Smith, 2004] Smith, D. 2004. Choosing Objectives in Over-Subscription Planning. To appear in *ICAPS-04*.

[van den Briel et. al., 2004] van den Briel, M., Nigenda, R.,Do, M. and Kambhampati, S.: Effective Approaches for Partial Satisfaction (Over-Subscription) Planning. AAAI 2004: 562-569

[7] The best beneficial plan may have negative net benefit (since we no longer can assume empty plan is a legal solution); and the second "minimization" scan of the heuristic must only focus on the soft goals.