Effective Approaches for Partial Satisfaction (Over-Subscription) Planning

Menkes van den Briel, Romeo Sanchez, Minh B. Do, and Subbarao Kambhampati*
Department of Computer Science and Engineering
Arizona State University, Tempe AZ, 85287-5406
{menkes, rsanchez, binhminh, rao}@asu.edu

Abstract

In many real world planning scenarios, agents often do not have enough resources to achieve all of their goals. Consequently, they are forced to find plans that satisfy only a subset of the goals. Solving such partial satisfaction planning (PSP) problems poses several challenges, including an increased emphasis on modeling 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 paper, we start by describing a spectrum of PSP problems and focus on one of the more general PSP problems, termed PSP NET BENEFIT. We develop three techniques, (i) one based on integer programming, called *OptiPlan*, (ii) the second based on regression planning with reachability heuristics, called *AltAlt*^{ps} , and (iii) the third based on anytime heuristic search for a forward state-space heuristic planner, called *Sapa*^{ps} . Our empirical studies with these planners show that the heuristic planners generate plans that are comparable to the quality of plans generated by *OptiPlan*, while incurring only a small fraction of the cost.

Introduction

In classical planning the aim 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. In many real world scenarios, however, the agent may only be able to satisfy a subset of the goals. The need for such partial satisfaction might arise in some cases because the set of goal conjuncts may contain logically conflicting fluents, and in other cases there might just not be enough time or resources to achieve all of the goal conjuncts. Effective handling of partial satisfaction planning (PSP) problems poses several challenges, including an added emphasis on the need to differentiate between feasible and optimal plans. Indeed, for many classes of PSP problems, a trivially feasible, but decidedly non-optimal solution would be the "null" plan.

Despite the ubiquity of PSP problems, surprisingly little attention has been paid to the development of effective approaches in solving them. In this paper, we provide a systematic analysis of PSP problems. We will start by distinguishing several classes of PSP problems, and then focus on one of the more general classes, PSP NET BENEFIT. In this class, each goal conjunct has a fixed utility assigned 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* (cumulative utility minus cumulative cost).

We investigate three customized algorithms for solving PSP NET BENEFIT. The first, called "*OptiPlan*", solves the problem by encoding it as an integer program (IP). *OptiPlan* builds on the work of solving planning problems through IP (Vossen *et al*, 1999), and uses a more involved objective function that directly captures the net benefit. The second and third approaches, called "*AltAlt*^{ps} " and "*Sapa*^{ps} ", model PSP in terms of heuristic search with cost-sensitive reachability heuristics. *AltAlt*^{ps} builds on the AltAlt family of planners (Nguyen *et al*, 2001; Sanchez & Kambhampati 2003) that derive reachability heuristics from planning graphs. The main extension in *AltAlt*^{ps} involves a novel approach for heuristically selecting upfront a subset of goal conjuncts that is likely to be most useful. Once a subset of goal conjuncts is selected, they are solved by a regression search planner with cost sensitive heuristics. *Sapa*^{ps} is an extension of the forward state-space planner *Sapa* (Do & Kambhampati 2003). Unlike *AltAlt*^{ps} , *Sapa*^{ps} does not select a subset of goals up front but uses an anytime heuristic search framework in which goals are treated as "soft constraints". Any executable plan is considered a potential solution, with the quality of the plan measured in terms of its net benefit. The objective of the search is to find the plan with the highest net benefit. *Sapa*^{ps} uses novel ways of estimating the g and h values of partial solutions, and uses them to guide an anytime A* search.

OptiPlan generates plans that are optimal for a given plan length. *Sapa*^{ps} and *AltAlt*^{ps} , while capable of generating globally optimal plans[1], focus on effective but inadmissible

*This paper is the result of merging two separate efforts–one on *OptiPlan* and *AltAlt*^{ps} , and the other on *Sapa*^{ps} . Authors names are listed in the reverse order of seniority. We thank Will Cushing, David Smith and the AAAI reviewers 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.

