Over-Subscription in Planning: a Partial Satisfaction Problem

Menkes van den Briel, Romeo Sanchez, and Subbarao Kambhampati
Department of Computer Science and Engineering
Ira A. Fulton School of Engineering
Arizona State University, Tempe AZ, 85287-8809
{menkes, rsanchez, rao}@asu.edu

Abstract

In many real world planning scenarios, agents often do not have enough resources to achieve all of their goals. Hence, this requires finding plans that satisfy only a subset of the them. Solving such partial satisfaction planning (PSP) problems poses several challenges, including an increased emphasis on modelling and handling plan quality (in terms of action costs and goal utilities). Despite the ubiquity of such PSP problems, very little attention has been paid to them in the planning community. In this 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 two techniques, one based on integer programming, called OptiPlan, and the other based on regression planning with reachability heuristics, called *AltAlt*^{ps}. Our empirical studies with these two planners show that the heuristic planner *AltAlt*^{ps} generates plans that are quite close to the quality of plans generated by OptiPlan, while incurring only a small fraction of the cost. Finally, we also present interesting connections among our work and over-subscription scheduling.

Introduction

In classical planning the aim is to find a sequence of actions that transforms a given initial state $\mathcal{I}$ to some goal state $\mathcal{G}$, where $\mathcal{G} = g_1 \wedge g_2 \wedge ... \wedge g_n$ 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 $\mathcal{G}$ are achieved. This all-or-nothing criterion is quite rigorous, no distinction is being made between plans that achieve some of the conjuncts and plans that achieve none of them at all.

Often times, real world planning problems are over-subscribed in the sense that there is an excess of conjuncts that could possibly be achieved at the same time. Need for partial satisfaction might arise due to a variety of reasons. In some cases, 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 these problems in the planning community. In this paper, we provide a first systematic analysis of PSP problems. We will start by distinguishing several classes of PSP problems, but focus on one of the more 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).

Earlier work by the PYRRHUS system (Williamson & Hanks 1994) allows for partial satisfaction in planning problems with goal utilities. However, unlike the PSP problems discussed in this paper, PYRRHUS requires all goals to be achieved; partial satisfaction is interpreted by using a non-increasing utility function on each of the goals. Many NASA planning problems have been identified as partial satisfaction problems (Smith 2003). In (Smith 2004) a technique is presented for solving over-subscribed planning problems. This technique is based on abstracting the planning problem by using a plan graph to estimate the costs of achieving each goal. The abstracted version is then used to help the planner decide on what goals and actions should be considered and in what order they should be considered.

An important aim of our research is to get an empirical understanding of the cost-quality tradeoffs offered by optimal and heuristic approaches. To this end, we develop two different techniques for handling PSP NET BENEFIT problems–one, called "OptiPlan" which solves the problem by encoding it as an integer program; and the other called "*AltAlt*^{ps}" which is based on regression planning with cost-sensitive reachability heuristics. OptiPlan builds on earlier work on solving planning problems with IP encodings (Vossen *et al.* 1999), and models the net benefit optimization in terms of a more involved objective function. *AltAlt*^{ps} builds on the AltAlt family of planners (Nguyen, Kambhampati, & Sanchez 2002) which derive reachability heuristics from planning graphs. The main extensions in *AltAlt*^{ps} involve techniques for propagating cost information on planning graphs (Do & Kambhampati 2002), and a novel approach for heuristically selecting a subset of goal conjuncts. Selecting such a subset is a key issue in PSP, if there are n conjuncts then there are 2^n goal subsets. In order to avoid an exhaustive search, a

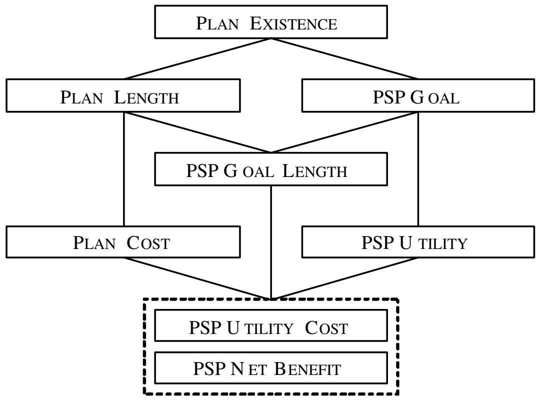


Figure 1: Hierarchical overview between several types of complete and partial satisfaction planning problems

good heuristic is needed that evaluates up front which subset of conjuncts is likely to be most useful. Once a subset of goal conjuncts is selected, it is solved by a regression search that has cost sensitive heuristics.

