Planning with Goal Utility Dependencies

Minh B. Do
Embedded Reasoning Area
Palo Alto Research Center
Palo Alto, CA 94304
minh.do@parc.com

J. Benton & Subbarao Kambhampati[*]
Computer Science & Engineering Department
Arizona State University
Tempe, AZ 85287
{j.benton,rao}@asu.edu

Abstract

Work in partial satisfaction planning (PSP) has hither to assumed that goals are independent. This implies that that individual goals have additive utility values. In many real-world problems we cannot make this assumption and thus goal utility is not additive. In this paper, we motivate the need for representing and handling goal utility dependencies in PSP and we provide a framework for representing them using the General Additive Independence (GAI) model (Bacchus & Grove 1995). We then present an algorithm based on forward heuristic planning to solve this problem using heuristics derived from the planning graph. To show the effectiveness of our framework, we provide empirical results on benchmark planning domains.

Introduction

Classical planning aims at finding a plan that achieves a set of conjunctive goals. Partial satisfaction (or oversubscription) planning relaxes this all-or-nothing constraint, focusing on finding a plan to achieve the "best" subset of goals (i.e. the plan that gives the maximum tradeoff between total achieved goal utilities and total incurred action cost). The process of selecting the set of goals on which to focus is complicated by two types of dependencies between goals: (i) A set of goals may have *cost dependencies* in that there are dependencies among the actions to achieve them (making the cost of achieving them together significantly more or less than the sum of costs of achieving them in isolation) (ii) A set of goals may have *utility dependencies* in that achieving the goals together may lead to utility that is significantly different from the sum of utilities of achieving each goal in isolation. Both cost and utility dependencies are common in many real world applications such as NASA's data collection domains.

Although some recent work in partial satisfaction planning (van den Briel *et al.* 2004) has begun to handle cost dependencies between goals, there has not yet been any work on handling utility dependencies. The primary contribution of our paper is a systematic approach for handling cost and utility dependencies together in PSP. The main technical challenges here include developing a model where goal utility dependencies can be compactly represented and using utility interactions with cost interactions to find a high net benefit plan. Our approach builds upon the methods for handling utility dependencies from decision theory (c.f. (Bacchus & Grove 1995; Boutilier et. al. 2001)) and combinatorial auctions (c.f. (Nisan 2005)).

We start with a brief overview of two types of utility dependencies that need to be handled:

1. *Mutual dependency:* Utility of the set of goals is different from the summation of the utility of each individual goal. Thus, for $S \subseteq G, u(S) \neq \Sigma_{g \in S} u_g$. Examples: (1) while the utility of having either a left or right shoe alone is *zero*, utility of having both of them is much higher (i.e. the goals "complement" each other); (2) utility of having two cars is smaller than the summation of the individual utilities of having each of them (i.e. the goals "substitute" each other).
2. *Conditional dependency:* The utility of a goal or set of goals depend on whether or not another goal or set of goals are already achieved. Examples: (1) the utility of having a hotel reserved in Hawaii depends on whether or not we have a ticket to Hawaii; (2) in the logistics domain, packages containing parts of a machine that need to be delivered to a given location are only useful in the presence of other parts in the same group.

In this paper, we develop an approach for representing these utility dependencies between planning goals using the *Generalized Additive Independence (GAI)* model (Bacchus & Grove 1995) and describe a planning algorithm based on forward search that solves this extended PSP problem. The algorithm is based on the forward heuristic search used in the *Sapa*$^{\mathcal{PS}}$ planner (van den Briel *et al.* 2004). To guide the heuristic search algorithm, we introduce two different heuristics based on the planning graph structure. The first one is admissible and can be used to find optimal solutions. The second one is inadmissible but is more effective in finding good quality solutions in less time. The main innovation of our heuristic approach is its ability to take into account both goal utility and goal achievement cost dependencies. Our heuristic framework combines both (1) a greedy search for low-cost relaxed plans that handle cost interactions be-

[*]Kambhampati's research is supported in part by the NSF grant IIS-0308139, and the ONR Grant N000140610058.

tween goals, and (2) a declarative Integer Linear Programming (ILP) encoding that captures both mutual goal achievement cost and goal utility dependencies to select the best subset of goals. The solution of this latter ILP encoding is used to guide an anytime best-first search algorithm that returns higher net benefit solutions.