[1] In theory, both *AltAlt*^{ps} and *Sapa*^{ps} can be made to generate

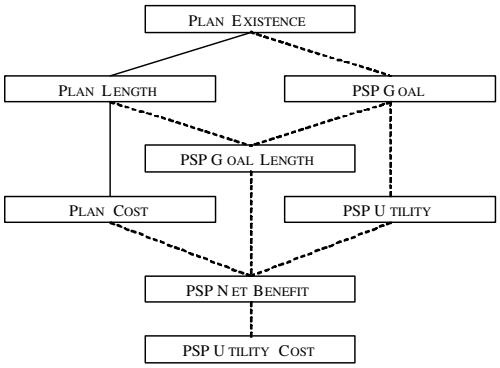


Figure 1: Hierarchical overview of several types of complete and partial satisfaction planning problems

heuristics for efficiency. Our empirical studies with these planners demonstrate that the heuristic planners *AltAlt*^{ps} and *Sapa*^{ps} can generate plans that are comparable to the quality of plans generated by *OptiPlan*, while incurring only a small fraction of the cost. The rest of this paper is organized as follows. In the next section, we give a taxonomy of PSP problems and discuss their complexity. In the following section, we describe how PSP problems can be modeled in *OptiPlan* through a more involved objective function. The next part of the paper describes the heuristic approaches for the PSP problem. We start with a discussion of cost-sensitive reachability heuristics, and then describe how they are used in qualitatively different ways in *AltAlt*^{ps} and *Sapa*^{ps} . We then present an empirical study that compares the effectiveness of the various approaches. We will end with a discussion of the related work and conclusions.

Definition and complexity

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 $P = (F, A, I, G)$. Having defined a planning problem we can now describe the following classical planning decision problems.

The problems of PLAN EXISTENCE and PLAN LENGTH represent the decision problems of plan existence and bounded plan existence respectively. They are probably the most common planning problems studied in the literature. We could say that PLAN EXISTENCE is the problem of deciding whether there exists a sequence of actions that transforms I into G, and PLAN LENGTH is the decision problem that corresponds to the optimization problem of finding a minimum sequence of action that transforms I into G.

The PSP counterparts of PLAN EXISTENCE and PLAN LENGTH are PSP GOAL and PSP GOAL LENGTH respectively. Both of these decision problems require a minimum number of goals that need to be satisfied for plan success.

globally optimal plans. In the case of *Sapa*^{ps} , this involves using admissible heuristics, while for *AltAlt*^{ps} , we need to do both an exhaustive search over subgoal sets, and use admissible heuristics during search.

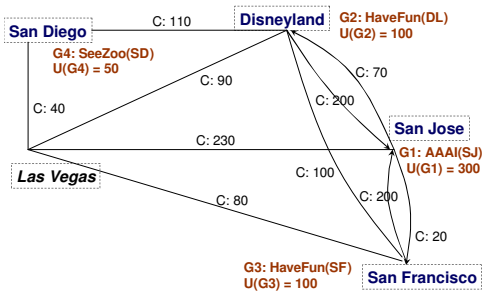


Figure 2: The travel example

Figure 1 gives a taxonomic overview of several types of complete (planning problems that require all goals to be satisfied) and partial satisfaction problems, with the most general problems listed below. Complete satisfaction problems are identified by names starting with PLAN and partial satisfaction problems have names starting with PSP.

Some of the problems given in Figure 1 involve action costs and/or goal utilities. Basically, PLAN COST corresponds to the optimization problem of finding minimum cost plans, and PSP UTILITY corresponds to the optimization problem of finding plans that achieve maximum utility. The problems of PSP NET BENEFIT is a combination of PLAN COST and PSP UTILITY, and PSP UTILITY COST is a generalization of PSP NET BENEFIT. Here we will formally define the decision problem of PSP NET BENEFIT and analyze its complexity. The corresponding optimization problem of finding a plan with maximum net benefit is the focus of this paper.

Definition 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: Figure 2 illustrates a simple example in which a student living in Las Vegas (LV) needs to go to San Jose (SJ) to present a AAAI paper. The cost of travelling is $C_{travel(LV,SJ)} = 230$. We assume that if the student arrives at San Jose, he automatically achieves the goal $g_1 = Attended\_AAAI$ with utility $U_{g_1} = 300$. The student also wants to go to Disneyland (DL) and San Francisco (SF) to have some fun ($g_2 = HaveFun(DL)$, $g_3 = HaveFun(SF)$) and to San Diego (SD) to see the zoo ($g_4 = SeeZoo(SD)$). The utilities for having fun in these places (U_{g_2}, U_{g_3}, U_{g_4}), and the travel cost of going from one place to the other are given in Figure 2. The goal of the student is to find a travel plan that gives him the best cost-utility tradeoff. In this example, the best plan is $P = \{travel(LV, DL), travel(DL, SJ), travel(SJ, SF)\}$ which achieves the goals g_1, g_2 and g_3, and ignores g_4.