PSP have been studied in the scheduling community under the name of over-subscription scheduling. In general, an over-subscription scheduling problem is one where there are more tasks to be accomplished than there are time and resources available. Typical objectives are to maximize resource usage or to accommodate as many tasks as possible (Kramer & Smith 2003). Over-subscription scheduling problems are resource driven, while PSP planning problems are more goal driven. Over-subscribed scheduling problems have been solved by iterative repair (Kramer & Smith 2003), constructive approaches (Potter & Gasch 1998), greedy search approaches (Frank *et al.* 2001), and even genetic algorithms (Globus *et al.* 2003).

Our empirical studies show that the heuristic planner *AltAlt*^{ps} can generate plans that are quite close to the quality of plans generated by OptiPlan, while incurring only a small fraction of the cost. The empirical study also sheds light on possible improvements that can be made to *AltAlt*^{ps} to make it even more competitive. At the end of this paper we will try to identify similarities and differences between techniques for solving over-subscription scheduling problems and our approach to solving the PSP planning problem.

Definition and complexity

The following notation will be used: $\mathcal{F}$ is a finite set of fluents and $\mathcal{A}$ is a finite set of actions, where each action consists of a list of preconditions and a list of add and delete effects. $\mathcal{I} \subseteq \mathcal{F}$ is the set of fluents describing the initial state and $\mathcal{G} \subseteq \mathcal{F}$ is the set of goal conjunctions. Hence we define a planning problem as a tuple $\mathbf{P} = (\mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G})$. 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 $\mathcal{I}$ into $\mathcal{G}$, and PLAN LENGTH is the decision problem that corresponds to the optimization problem of finding a minimum sequence of action that transforms $\mathcal{I}$ into $\mathcal{G}$.

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 needs to be satisfied for plan success.

Figure 1 gives a hierarchical 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 cost 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 UTILITY COST and PSP NET BENEFIT are both combinations of PLAN COST and PSP UTILITY. Here we will formally define PSP NET BENEFIT since it is the focus of this paper.

Definition PSP NET BENEFIT: Given a planning problem $\mathbf{P} = (\mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G})$ 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 $\mathcal{I}$ leads to a state $\mathcal{S}$ that has net benefit $\sum_{f \in (S \cap G)} u_f - \sum_{a \in \Delta} c_a \geq k$?

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 NPSPACE because if a solution exists we can generate a solution by nondeterministically choosing a sequence of actions $\Delta = \langle a_1, ..., a_n \rangle$ that solves $\mathbf{P}$. Note that, the size of the states is independent of action costs and goal utilities, and since the size of each state is bounded by the number of (binary) fluents $m = |\mathcal{F}|$, the solution comprised by the minimum action sequence cannot be greater than 2^m. Hence, all action sequences greater than 2^m must contain duplicate states and therefore contain loops. The action sequence that is obtained by eliminating these loops will have a length that will be less then 2^m. Hence, no more than 2^m nondeterministic choices are needed. Since NPSPACE = PSPACE (Savitch 1970), PSP NET BENEFIT is in PSPACE.

PSP NET BENEFIT is PSPACE-hard because we can restrict to PLAN EXISTENCE by allowing only instances having $u_f = 0 \forall f \in \mathcal{F}$, $c_a = 1 \forall a \in \mathcal{A}$, 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.

OptiPlan: an IP based approach for solving PSP problems

OptiPlan is a planning system that provides an extension to the state change integer programming (IP) model by (Vossen et al. 1999). The model by Vossen et al. 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 reads PDDL input files. In this respect, OptiPlan is very similar to the BlackBox planner (Kautz & Selman 1999) but instead of using a SAT formulation the plan is encoded as an IP.

The state change formulation is built around the "state change" variables $x^{\text{add}}_{f,i}$, $x^{\text{pre-add}}_{f,i}$, $x^{\text{pre-del}}_{f,i}$, and $x^{\text{maintain}}_{f,i}$. These variables are defined in order to express the possible state changes of a fluent, with $x^{\text{maintain}}_{f,i}$ representing the propagation of a fluent f at period i. Besides the state change variables the IP model contains action variables $y_{a,i}$. Where $y_{a,i} = 1$ if and only if action a is executed in period i. The constraints of the state change model are as follows:

$$\sum_{a \in \text{add}_f/\text{pre}_f} y_{a,i} \geq x^{\text{add}}_{f,i} \tag{1}$$

$$y_{a,i} \leq x^{\text{add}}_{f,i} \quad \forall a \in \text{add}_f/\text{pre}_f \tag{2}$$

$$\sum_{a \in \text{pre}_f/\text{del}_f} y_{a,i} \geq x^{\text{pre-add}}_{f,i} \tag{3}$$

$$y_{a,i} \leq x^{\text{pre-add}}_{f,i} \quad \forall a \in \text{pre}_f/\text{del}_f \tag{4}$$