For the rest of this paper, we will first formalize the problem of goal utility dependency. After that we will introduce the GAI model (Bacchus & Grove 1995) and then describe the important steps of the main heuristic search framework. We finish the paper with empirical results showing the effectiveness of our approach and the related and future work.

Problem Formulation

A classical planning problem is a 4-tuple $\langle F, I, G, A\rangle$ where: F is a set of predicate symbols representing state facts; I is the initial state, completely defined by predicates in F; G is a goal state, which is partially defined by a set of predicates in F; A is a set of actions with $a \in A$ defined by pre- and post-conditions $Precond(a), Effect(a) \subseteq F$. A plan P is a sequence of actions from A such that, when executed from I, P will result in a state that achieves all $g \in G$.

In partial satisfaction planning (PSP) (Smith 2004; van den Briel *et al.* 2004), goals $g \in G$ have utility values $u_g \geq 0$, representing how much each goal is worth to a user, and each action $a \in A$ has an associated positive execution cost c_a, which represents the resources spent executing each action. Moreover, not all goals in G need to be achieved. Let P be the lowest-cost plan that achieves a subset $G' \subseteq G$ of those goals, the objective is to maximize the tradeoff between total utility $u(G')$ of G' and total cost of all actions $a \in P$:

$$maximize_{G' \subseteq G} \;\; u(G') - \sum_{a \in P} c_a \tag{1}$$

Work on PSP has until now assumed that goals have no utility dependencies and thus their utilities are additive: $u(G') = \Sigma_{g \in G'} u_g$. As we showed in the previous section, there are many scenarios in which this assumption is not true. To represent the goal utility dependencies as discussed, we adopt the *Generalized Additive Independence* (GAI) model (Bacchus & Grove 1995). Specifically, we assume that the utility of the goal set G can be represented by *k* local utility functions $f^u(g[k]) \in R$ over sets $g[k] \subseteq G$ of goals where $g[k]$ may contain a single goal thereby capturing the utility of that goal. For any subset $G' \subseteq G$ the utility of G' is:

$$u(G') = \sum_{g[k] \subseteq G'} f^u(g[k]) \tag{2}$$

For the rest of this paper, we name the new PSP problem with utility dependencies represented by GAI model *PSP*$^{\mathcal{UD}}$. If there are $|G|$ local functions $f^k(g[k])$ and each $g[k]$ contains a single goal then there are no utility dependencies and thus *PSP*$^{\mathcal{UD}}$ reduces to the original PSP problem. We chose the GAI model because it is simple, intuitive, expressive and it is more general than other commonly used models such as UCP-Net (Boutilier et. al. 2001). To facilitate the discussion on the GAI model for *PSP*$^{\mathcal{UD}}$, we will use the following example in the Mars Rover domain (Smith 2004).

Example: In the Mars Rover domain, a Rover needs to travel between different locations. It then collects the samples and takes either high or low resolution pictures at different locations. Achievement of each goal gives a utility value. We have $g_1^l = sample(l)$, $g_2^l = high\_res(l)$, $g_3^l = low\_res(l)$ with the utility values $u(g_1^l) = 200, u(g_2^l) = 150$ and $u(g_3^l) = 100$ for all locations l of interest. There are also utility dependencies between combinations of goals such as:

- *complement* relation: Utility of having samples at related locations l_1 and l_2 will give additional utility (e.g. $u(\{sample(l_1), sample(l_2)\}) = u(g_1^{l_1}) + u(g_1^{l_2}) + 50$).
- *substitute* relation: Taking both low and high resolution images of the same location will reduce the overall utility (e.g. $u(\{g_2^l, g_3^l\}) = u(g_2^l) + u(g_3^l) - 80$).
- *conditional* relation: Finally, if we already have a picture of a given location l, then the utility of taking a sample at l increases, due to the available information to aid future analysis (e.g. if g_2^l then $u(g_1^l)$ += 100, if g_3^l then $u(g_1^l)$ += 50 and if $g_2^l \wedge g_3^l$ then $u(g_1^l)$ += 110).