Theorem 1 PSP NET BENEFIT *is PSPACE-complete.*

Proof We will show that PSP NET BENEFIT is in PSPACE and we will polynomially transform it to PLAN EXISTENCE,

which is a PSPACE-hard problem (Bylander 1994).

PSP NET BENEFIT is in PSPACE follows from Bylander (1994). PSP NET BENEFIT is PSPACE-hard because we can restrict it to PLAN EXISTENCE by allowing only instances having $U_f = 0, \forall f \in F$, $C_a = 1, \forall a \in A$, and $k = -2^m$. This restriction obtains $\sum_{a \in \Delta} C_a \leq 2^m$, which is the condition for PLAN EXISTENCE. ■

Given that PLAN EXISTENCE and PSP NET BENEFIT are PSPACE-hard problems, it should be clear that the other problems given in Figure 1 also fall in this complexity class. PSP NET BENEFIT does, however, foreground the need to handle plan quality issues.

OptiPlan: An Integer Programming Approach

OptiPlan is a planning system that provides an extension to the state change integer programming (IP) model by (Vossen *et al*, 1999). The original state change model uses the complete set of ground actions and fluents; *OptiPlan* on the other hand eliminates many unnecessary variables simply by using Graphplan (Blum & Furst 1997). In addition, *OptiPlan* has the ability to read in PDDL files. In this respect, *OptiPlan* is very similar to the BlackBox (Kautz & Selman 1999) and GP-CSP (Do & Kambhampati 2001) planners but instead of using a SAT or CSP formulation, the planner uses an IP formulation.

The state change formulation is built around the state change variables $x_{f,l}^{add}$, $x_{f,l}^{pre-add}$, $x_{f,l}^{pre-del}$, and $x_{f,l}^{maintain}$. These variables are defined in order to express the possible state changes of a fluent, with $x_{f,l}^{maintain}$ representing the propagation of a fluent f at level l. Besides the state change variables the IP model contains variables for actions, with $y_{a,l} = 1$ if and only if action a is executed in level l.

It is quite straightforward to model PSP NET BENEFIT in *OptiPlan*, all we do is transfer goal satisfaction from the hard constraints to the objective function. In the case of maximizing net benefit the objective becomes:

$$\sum_{f \in G} U_f (x_{f,n}^{add} + x_{f,n}^{pre-add} + x_{f,n}^{maintain}) - \sum_{l \in L} \sum_{a \in A} C_a y_{a,l} \quad (1)$$

where $L = 1, ..., n$ is the set of plan step levels, A is the set of actions, and G the set of goal fluents.

OptiPlan will find optimal solutions for a given parallel length l, however, the global optimum may not be detected as there might be solutions of better quality at higher values of l.

Heuristic Approaches: Preliminaries

In this section we describe *AltAlt*^{ps} and *Sapa*^{ps}, the two heuristic search planners capable of handling PSP problems. Given that the quality of the plan for PSP problem depends on both the utility of the goals achieved and the cost to achieve them, these planners need 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. In *AltAlt*^{ps} and *Sapa*^{ps}, this is done using the planning graph structure. In this section, we will first describe the cost propagation procedure over the planning graph as a basis for heuristic estimation in *AltAlt*^{ps} and *Sapa*^{ps}. In subsequent sections, we discuss how *AltAlt*^{ps} and *Sapa*^{ps} use this information.

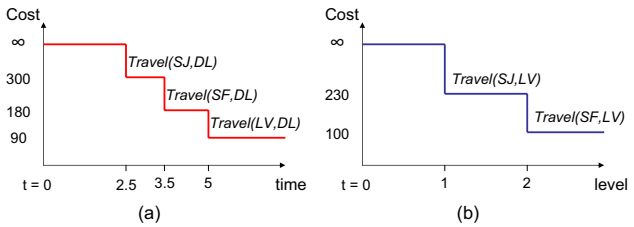


Figure 3: Cost function of goal $At(DL)$

Cost-propagation to Estimate the Goal Achievement Costs

To estimate the overall benefit of achieving the goals, we can use the planning graph structure to propagate the cost of facts from the initial state through applicable actions until we can estimate the lowest cost to achieve the goals. Following (Do & Kambhampati 2003), we use cost functions to capture the way cost of achievement changes as the graph is expanded. In the following, we briefly review the procedure. (Note that the discussion below is done in the more general context of temporal planning; to apply it to classical planning scenarios, we need only assume that all actions have uniform durations).