$$\sum_{a \in \text{pre}_f \cup \text{del}_f} y_{a,i} = x^{\text{pre-del}}_{f,i} \tag{5}$$

$$x^{\text{add}}_{f,i} + x^{\text{maintain}}_{f,i} + x^{\text{pre-del}}_{f,i} \leq 1 \tag{6}$$

$$x^{\text{pre-add}}_{f,i} + x^{\text{maintain}}_{f,i} + x^{\text{pre-del}}_{f,i} \leq 1 \tag{7}$$

$$x^{\text{pre-add}}_{f,i} + x^{\text{maintain}}_{f,i} + x^{\text{pre-del}}_{f,i} \leq x^{\text{pre-add}}_{f,i-1} + x^{\text{add}}_{f,i-1} + x^{\text{maintain}}_{f,i-1} \tag{8}$$

$$x^{\text{add}}_{f,0} = \begin{cases} 1 & \text{if } f \in \mathcal{I} \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

$$x^{\text{pre-add}}_{f,t} + x^{\text{add}}_{f,t} + x^{\text{maintain}}_{f,t} \geq 1 \quad \forall f \in \mathcal{G} \tag{10}$$

Where constraints (1) through (5) describe the logical interpretations between the action and state change variables for all $f \in \mathcal{F}, i \in 1, ..., t$. Constraints (6) and (7) make sure that for all $f \in \mathcal{F}, i \in 1, ..., t$ fluents can only be propagated at period i if and only if there is no action in period i that adds or deletes the fluent. Constraints (8) describe the backward chaining requirements for all $f \in \mathcal{F}, i \in 1, ..., t$ for the state change variables, and the initial and goal state constraints are represent by (9) and (10) respectively.

Since the constraints guarantee plan feasibility, the objective function of the IP model can take on any linear function. In the case of maximizing net benefit the objective becomes

$$\max z = \sum_{f \in G} u_f(x^{\text{add}}_{f,t} + x^{\text{pre-add}}_{f,t} + x^{\text{maintain}}_{f,t}) - \sum_{i \in T} \sum_{a \in A} c_a y_{a,i}, \tag{11}$$

where $T = 1, ..., t$ is the set of time periods, A is the set of actions, and G the set of goal fluents.

Note that the above formulation can easily be changed to handle any of the other partial satisfaction problems described in the previous section, it merely requires changing the objective function and sometimes adding or removing some constraints.

Although integer programming finds optimal solutions, the formulation that was used only finds optimal solutions for a given parallel length t. Hence the global optimum might not be detected, there could still be solutions of better quality at higher values of t.

AltAlt^{ps} : a heuristic approach for solving PSP problems

In this section we present *AltAlt^{ps}* , a variant of the heuristic state-space search planner *AltAlt* (Nguyen, Kambhampati, & Sanchez 2002). Both these planners are based on a combination of Graphplan (Blum & Furst 1997; Long & Fox 1999; Kautz & Selman 1999) and heuristic state-space search technology (Bonet, Loerincs, & Geffner 1997; Bonet & Geffner 1999; McDermott 1999). However, the design of *AltAlt^{ps}* involves some additional challenges. Specifically, the implementation of a cost propagation procedure during the planning graph construction phase. The propagated cost information will be used in two different ways: First, we use it to greedily select a subset of the top-level goals upfront, such that we can avoid the exponential branching factor of the search. Second, once the subgoals are selected, then we use the cost information to drive the search of *AltAlt^{ps}* in order to find a cost-sensitive plan.

In *AltAlt^{ps}* , the cost of a plan is computed in terms of the execution costs of its actions, and its heuristics have to estimate the cost of a state based on the individual cost of the propositions composing such state. However, only the execution costs of the actions are given to the planner, and the propositions in the initial state are known to have zero cost, but we still need to propagate the cost of every other single proposition, including the top level goals. We use the planning graph construction phase of *AltAlt^{ps}* to propagate the costs of the propositions using the execution costs of the actions that achieve them.

As mentioned earlier, the cost information will be used not only for heuristics to guide the search, but also for selecting the set of subgoals to work on. The basic idea of the goal set selection procedure is to construct the partial subgoal set iteratively by adding one goal at a time, ensuring that the cost of the added subgoal does not degrade our final net benefit. The cost of each subgoal is computed using a generalization of the relaxed plan heuristic idea (Nguyen & Kambhampati 2000; Nguyen, Kambhampati, & Sanchez 2002). The cost propagation procedure in the planning graph, as well as the goal set selection algorithm are described in more detail in the next subsections.

Cost propagation procedure and heuristics in *AltAlt*^{ps}

Initially, only the execution costs of the actions and the utilities of the top level goals are provided. Propositions that are in the initial state are assumed to have cost zero. Therefore, we need to propagate the cost of the rest of the propositions in the planning graph to estimate the cost of the goals.