For our GAI model, the local functions for these relations in this example would be: $f(g_1^l) = 200$, $f(g_2^l) = 150$, $f(g_3^l) = 100$, $f(\{g_1^{l_1}, g_1^{l_2}\}) = 50$, $f(\{g_2^l, g_3^l\}) = -80$, $f(\{g_1^l, g_2^l\}) = 100$, $f(\{g_1^l, g_3^l\}) = 50$ and $f(\{g_1^l, g_2^l, g_3^l\}) = 110-(100+50) = -40$. At first glance, it may seem strange to have the local function $f(\{g_1^l, g_2^l, g_3^l\})$ having negative value even though those three goals have a complement relation. That's because the formulation to calculate the utility of $\{g_1^l, g_2^l, g_3^l\}$ includes two other complement functions for $\{g_1^l, g_2^l\}$ and $\{g_1^l, g_3^l\}$. Using GAI functions, the utility of a set of goals can be calculated as: $U(\{g_1^l, g_2^l, g_3^l\}) = f(g_1^l) + f(g_2^l) + f(g_3^l) + f(\{g_1^l, g_3^l\}) + f(\{g_1^l, g_2^l\}) + f(\{g_1^l, g_2^l, g_3^l\}) = 200 + 150 + 100 + 50 + 100 - 40 = 560$

Search Algorithm

There are several approaches to solve the PSP problems such as selecting the subset of goals up-front, compilation to ILP, or adapting the A^* search algorithm to PSP (Smith 2004; van den Briel *et al.* 2004). We choose to extend the best-first search framework in *Sapa*$^{\mathcal{PS}}$ to handle *PSP*$^{\mathcal{UD}}$, which uses an $A*$ based search algorithm. With this framework we can analyze the action cost and utility dependencies at each search node.

The search algorithm, which we call A^*_{PSP} is sketched in Figure 1. It starts with the initial state S_{init} and continues to dequeue the most promising node S (i.e. highest $f(S) = g(S) + h(S)$ value). For each search node S, let P_C be the partial plan leading from S_{init} to S, let G_S be the set of goals satisfied in S, $U(S) = \Sigma_{g \in G_S} u_g$ be the utility of S, and let $c(S) = c(P_C) = \Sigma_{a \in P_C} c_a$ be the total cost to visit S. We have $g(S) = u(S) - c(S)$. Let P_R be the plan that, when we apply in S, will lead to S' such that P_R maximizes $h(S) = (U(S') - U(S)) - c(P_R)$. While calculating $g(S)$

```
Open State Queue: $SQ$={$S_{init}$}
Best achieved benefit: $B_B = U(S_{init})$
while $SQ \neq \{\}$
    $S$:= $Dequeue(SQ)$
    if $(g(S) > 0) \wedge (h(S) = 0)$ then
        Terminate Search;
    forall $a$ applicable in $S$
        $S'$ := $Apply(a,S)$
        if $g(S') > B_B$ then
            $Print\ BestBeneficialNode(S')$
            $B_B \leftarrow g(S')$
        if $f(S') = g(S') + h(S') \leq B_B$ then
            $Discard(S)$
        else $Enqueue(S',SQ)$
end while;
```

Figure 1: A* search with negative edges for PSP problems.

is trivial, having a good estimate of $h(S)$ is hard and is the key to the success of best-first search algorithms.

Definition S_T is a termination node if: $h(S_T) = 0$, $g(S_T) > 0$, and $\forall S : g(S_T) > f(S)$.

If a state S is a termination node, we stop the search. If not, we generate children of S by applying applicable actions a to S. If the newly generated node $S' = Apply(a, S)$ is a beneficial node (i.e. $g(S') > 0$) 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 a promising node (i.e. $f(S') > B_B$ where $f(S')$ is the f value of state S' and B_B is the g value of the best beneficial node found so far), 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 plans while conducting search (until a terminal node is found), the algorithm has an "anytime" property. Thus, it can quickly return some beneficial plan. It also can continue to return plans with better net benefit[1].

Proposition *If $h(S)$ is admissible (over-estimates the achievable benefit), then A^*_{PSP} returns an optimal solution.*

Proof sketch: If $f(S)$ over-estimates the real achievable benefit, then the discarded nodes (not enqueued) cannot lead to nodes with higher benefit value than the current best node (B_B). If A^*_{PSP} finishes with an empty queue then the optimal solution should be found because all nodes enqueued are visited. If A^*_{PSP} found a termination node S_T, all nodes remaining in the queue can lead to solutions with lower total benefit than S_T. Thus S_T is an optimal solution.

[1]If we consider heuristic planning search as a graph search problem, then PSP has some interesting properties (1) the edge cost $v = (u(S') - u(S)) - c_a$ of moving from a state S to state $S' = apply(S, a)$ can be negative; (2) any reachable state can be a valid goal state. We have not found a cyclic graph search algorithm dealing with problems having the same properties.