The purpose of the cost-propagation process is to build the cost functions $C(f, t_f)$ and $C(a, t_a)$ that estimate the cheapest cost to achieve fluent f at time (level) t_f and the cost to execute action a at level t_a. At the beginning ($t = 0$), let S_{init} be the initial state and C_a be the cost of action a then[2]: $C(f, 0) = 0$ if $f \in S_{init}$, $C(f, 0) = \infty$ otherwise; $\forall a \in A : C(a, 0) = \infty$. The propagation rules are as follows:

- $C(f, t) = min\{C(a, t - Dur_a) + C_a) : f \in Eff(a)\}$
- Max-prop: $C(a, t) = max\{C(f, t) : f \in Prec(a)\}$
- Sum-prop: $C(a, t) = \Sigma\{C(f, t) : f \in Prec(a)\}$

The max-propagation rule will lead to an admissible heuristic, while the sum-propagation rule does not. In our travel example, 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. Figure 3(a) shows the cost function for goal $g_2 = At(DL)$, which indicates that the earliest time to achieve g_2 is at $t = 2.5$ 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:

[2] C_a and $C(a, t)$ are different. If $a = Fly(SD, DL)$ then C_a is the airfare cost and $C(a, t)$ is the cost to achieve preconditions of a at t, which is the cost incurred to be at SD at t.

$LV \rightarrow SF \rightarrow DL$) and again at $t = 5.0$ to 90 (direct path: $LV \rightarrow DL$). For the levelled planning graph, where actions are non-durative, Figure 3(b) shows the cost function for the fact $At(LV)$ assuming that the student is at SJ in the initial state. At level 1, she can be at LV by going directly from SJ with cost 230. Then at level 2 she can at be LV with cost 100, using route $SJ \rightarrow SF \rightarrow LV$.

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 *et al*, 2001).[3]

Cost-sensitive heuristics

After building the planning graph with cost information, both *AltAlt*^{ps} and *Sapa*^{ps} use variations of the relaxed plan extraction process (Hoffmann & Nebel 2001; Nguyen *et al*, 2001) 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 both *AltAlt*^{ps} and *Sapa*^{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 $C(g,t)$) 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.

AltAlt^{ps} : Heuristic Search and Goal Selection

AltAlt^{ps} is a heuristic regression planner that can be seen as a variant of *AltAlt* (Nguyen *et al*, 2001) equipped with cost sensitive heuristics using *Max-prop* rules (see previous section). 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 the goal subset up front. The approach is sophisticated in the sense that it considers 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 (see below). Once a subset of goal conjuncts is selected, *AltAlt*^{ps} finds a plan that achieves such subset using its regression search engine augmented with cost sensitive heuristics. The goal set selection algorithm is described in more detail below.

[3]Stopping the cost propagation when the graph levels-off (i.e. no new facts or actions can be introduced into the graph) does not guarantee that the cost-functions are stabilized. Actions introduced in the last level still can reduce the cost of some facts and lead to the *re-activation* chain reaction process that reduce the costs of other propositions.

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 4: Goal set selection algorithm.

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 4 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 *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

[4]$getBestBenefitialGoal(G)$ returns the subgoal with the best benefit, $U_g - C(g,t)$ tradeoff

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'.