Let $h_l(p)$ be the cost associated with an individual proposition p at level l of the planning graph. Propositions in the initial state I have cost zero, and ∞ otherwise. Thus, for each action a achieving p, the cost of p is propagated using the iterative planning graph construction procedure of *AltAlt*^{ps} , as follows:

$$h_l(p) = \begin{cases} 0 & \text{if } p \in I \\ min\{h_l(p), cost(a) + \mathcal{C}_l(a)\} & \\ \infty & \text{otherwise} \end{cases} \tag{12}$$

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

$$\mathcal{C}_l(a) = max\{h_{l-1}(q) : q \in Prec(a)\} \tag{13}$$

or under independence assumption:

$$\mathcal{C}_l(a) = \sum\{h_{l-1}(q) : q \in Prec(a)\} \tag{14}$$

Notice that we use the notion of level of the planning graph as a unit of time to propagate the costs of propositions. As a consequence, cost propagation will terminate as soon as the planning graph has been built. We then let the cost of a proposition be the final cost of that proposition in the final level of the planning graph. Similar cost propagation procedures are developed in the context of metric temporal planning (Do & Kambhampati 2002). From the cost propagation procedure described above, we can easily derive the first heuristic for our cost-based planning framework:

Max Cost Heuristic 1 $h_{maxC}(S) = \max_{p \in S} h_l(p)$

Although the h_{maxC} heuristic is admissible, it tends to be too conservative given that it considers the cost of state S be equal to the cost of one of the state's components. The idea behind the h_{maxC} heuristic is that by choosing the proposition with maximum cost we will know at least the minimum cost to solve the overall state.

We could try to improve the efficiency of our heuristic by considering the positive interactions among the subgoals. We adapt the "relaxed plan" heuristic idea from *AltAlt* (Nguyen, Kambhampati, & Sanchez 2002), with some important modifications. Specifically, we will not compute the relaxed plan length anymore, but the relaxed plan cost, where the cost of the relaxed plan is given by the execution costs of the actions in it. Furthermore, we will sort the supporting actions in increasing value of their accumulative costs given that we want to consider the cheapest ones first in building the relaxed plan R_p.

More formally speaking, let S be a state to regress. We consider a proposition $p \in S$ such that $h_l(p) = max_{p \in S} h_l(p)$ (hardest to support first). Having found the proposition p to regress, we sort the set of supporting actions A_p of p on increasing value of their accumulative costs. Where the accumulative cost of an action is defined conservatively as $AcumCost(a_p) = cost(a_p) + max_{q \in Prec(a_p)} h_l(q)$. So, we select the action a_p from A_p with the lowest $AcumCost(a_p)$. The rationale for this is that we want to consider not only actions with minimum cost but also actions that are likely going to introduce cheaper subgoals in the relaxed plans. By regressing S over action a_p we get the state $S' = S \cup Prec(a_p) \setminus Eff(a_p)$, obtaining the following recurrence relation:

$$relaxCost(S) = cost(a_p) + relaxCost(S') \tag{15}$$

This regression accounts for the positive interactions in the state S given that by subtracting the effects of a_p, any other proposition that is co-achieved when p is being supported is not counted in the cost computation. The recursive application of the last equation is bounded by the final level of the planning graph, and it will eventually reduce to a state S_0 where each proposition $q \in S_0$ is also in the initial state I, having cost $h_l(q) = 0$. Given the last equation, we are now ready to set up our next heuristic:

RelaxCost Heuristic 2 $h_{relaxC}(S) = relaxCost(S)$[1]

The application of the h_{relaxC} heuristic indirectly extracts a sequence of actions (the actions a_p selected at each reduction), which would have achieved the set S from the initial state I if there were no negative interactions. Such sequence of actions is our relaxed plan R_p. Notice that the value of the heuristic function is not more than the addition of the execution costs of the actions $a_p \in R_p$.

Goal set selection algorithm

The general idea of our algorithm is to incrementally extend a partial goal set G' using the utilities of the top level goals G and their costs of achievement. We want to consider only the goals that are likely going to increment our net benefit. A general description of our algorithm is given in Figure 2. First, the function *getBestBenefitialGoal(G)* returns the initial subgoal $g \in G$ with maximum remaining utility,[2] and we use it to initialize G'. If no subgoals can be chosen then we could easily conclude that there is no solution to our problem.