Heuristics for Maximizing Plan Benefit

The main objective of PSP planners, as described in Equation 1, is to find the best plan in terms of total benefit, which is calculated as *benefit = total achieved utility - total action cost*. The key to the success of A^*_{PSP} is an effective heuristic function capable of estimating the remaining achievable benefit $h(S)$ for each generated state S during forward search. We base our heuristic routine on the planning graph cost-propagation framework first used in the $Sapa$ planner (Do & Kambhampati 2001). We how describe how our heuristic estimates cost and utility.

Cost-Propagation on the Relaxed Planning-Graph

For PSP problems, the cost-propagation process on the planning graph is used to estimate the achievement cost for each individual goal. Starting with the achievement cost of $c(f) = 0$ for facts f in the initial state I and $c(f) = c(a) = \infty$ for all other facts and all actions, the propagation rules to estimate costs to achieve different facts p and to execute actions a are[2]:

1. Facts: $\forall f : c(f) = min\ (c(a) + c_a) : f \in Effect(a)$
2. Max-prop: $\forall a \in A, f \in Precond(a) : c(a) = max\ c(f)$
3. Sum-prop: $\forall a \in A, f \in Precond(a) : c(a) = \Sigma c(f)$

The update rules are used while extending the (relaxed) planning graph structure (Blum & Furst 2001) from the initial state, with each $c(f)$ or $c(a)$ value updated exactly once. After the propagation is done, for each individual goal $g \in G$, the value $c(g)$ is an estimate on the cost to achieve g. As shown in (Do & Kambhampati 2001), if we use *max* propagation, then $c(g)$ will underestimate the cost to achieve g while there is no such guarantee for *sum* propagation. Using $c(g)$ calculated by *max* propagation, we can estimate the achievable benefit value as below:

$$h_{max} = max_{G' \subseteq G}\ [u(G') - (max_{g \in G'} c(g))] \tag{3}$$

It's easy to see that h_{max} overestimates the real achievable benefit and thus A^*_{PSP} using h_{max} will output an optimal solution. One brute force way to estimate h_{max} is by enumerating over all ($2^{|G|}$) possible subsets of G, which can be prohibitive for a large goal set. In the following, we will introduce an approach for estimating h_{max} using an Integer Linear Programming (ILP) encoding.

Relaxed-Plan based Heuristic

Because the cost estimate for each goal using the cost-propagation on the planning graph can highly underestimate the real cost of a set of goals, the max family of heuristics as in h_{max} tends to perform badly in practice. Therefore, we use an alternative approach of utilizing the relaxed plan employed by *Sapa*$^{\mathcal{PS}}$ for partial satisfaction

[2]c_a, which is the execution cost of a, is different from $c(a)$, which is the cost to enable the execution of a (i.e. cost to achieve preconditions of a)

planning[3]. For each state S explored in a progression planner we build the relaxed planning graph and perform a forward cost propagation on the graph. After this we use the following general procedure to extract a relaxed plan to support a subset of goals $G' \subseteq G$:

1. Let subgoal set $SG = G$ and the relaxed-plan $RP = \emptyset$.
2. For each $g \in SG \setminus I$ select action $a : g \in Effect(a)$. Add a to RP and $p \in Precond(a) \setminus I$ to SG.
3. Repeat until $SG = \emptyset$.

The plan RP is called a "relaxed plan" because we ignore negative effects of all actions in the planning graph while extracting it. This relaxation allows a non-backtracking plan extraction process that finds RP quickly. Ideally, we would like to extract the relaxed-plan with the highest net benefit. Let $RP(G')$ be the relaxed-plan with highest net benefit value among those achieving $G' \subseteq G$. The relaxed plan heuristic for *PSP*$^{\mathcal{UD}}$ is:

$$h_{relax} = max_{G' \subseteq G} \; (u(G') - \sum_{a \in RP(G')} c_a) \qquad (4)$$

Extracting the relaxed plan that has the highest net benefit (i.e. utility minus cost), tends to get complex as the set of goals G' we need to focus on depends both on the cost dependencies as well as their utility dependencies. To approximate h_{relax} in previous work, the cost dependencies were partially handled by biasing the relaxed plan extraction to: (i) greedily select actions with lowest achievement cost ($c(a) + c_a$); (ii) reuse actions selected for other goals. We further extend this approach to approximate the optimal relaxed plan in the presence of utility dependencies as follows:

1. Greedily extract the lowest-cost relaxed plan RP that achieves the *largest* set of achievable goals. The relaxed plan extraction is not sensitive to utility dependencies. We now can search in its neighborhood for a plan that attempts higher net benefit, while taking utility dependencies into account.
2. Capture the achievement cost dependencies between achievable goals using the causal structure of RP.
3. Pose the problem of extracting the optimal relaxed plan within RP that takes both cost and benefit dependencies into account as an ILP encoding. This encoding is then solved by an ILP solver, and the solution is used as the heuristic $h(S)$ to guide A^*_{PSP}.

Alert readers would have noted that if we compile the entire relaxed planning graph (rather than just the greedily extracted relaxed plan) we can post the entire relaxed plan extraction process (including step 1) as an ILP problem. Despite its conceptual elegance, we chose not to follow this route as the cost of heuristic computation increases quite significantly (especially for our progression planner which extracts relaxed plans at each node). Related to this is whether we can encode the full general problem as an ILP. While this is possible, previous work has shown that this approach does not scale well in even simpler problems (van den Briel *et al.* 2004).

[3]Variants of this approach are also used in several other PSP planners such as *AltAlt*^{ps} (van den Briel *et al.* 2004) and the orienteering planner (Smith 2004).

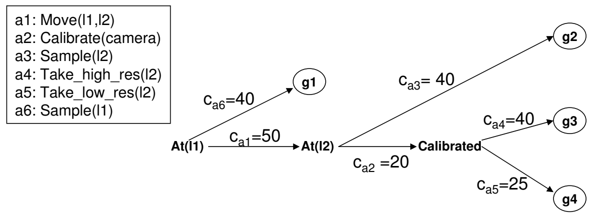

Figure 2: Relaxed plan for Mars Rover domain.

As another observation, note the possibility of performing steps 2 and 3 in a greedy procedural form. We found that such a procedure becomes quite hairy because of the complex utility dependencies. We also note that because the relaxed planning graph may not contain the optimal relaxed plan, h_{relax} is not guaranteed to be admissible (and can thus underestimate the achievable net benefit).

Cost Dependencies In our ongoing example, plans consist of actions for traveling between different locations, calibrating instruments and carrying out experiments; all have costs (e.g. in proportion to the amount of energy and time consumed by different actions). Because certain actions contribute to the achievement of multiple goals, there are also mutual dependencies between the costs of achieving sets of goals. Those relations can be discovered by using the causal structure of the extracted relaxed plan.

Figure 2 shows the relaxed plan for the planning instance in which the Rover is initially located at l_1 and the desired goals are $\{g_2, g_3, g_4\}$ with $g_2 = sample(l_2)$, $g_3 = high\_res(l_2)$, and $g_4 = low\_res(l_2)$. The relaxed plan contains 5 actions: $a_1 = move(l_1, l_2)$ with cost $c_{a_1} = 50$, $a_2 = calibrate(camera)$ with $c_{a_2} = 20$, $a_3 = take\_sample(l_2)$ that achieve goal g_2 with cost $c_{a_3} = 40$, and two actions a_4, a_5 to take the pictures at l_2 of different quality with cost $c_{a_4} = 40$ and $c_{a_5} = 25$ respectively. As discussed above, partial plans achieving different individual goals can overlap. For example, the partial plans to achieve individual goals g_2, g_3 and g_4 all share action a_1 and plans for g_3 and g_4 share the action a_2. Thus, the cost to achieve the set S_g of goals follows the *substitute* dependencies in which the cost to achieve S_g can be smaller than the summation of the individual costs to achieve each $g \in S_g$. For example, the cost to achieve goal g_3 is $c(g_3) = c_{a_1} + c_{a_2} + c_{a_4} = 110$ and $c(g_4) = c_{a_1} + c_{a_2} + c_{a_5} = 95$ while $c(\{g_3, g_4\}) = c_{a_1} + c_{a_2} + c_{a_4} + c_{a_5} = 135 < c(g_3) + c(g_4) = 205$.

To capture the mutual dependencies between the goal achievement costs, we find the set of actions shared between different partial plans achieving different goals. This procedure utilizes the causal links, each of which specifies the achievement action for a goal or action precondition, gathered while extracting the relaxed plan.