In our running example the original subgoals are $\{g_1 = AttendedAAAI,\ g_2 = HaveFun(DL),\ g_3 = HaveFun(SF),\ g_4 = SeeZoo(SD)\}$, with final costs $C(g,t) = \{230, 90, 80, 40\}$ and utilities U $= \{300, 100, 100, 50\}$ respectively. Following our algorithm, our starting goal g would be g_1 because it returns the biggest benefit (e.g. 300 - 230). Then, G' is set to g_1, and its initial relaxed plan R^*_P is computed. Assume that the initial relaxed plan found is $R^*_P = \{travel(LV, DL), travel(DL, SJ)\}$. We proceed to compute the best net benefit using R^*_P , which in our example would be $B^*_{MAX} = 300 - (200 + 90) = 10$. Having found our initial values, we continue iterating on the remaining goals $G = \{g_2, g_3, g_4\}$. On the first iteration we compute three different set of values, they are: (i) $G_{P_1} = \{g_1 \cup g_2\}$, $R_{P_1} = \{travel(LV, DL), travel(DL, SJ))\}$, and $B_{g_{P_1}} = 110$; (ii) $G_{P_2} = \{g_1 \cup g_3\}$, $R_{P_2} = \{travel(LV, DL), travel(DL, SJ), travel(LV, SF)\}$, and $B_{g_{P_2}} = 30$; and (iii) $G_{P_3} = \{g_1 \cup g_4\}$, $R_{P_3} = \{travel(LV, DL), travel(DL, SJ), travel(LV, SD)\}$, and $B_{g_{P_3}} = 20$. Notice then that our net benefit B^*_{MAX} could be improved most if we consider goal g_2. So, we update $G' = g_1 \cup g_2$, $R^*_P = R_{P_1}$, and $B^*_{MAX} = 110$. The procedure keeps iterating until we consider goal g_4, which decreases the net benefit. The procedure returns $G' = \{g_1, g_2, g_3\}$ as our goal set. In this example, there is also a plan that achieves the four goals with a positive benefit, but it is not as good as the plan that achieves the selected G'.

Sapa^{ps} : Heuristic Search using Goals as Soft Constraints

The advantage of the *AltAlt*^{ps} approach 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 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. In this section, we discuss an alternative method which models the top-level goals as "soft constraints." All executable plans are considered potential solutions, and the quality of a plan is measured in terms of its net benefit. The objective of the search is then to find a plan with the highest net benefit. To model this search problem in an A* framework, we need to first define the g and h values of a partial plan. The g value will need to capture the current net benefit of the plan, while the h value needs to estimate the net benefit that can be potentially accrued by extending the plan to cover more goals. Once we have these definitions, we need methods for efficiently estimating the the h value. We will detail this process in he context of *Sapa*^{ps} , which does forward (progression) search in the space of states.

g value: In forward planners, applicable actions are executed in the current state to generate new states. For a given state S, let partial plan $P_P(S)$ be the plan leading from the initial state S_{init} to S, and the goal set $G(S)$ be the set of goals accomplished in S. The overall quality of the state S depends on the total utility of the goals in $G(S)$ and the costs of actions in $P_P(S)$. The g value is thus defined as :

$$g(S) = \mathrm{U}(G(S)) - \mathrm{C}(P_P(S))$$

Where $\mathrm{U}(G(S)) = \sum_{g \in G(S)} \mathrm{U}_g$ is the total utility of the goals in $G(S)$, and $\mathrm{C}(P_P(S)) = \sum_{a \in P_P(S)} \mathrm{C}_a$ is the total cost of actions in $P_P(S)$. In our ongoing example, at the initial state $S_{init} = \{at(LV)\}$. Applying action $a_1 = travel(LV, DL)$ would leads to the state $S_1 = \{at(DL), g_2\}$ and applying action $a_2 = travel(DL, SF)$ to S_1 would lead to state $S_2 = \{at(SF), g_2, g_3\}$. Thus, for state S_2, the total utility and cost values are: $\mathrm{U}(G(S_2)) = \mathrm{U}_{g_2} + \mathrm{U}_{g_3} = 100 + 100 = 200$, and $\mathrm{C}(P_P(S_2)) = \mathrm{C}_{a_1} + \mathrm{C}_{a_2} = 90 + 100 = 190$.

h value: The h value of a state S should estimate how much additional net benefit can be accrued by extending the partial plan P_S to achieve additional goals beyond $G(S)$. The perfect heuristic function h^* would give the maximum net benefit that can be accrued. Any h function that is an upper bound on h^* will be admissible. Notice that we are considering the maximizing variant of A*. Before we go about investigating efficient h functions, it is instructive to pin down the notion of h^* value of a state S.

For a given state S, let P_R be a plan segment that is applicable in S, and $S' = Apply(P_R, S)$ be the 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 plan P_R according to state S is defined as follows: $U(Apply(P_R, S) = U(G(S')) - U(G(S))$. For a given state S, the **best beneficial remaining plan** P^B_S is a plan applicable in S such that there is no other plan P applicable in S for which $U(Apply(P, S)) - C(P) > U(Apply(P^B_S, S)) - C(P^B_S)$. If we have P^B_S, then we can use it to define the $h^*(S)$ value as follows:

$$h^*(S) = U(Apply(P^B_S, S)) - C(P^B_S) \qquad (2)$$

In our ongoing example, from state S_1, the most beneficial plan turns out to be $P^B_{S_1} = \{travel(DL, SJ), travel(SJ, SF)\}$, and $U(Apply(P^B_{S_1}, S_1)) = U_{\{g_1, g_2, g_3\}} - U_{\{g_2\}} =$

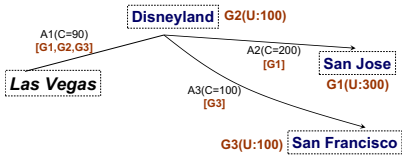


Figure 5: A relaxed plan and goals supported by each action.