Then, we extract an initial relaxed plan R_p^* for G' using the procedure *extractRelaxPlan(G',$\emptyset$)*. Notice that two arguments are passed to the function, one is the current partial goal set from where the relaxed plan will be computed, and the second parameter is another relaxed plan that will be used as a guideline for computing the returning plan. At the beginning, no relaxed plan is provided. Notice that our

[1] The h_{relaxC} heuristic will be used only in the hardest problems in our empirical evaluation

[2] The remaining utility of a goal g is computed by $\mathcal{U}_g - h_t(g)$, where $\mathcal{U}_g$ is the utility of goal g and $h_t(g)$ is the subgoal cost at the final level t of the planning graph

```
Procedure partialize(G)
    g ← getBestBenefitialGoal(G);
    if(g = NULL)
        return Failure;
    G' ← {g};
    G ← G \ g;
    R*_p ← extractRelaxPlan(G', ∅)
    B*_MAX ← getUtil(G') − getCost(R*_p);
    B_MAX ← B*_MAX
    while(B_MAX > 0 ∧ G ≠ ∅)
        for(g ∈ G \ G')
            G_P ← G' ∪ g;
            R_p ← ExtractRelaxPlan(G_P, R*_p)
            B_g ← getUtil(G_P) − getCost(R_p);
            if(B_g > B*_MAX)
                g* ← g;
                B*_MAX ← B_g;
                R*_g ← R_p;
            else
                B_MAX ← B_g − B*_MAX
        end for
        if(g* ≠ NULL)
            G' ← G' ∪ g*;
            G ← G \ g*;
            R*_p ← R*_g;
            B_MAX ← B*_MAX
    end while
    return G';
End partialize;
```

Figure 2: Goal set selection algorithm.

greedy algorithm incrementally adds subgoals to our partial goal set G', in consequence, the relaxed plans among iterations (e.g. those for G' and $G' \cup g$) are very similar. By passing the previous computed relaxed plan to our procedure we avoid introducing unnecessary actions as well as recomputing the similarities among them.

Having found the initial partialized set G' and its correspondent 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'.[3] 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 our overall net benefit B^*_{MAX} increases. This can be seen in the main loop of our algorithm. Basically, we consider one goal at a time and compute its correspondent relaxed plan R_p and benefit B_g. Notice that we use now the relaxed plan R^*_p found in a previous iteration to compute R_p. If the benefit B_g of the current subgoal g is greater than our current best benefit B^*_{MAX} then we mark the current subgoal as a possible candidate g^* for G'. Notice that the same procedure is followed for the remaining subgoals, and it is not until we have considered all of them that we add the subgoal $g*$ with the best net benefit to our partial goal set G'. Then, we update our current best relaxed plan R^*_p and our new best benefit threshold B^*_{MAX}. This iterative procedure will end when we can not improve the net benefit any more or when there are no more subgoals to add, returning our partial goal set G'. Notice that the procedure above is greedy, so we are not immune from selecting a bad subset.

[3]The descriptions of the functions are $getUtil(G) = \sum_{g \in G} \mathcal{U}_g$ and $getCost(R_p) = \sum_{a \in R_p} cost(a_p)$

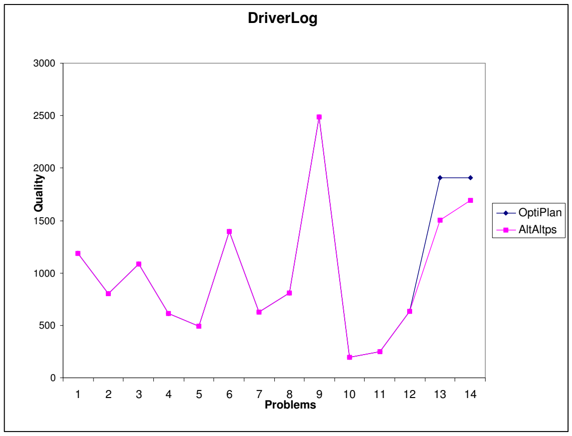
Example Let's assume that we have an initial set of goals $G = \{g_1, g_2, g_3, g_4\}$, and their final costs and utilities are respectively $h_l = \{40, 60, 30, 70\}$, and $\mathcal{U} = \{50, 30, 40, 100\}$. Following our algorithm, our starting goal would be g_4, because it is the one with the highest net benefit, so $G' = \{g_4\}$ and $G = \{g_1, g_2, g_3\}$. Assume that the relaxed plan found for g_4 is R_{p4} with cost of 70. Then, for each of the remaining subgoals in G we compute their benefits the following way: $B_g = getUtil(g_4 \cup g) - getCost(extractRelaxPlan(g_4 \cup g, R_{p4}))$ for each $g \in G$. Suppose that any of the subgoals adds more actions to the relaxed plan R_{p4} (they come for free), so we can assume the following benefits, $B_{g_1} = 150, B_{g_2} = 130$, and $B_{g_3} = 140$. We can conclude then that our best subgoal to add is g_1, so we set $G' = g_1 \cup g_4$. The relaxed plan also gets updated to $R_{p4\cup1}$, which this time no new action has been added. Notice that in most of the cases new actions could be added to the relaxed plans when we consider new subgoals, decreasing then their benefits, and in consequence pruning some of the top level goals when no benefit is found. The procedure gets called again, this time having two subgoals (g_1 and g_4) in our partial goal set G'.