1. Initialize: $\forall a \in P : GS(a) = \emptyset$; $\forall p \in Effect(a) \bigcup Prec(a) : GS(p) = \emptyset$; $\forall g \in G : GS(g) = \{g\}$.
2. Backward sweep from goals and update: $GS(a) = \bigcup GS(p) : p \in Effect(a)$ and $GS(p) = \bigcup GS(a) : p \in Precond(a)$

Using the update procedure above, for each action a, $GS(a)$ contains the set of goals g to which a contributes. For example, $GS(a_1) = \{g_2, g_3, g_4\}$, $GS(a_2) = \{g_3, g_4\}$ and $GS(a_3) = \{g_2\}$.

Estimating Achievable Benefit A realistic planning problem with goal utility dependencies will likely include several goals involved in multiple dependencies, considerably increasing the complexity of the problem. The challenge is to use the relaxed plan as means of capturing possible goal combinations while also being informed by the cost of the relaxed actions involved in achieving these goals.

Given the utility dependencies represented by GAI local functions f^u and the goal achievement cost dependencies represented by goal supporting action set $GS(a)$, we set up an ILP encoding for h_{relax}:

- Binary Variables:
  - $\forall a \in P$: create one binary integer variable X_a.
  - $\forall g \in G$: create one binary integer variable X_g.
  - $\forall G' \subseteq G,\ f^u(G') \neq 0$: create one binary integer variable $X_{G'}$.
- Constraints:
  - $\forall a \in P, \forall g \in GS(a) : (1 - X_g) + X_a \geq 1$
  - $\sum_{g \in G'}(1 - X_g) + X_{G'} \geq 1$
  - $\forall g \in G' :\ (1 - X_{G'}) + X_g \geq 1$
- Objective function: $max\ (\Sigma f^u(G') * X_{G'} - \Sigma X_a * c_a)$

The purpose of this encoding is to capture the set of goals $G' \subseteq G$ that gives the maximum tradeoff between the utility of G' and the cost of actions in the relaxed plan supporting G'.

The first constraint enforces that if a given goal is selected for achievements, then any action that contributes to the achievement of that goal should be selected too. The second and third types of constraints ensure that if there is a GAI local function for a set of goals $G' \subseteq G$, then this local function (represented by a binary variable $X_{G'}$) will be activated ($X_{G'} = 1$) if and only if all goals $g \in G'$ are selected ($X_g = 1$). The value we get from solving this ILP encoding can then be used as an estimate of the achievable benefit for a given state ($h(S)$ value) for the A^*_{PSP} search algorithm outlined in Figure 1.

For h_{max} we can also setup an ILP encoding which is simpler than the encoding for h_{relax} because there is no need for variables and constraints related to actions and goal supporting $GS(a)$ sets.

- Variables: besides X_g and $X_{G'}$, create one variable X_{C_G} representing the cost to achieve G.
- Constraints: besides the second and third types of constraints as in the encoding for h_{relax} above, introduce one constraint: $\forall g \in G :\ X_{C_G} - X_g * c(g) \geq 0$ where $c(g)$ is calculated during cost propagation on the graph (as in Equation 3).
- Objective Function: $max\ (\Sigma f^u(G') * X_{G'} - X_{C_G})$

The variable X_{C_G} and the constraint with X_g guarantee that the cost to achieve a set of goals G' is the maximum of the cost to achieve any goal $g \in G'$.

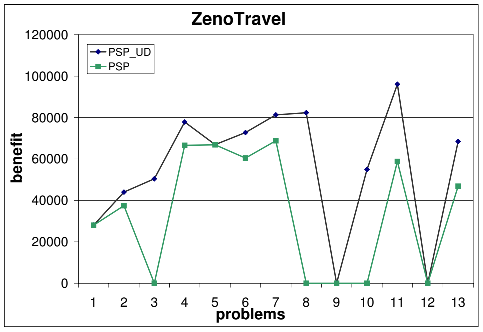

Figure 3: *SPUDS* and *Sapa*$^{\mathcal{PS}}$ in ZenoTravel domain.

Empirical Results

We have implemented the heuristic search algorithm for the *PSP*$^{\mathcal{UD}}$ problems discussed in this paper on top of the *Sapa*$^{\mathcal{PS}}$ planner. We call the new planner *SPUDS*. We tested our planner on two sets of random *ZenoTravel* and *Satellite* problems, which were generated on top of the problem sets used in the Third International Planning Competition (Long & Fox 2003). The ZenoTravel domain involves moving people by airplanes between different cities and the Satellite domain involves turning satellites to point at different objects and taking pictures of them. A more detailed description of these domains can be found at the IPC3 website[4].