$300 + 100 + 100 - 100 = 400$, $C(P^B_{S_1}) = 200 + 20 = 220$, and thus $h^*(S_1) = 400 - 220 = 180$.

Computing $h^*(S)$ value directly is impractical as searching for P^B_S is as hard as solving the PSP problem optimally. In the following, we will discuss a heuristic approach to approximate the h^* value of a given search node S by essentially approximating P^B_S using a relaxed plan from S.

Heuristic Estimation in *Sapa*^{ps}

Like *AltAlt*^{ps} , *Sapa*^{ps} also uses relaxed-plan heuristic extracted from the cost-sensitve planning graph (Do & Kambhampati 2003). However, unlike *AltAlt*^{ps} , in the current implementation of *Sapa*^{ps} we first build the relaxed plan supporting *all* the goals, then we use the second scan through the extracted relaxed plan to remove goals that are not beneficial, along with the actions that contribute solely to the achievement of those goals. For this purpose, we build the *supported-goals* list GS for each action a and fluent f starting from the top level goals as follows:

- $GS(a) = \bigcup GS(f) : f \in Eff(a)$
- $GS(f) = \bigcup GS(a) : f \in Prec(a)$

Assume that our algorithm extracts the relaxed plan $f = R_P(S_{init}) = \{a_1 : travel(LV, DL), a_2 : travel(DL, SJ), a_3 : travel(DL, SF)\}$ (shown in Figure 5 along with goals each action supports). For this relaxed plan, action a_2 and a_3 support only g_1 and g_3 so $GS(a_2) = \{g_1\}$ 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 $SA(S_G)$ 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 non beneficial 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 an effective but inadmissible h value.

Search in *Sapa*^{ps}

The complete search algorithm used by *Sapa*^{ps} is described in Figure 6. In this algorithm, search nodes are categorized as follows:

State Queue: *SQ*={S_{init}}
Best beneficial node: $N_B = \emptyset$
Best benefit: $B_B = 0$
while *SQ*≠{}
  S:= *Dequeue*(*SQ*)
  if $(g(S) > 0) \wedge (h(S) = 0)$ **then**
    Terminate Search;
  Nondeterministically select *a* applicable in *S*
    S' := *Apply*(*a*,*S*)
    if $g(S') > B_B$ **then**
      $Print\ BestBeneficialNode(S')$
      $N_B \leftarrow S'$; $B_B \leftarrow g(S')$
    if $f(S) \leq B_B$ **then**
      $Discard(S)$
    else *Enqueue*(*S'*,*SQ*)
end while;

Figure 6: Anytime A* search algorithm for PSP problems.

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, both nodes S_1, S_2 are beneficial nodes. If we decide to extend S_1 by applying the action $a_3 = travel(DL, LV)$ then we will get state $S_3 = \{at(LV), HaveFun(DL)\}$, which is not a beneficial node ($g(S_3) = U_{g_2} - C_{\{a_1,a_3\}} = 100 - 180 = -80$).