Experimental Results

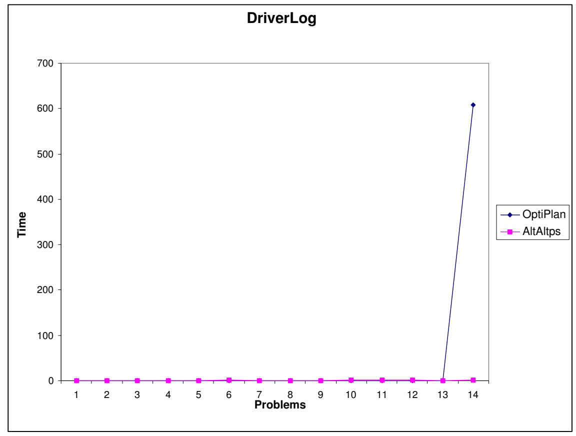
It is not that straightforward to obtain a clear comparison between *AltAlt^{ps}* and OptiPlan. For instance, OptiPlan searches in the planning graph and gives optimal solutions for the level at which the planning graph was grown. *AltAlt^{ps}* on the other hand, searches in the space of states and uses the planning graph only to derive efficient heuristics, hence the solution obtained by *AltAlt^{ps}* is not directly related to a particular planning graph level. To overcome this discrepancy, we post-process the solution of *AltAlt^{ps}* by using techniques introduced in (Sanchez & Kambhampati 2003). We compare *AltAlt^{ps}* with OptiPlan by running *AltAlt^{ps}* first, then post-process its plan to obtain a parallel plan, and then use the parallel length of this plan as the input for OptiPlan. (We could have given the serial length of *AltAlt^{ps}* as input level for OptiPlan, but this would have caused a blowup in the IP encoding size.)

The IP encodings in OptiPlan were solved using ILOG CPLEX8.1, a commercial linear programming solver. Default settings were used except the start algorithm was set to the dual problem, the variable select rule was set to pseudo reduced cost, and a 600 seconds limit was set to the CPLEX solver. Both OptiPlan and *AltAlt^{ps}* ran the problems on a 2.67Ghz CPU with 1.0GB RAM.

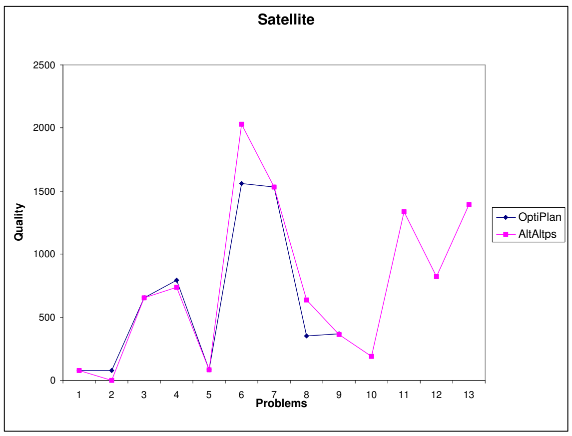
One problem that we faced is that there are no benchmark PSP problems. Therefore, we used existing STRIPS planning domains from the last International Planning Competition (Long & Fox 2002), and randomly generated costs and utilities for each particular planning problem. However,