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

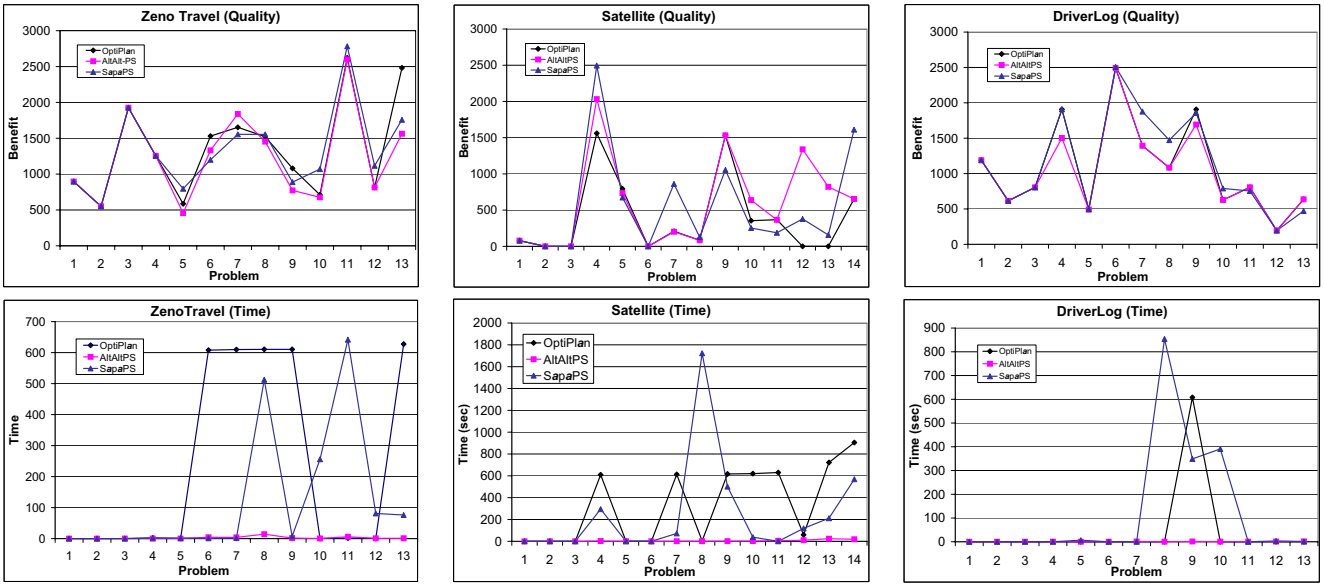
Termination node S_T is the *best* beneficial node in the queue. Moreover, because $h(S_T) = 0$, there is no benefit of extending S_T and therefore we can terminate the search at S_T. Notice that if the heuristic is *admissible*, then the set of actions leading to S_T represents an optimal solution for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

Unpromising Node: S is a unpromising node if $f(S) \leq B_B$ with B_B is the g value of a best beneficial state found so far.

The net benefit value of the best beneficial node (B_B) found during search can be used to set up the lower-bound value and thus nodes that have f values smaller than that lower-bound can be discarded.[5]

As described in Figure 6, 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 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 (until a terminal node is found), this is an *anytime algorithm*. The benefit of this approach is that the

[5]Being a *unpromising node* is not equal to not being a *beneficial node*. A given node S can have value $g(S) < 0$ but $f(S) = g(S) + h(S) > B_B$ and is still promising to be extended.



a) ZenoTravel domain b) Satellite domain c) DriverLog domain

Figure 7: Empirical evaluation

planner can return some plan with a positive net-benefit fast and keep on returning plans with better net benefit, if given more time to do more search.

Notice that the current heuristic used in *Sapa*^{ps} is not admissible, this is because: (i) a pruned unpromising nodes 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, another 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 $h(S) \leq 1/w * h^*(S)$ are pruned. For the second issue, we can continue the search for a better beneficial nodes after a termination node is found until some criteria are met (e.g. reached certain number of search node limit).

Empirical Evaluation

In the foregoing, we have described several qualitatively different approaches for solving the PSP problem. Our aim in this section is to get an empirical understanding of the cost-quality tradeoffs offered by this spectrum of methods.

Since there are no benchmark PSP problems, we used existing STRIPS planning domains from the last International Planning Competition (Long & Fox 2003). In particular, our experiments include the domains of Driverlog, Satellite, and Zenotravel. Utilities ranging from 100 to 600 were assigned to each of the goals, and costs ranging from 10 to 800 were assigned to each of the actions by taking into account some of the characteristics of the problem. For example, in the Driverlog domain, a driving action is assigned a higher cost than a load action. Under this design, the planners achieved around 60% of the goals on average over all the domains.

All three planners were run on a 2.67Ghz CPU machine with 1.0GB RAM. The IP encodings in *OptiPlan* were solved using ILOG CPLEX8.1 with default settings except that the start algorithm was set to the dual problem, the variable select rule was set to pseudo reduced cost, and a time limit of 600 seconds was imposed. In case that the time limit was reached, we denoted the best feasible solution as *OptiPlan*'s solution quality. Given that *OptiPlan* returns optimal solutions up to a certain level, we set the level limit for *OptiPlan* by post-processing the solutions of *AltAlt*^{ps} to get the parallel length using techniques discussed in (Sanchez & Kambhampati 2003).