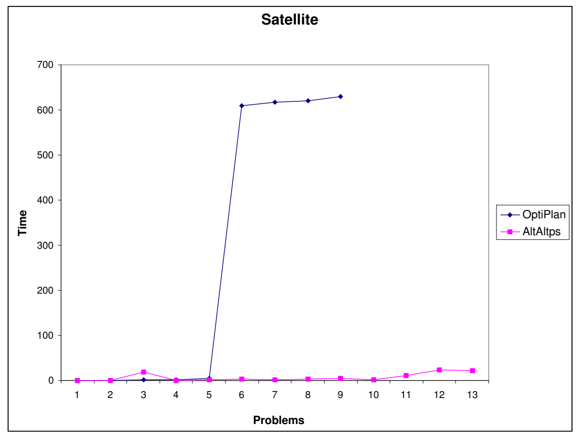
a) DriverLog Solution quality



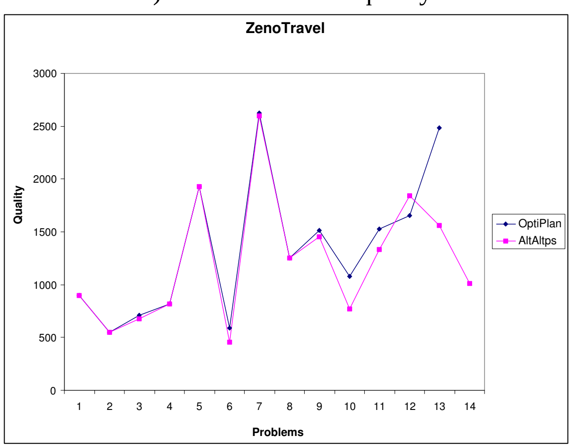
b) DriverLog Solution time



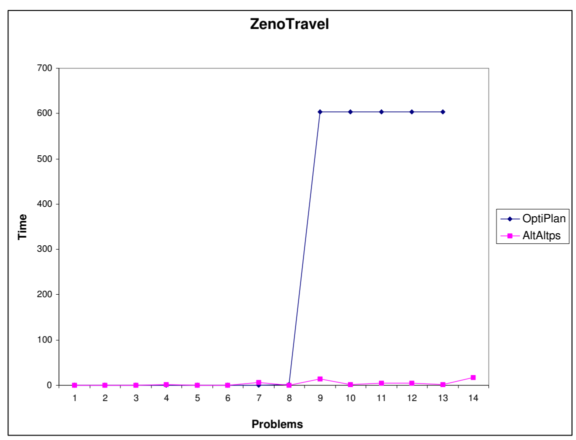
c) Satellite Solution quality



d) Satellite Solution time



e) Zenotravel Solution quality



f) Zenotravel Solution time

Figure 3: Empirical Evaluation

the range of the random values generated were different for each particular planning domain. In other words, a *swithOn* action from the Satellite domain will have execution costs much lower than a *fly* action from the Zenotravel domain. So, we decided upfront the range of execution costs for each domain operator, and random values were generated inside such ranges for each particular instance of each operator during planning. Having generated the cost value of a ground action, this one is maintained across other problems in which the same action is being used. Under these design conditions the planners selected around 60% of the goals on average. Our experiments include the domains of Driverlog, Satellite and Zenotravel.

The optimal solution quality was determined by OptiPlan but is optimal only for the levels less than or equal to the level at which OptiPlan performed its search. OptiPlan does not always detect the optimal solution, this is because of the 600 seconds time limit that was set on the solver. In case the time limit was reached, we denoted the best feasible solution as OptiPlan's solution quality. Since the time limit was set on the CPLEX solver only, OptiPlan's total planning time can get above 600 seconds (total planning time also includes planning graph construction time and the time needed for converting the problem into IP).

Figure 3 displays the solutions on the Driverlog domain. We can clearly see on plot a) that *AltAlt*^{ps} produces excellent quality plans, often within just one second. It is remarkable that for most of the tested problems in Driverlog *AltAlt*^{ps} finds the exact same solutions as OptiPlan. In the problems in which *AltAlt*^{ps} produces different solutions, the solution quality remains close to that of OptiPlan.

Figure 3 c) and Figure 3 e) display the solution quality on the Satellite and Zenotravel domains respectively. We can see that in both of these domains *AltAlt*^{ps} remains very competitive with respect to OptiPlan. In fact, there are solution points in which *AltAlt*^{ps} has generated better plans (see Figure 3 c)). For these plans OptiPlan has just returned the lower bound on the optimal solution due to time restrictions. Furthermore, we can see in Figures 3 b), 3 d), and 3 f) that *AltAlt*^{ps} is more efficient and scalable than OptiPlan.

OptiPlan still performs better, in terms of quality, than *AltAlt*^{ps} in a few problems. This could be due to the following reasons: either the subgoal selection procedure of *AltAlt*^{ps} is too greedy, or the 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 second reason dominates more often. We found in most of these problems that in fact our subgoal selection procedure returns the correct set of subgoals, but the plan found is a little more expensive than the optimal plan found with OptiPlan. To resolve this issue we intend to develop more powerful heuristics for the planning phase of *AltAlt*^{ps}. Heuristics that take into account negative interactions (Nguyen, Kambhampati, & Sanchez 2002) could help us to improve the solutions returned.

In all the problems, when a classical planner was used that would automatically achieve all the goals, the resulting plans often had a significantly lower and sometimes even negative total net benefit when compared to OptiPlan and *AltAlt*^{ps}.

Over-subscription Scheduling and Partial Satisfaction Planning

Planning and scheduling are two related areas that contain many real world problems that are over-subscribed. Examples of over-subscription problems include many space applications. For instance, over-subscription problems have been identified in for image scheduling on Landsat 7 (Potter & Gasch 1998), satellite observation scheduling (Frank *et al.* 2001; Globus *et al.* 2003), space shuttle ground processing (Zweben *et al.* 1994), airlift scheduling (Kramer & Smith 2003), and the scheduling Hubble Space Telescope (Johnston & Miller 1994). The problem of partial satisfaction and over-subscription deals with the problem of choosing a subset of goals or tasks that can be achieved within the available time and resources. There are, however, subtle differences between partial satisfaction planning and over-subscription scheduling.

In scheduling, the problem of over-subscription is centered on the allocation of resources to tasks over time. In an over-subscription problem there are more tasks to accomplish within a given time period than there are available resources. Typical objectives are to optimize resource usage or to accomplish as many tasks as possible. Constructive and repair based approaches have been developed to solve these kind of problems and, recently, stochastic greedy search algorithms on constraint based intervals (Frank *et al.* 2001) and genetic algorithms (Globus *et al.* 2003) have successfully been applied.

Repair based approaches start with a feasible or a nearly feasible schedule and then apply iterative repairs or improvements to the schedule (Minton *et al.* 1992; Johnston & Miller 1994; Kramer & Smith 2003). *AltAlt*^{ps} also applies an iterative approach, it constructs an initial relaxed-plan for a single goal, and then this plan is used to augment the final set of goals that will be considered during search. However, the heuristics employed by *AltAlt*^{ps} are quite different to the set of heuristics used in over-subscription scheduling (Kramer & Smith 2003) because we do not consider time or resource usage, but just the final benefit of the plan.

In a constructive based approach, schedules are built incrementally by heuristically adding more tasks to the partial schedule until no more tasks can be added. Potter and Gasch (1998) use a constructive based approach that allows limited backtracking for image scheduling on the Landsat 7.

In planning, over-subscription is more centered on the goals. Over-subscribed planning problems or PSPs are problems in which there are too many goals that can feasibly be achieved within the available time and resources, or in which goals are conflicting, necessitating choosing a subset of non-conflicting goals. Current planning systems are not specifically designed to handle problems in which a conjunction of goals can not be entirely supported. In other words, problems in which it is necessary to partially support a subset of the top level goals given certain metrics. Unfortunately, most of the real world problems fall under this category.

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 an optimizing (OptiPlan) and a heuristic (*AltAlt*^{ps}) planner for it. Our preliminary empirical results show that the heuristic approach is able to generate plans whose quality is quite close to the ones generated by the optimizing approach, while incurring only a fraction of the cost. We have also presented an spectrum of over-subscribed scheduling problems, and the most common approaches for solving them. We have introduced some connections among PSP and over-subscribed scheduling problems, which will help us to develop new techniques in our future work.

Our future work will extend our greedy framework to more complex planning problems. Specifically, we plan to consider problems in which goals may be interacting (e.g. they are mutexes), requiring a more aggressive account for interactions. We also plan to extend our approach to handle more sophisticated metrics (e.g. resource usage, multi-objective estimations). Finally, we would like to take some ideas from the scheduling area to improve our goal selection procedure. Specifically, we plan to adapt the heuristics developed by (Kramer & Smith 2003) for swapping tasks to rank actions in multiple plans. Our general idea consists of generating feasible plans for single goals and then merge and retract those actions among the plans that maximize our metrics.

References

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

Bonet, B., and Geffner, H. 1999. Planning as Heuristic Search: New Results. In *Proceedings of ECP-99*.

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

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

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

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

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

Johnston, M., and Miller, G. 1994. *SPKIE: Intelligent scheduling of Hubble Space Telescope observations*. Morgan Kaufmann Publishers. 391–422.

Kautz, H., and Selman, B. 1999. Blackbox: Unifying sat-based and graph-based planning. In *IJCAI-99*, 318–325.

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

Long, D., and Fox, M. 1999. Efficient Implementation of the Plan Graph in STAN. *Journal of Artificial Intelligence Research* 10:87–115.

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

McDermott, D. 1999. Using Regression-Match Graphs to Control Search in Planning. *Artificial Intelligence* 109(1-2):111–159.

Minton, S.; Johnson, M.; Philips, A.; and Laird, P. 1992. Minimizing conflicts: A heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence* 58(1):161–205.

Nguyen, X., and Kambhampati, S. 2000. Extracting Effective and Admissible Heuristics from the Planning Graph. In *Proceedings of AAAI/IAAI*, 798–805.

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

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

Sanchez, R., and Kambhampati, S. 2003. Altalt-p: Online parallelization of plans with heuristic state search. *Journal of Artificial Intelligence Research* 19:631–657.

Savitch, W. J. 1970. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences* 4(2):177–192.

Smith, D. 2003. The mystery talk. Planet International Summer School.

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

Vossen, T.; Ball, M.; Lotem, A.; and Nau, D. 1999. On the use of integer programming models in ai planning. In *IJCAI-99*, 304–309.

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

Zweben, M.; Daun, B.; Davis, E.; and Deale, M. 1994. *Scheduling and rescheduling with iterative repair*. Morgan Kaufmann Publishers.