Figure 7 shows the results from the three planners. It can be observed that the two heuristic planners, *AltAlt*^{ps} and *Sapa*^{ps} , produce plans that are comparable to *OptiPlan*. In some problems they even produce better plans than *OptiPlan*. This happens when *OptiPlan* reaches its time limit and can not complete its search, or when the level given to *OptiPlan* is not high enough. When comparing the two heuristic planners, *AltAlt*^{ps} is often faster, but *Sapa*^{ps} usually returns better quality plans. The decrease on the performance of *AltAlt*^{ps} could be due to the following reasons: either the greedy goal set selection procedure of *AltAlt*^{ps} picks a bad goal subset upfront, or its cost-sensitive search requires better heuristics. A deeper inspection of the problems in which the solutions of *AltAlt*^{ps} are suboptimal revealed that although both reasons play a role, the first reason dominates more often. In fact, we found that the goal set selection procedure tends to be a little bit conservative, selecting fewer subgoals in cases where the benefit gain is small, which means that our relaxed plan cost is overestimating the residual costs of the subgoals. To resolve this issue, we may need to account for subgoal interactions more aggressively in the goal set selection algorithm. The higher running time of *Sapa*^{ps} is mostly due to the fact that it tries to search for multiple (better) beneficial plans and thus have higher number of search nodes. In many cases, it takes very short

time to find the first few solutions but much more to improve the solution quality (even slightly). Moreover, the heuristic used in $Sapa^{ps}$ seems to be misleading in some cases (mostly in the Satellite domain) where the planner spends a lot of time switching between different search branches at the lower levels of the search tree before finding the promising one and go deeper.

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 the 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.

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 $AltAlt^{ps}$ is more sophisticated in that it considers the residual cost of a subgoal in the context of an exisiting partial plan for achieving other selected 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 and Future Work

In this paper, we investigated 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 presented a spectrum of PSP problems. We then focused on one general PSP problem, called PSP Net Benefit, and developed a spectrum of planners– *OptiPlan*, $AltAlt^{ps}$ and $Sapa^{ps}$ for it. Our empirical results show that the heuristic approaches are able to generate plans whose quality is comparable to the ones generated by the optimizing approach, while incurring only a fraction of the running time. The two types of heuristic planners can also complement each other. The heuristic techniques used in $AltAlt^{ps}$ can be employed in $Sapa^{ps}$ as an alternative to its current approach of heuristic evaluation of each search node. Our future work will extend our heuristic framework to more complex PSP problems. Of particular interest to us is handling partial satisfaction in metric temporal planning problems with resources. $Sapa^{ps}$, developed on top of a temporal planner, is a promising candidate for this extension.

References

Blum, A., and Furst, M. 1997. Fast planning through planning graph analysis. *Artificial Intelligence* 90:281–300.

Boutilier, C., Dean, T., and Hanks, S. 1999. Decision-Theoretic Planning: Structural assumptions and computational leverage. In *JAIR 11 (p.1-94)*

Bylander, T. 1994. The computational complexity of propositional STRIPS planning. In *AIJ* 69(1-2):165–204.

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

Do, M. and Kambhampati, S. 2001. Planning as Constraint Satisfaction: Solving the planning graph by compiling it into CSP. In *AIJ* 132(2):151–182.

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 *Proc. Int. Conf. on Space Mission Challenges for Infor. Tech.*.

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

Kautz, H., and Selman, B. 1999b. Blackbox: Unifying Sat-based and Graph-based planning. In *Proc. of IJCAI-99*.

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

Long, D., and Fox, M. 1999. Efficient implementation of the plan graph in STAN. In *JAIR* 10:87–115.

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

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, W. and Gasch, J. 1998. A photo album of Earth: Scheduling Landsat 7 mission daily activities. In *Proc. of SpaceOps.*

Sanchez, R., and Kambhampati, S. 2003. Altalt-p: Online parallelization of plans with heuristic state search. In *JAIR 19 (p.631–657)*.

Smith, D. 2003. The Mystery talk. *Planet Summer School*

Smith, D. 2004. Choosing objectives in Over-Subscription planning. In *Proc. of ICAPS-04*.

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

Williamson, M., and Hanks, S. 1994. Optimal planning with a Goal-Directed utility model. In *Proc. of AIPS-94*.