All tests were run using a Pentium IV 2.66GHz with 1GB RAM and a 1200 second time limit. Because A^*_{PSP} continuously finds better solutions given more time (or a termination node is found), the results reported in this section represent the plan with highest benefit value found within the time limit. For solving the ILP encoding, we use the C version of `lp_solve version 5.5`, a free solver with a Java wrapper.

Generating Test Problems: Given that in general, action cost is decided by the amount of resources consumed and/or the time spent by that action, we decided to automatically generate the *PSP*$^{\mathcal{UD}}$ problems from a set of metric temporal planning problems from the ZenoTravel and Satellite domains (used in IPC3) as follows:

- *Domain File:* We modified the domain files by adding a cost field to each action description. Action cost is represented as a mathematical formula involving numeric variables that exist in the original problem description and

[4]http://planning.cis.strath.ac.uk/competition/

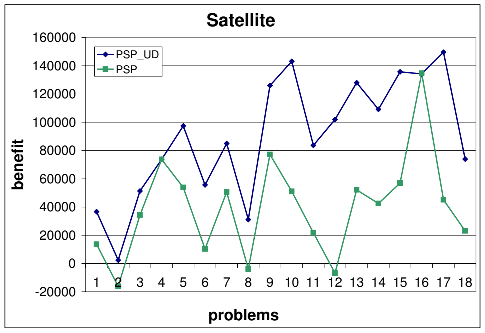

Figure 4: *SPUDS* and *Sapa*$^{\mathcal{PS}}$ in Satellite domain.

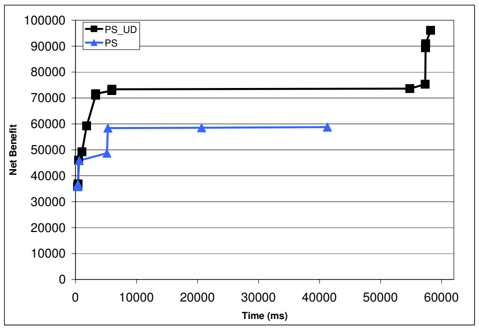

Figure 5: Example run of planners. (Problem 11 in ZenoTravel)

also new numeric variables that we have added to the domain description file. The cost field utilizes both the functions representing numeric quantities in the original problem descriptions, and the newly introduced cost-related functions used to convert the temporal and resource consumption aspects of each action into a uniform plan benefit represented by the amount of money spent (as in the ZenoTravel domain) or energy consumed (as in Satellite domain).

- *Problem File:* For each domain, we implemented a Java program that parses the problem files used in IPC3 and generates the *PSP*$^{\mathcal{UD}}$ version with cost-related function values randomly generated within appropriate upper and lower bounds. The goal utilities are also randomly generated within different upper and lower bounds. The goals are randomly selected to be "hard" or "soft". The set of goal dependencies along with their utility values are also randomly generated. Thus, the number of dependencies, size of the dependencies, set of goals involved and the utility values are all randomly selected within certain lower and upper bounds (e.g. upper bound on the number of dependencies is $3 * |G|$).

Analysis: We ran both *Sapa*$^{\mathcal{PS}}$ and *SPUDS* on problems from the two domains. While the latter is sensitive to both cost and utility dependencies, the former (*Sapa*$^{\mathcal{PS}}$) only accounts for cost dependencies. Due to poor performance of h_{max} in comparison to h_{relax} in tests[5], we focus only on h_{relax} in evaluating *SPUDS*. The empirical evaluation is designed to test whether *SPUDS* is able to solve the *PSP*$^{\mathcal{UD}}$ problems more effectively (i.e. with higher net benefit). Figures 3 and 4 show the comparison between those two planners.

In the ZenoTravel domain, *SPUDS* is better than *Sapa*$^{\mathcal{PS}}$ in 10/13 problems. Both planners find the same solutions in the remaining three problems.

In the Satellite domain, *SPUDS* is better in 16/18 problems, and most of the time is significantly better (up to 16 times better in overall best plan quality). They are equal in 1 problem and *Sapa*$^{\mathcal{PS}}$ is slightly (0.3%) better in 1 problem.

[5]In our empirical tests using h_{relax} allowed *SPUDS* to find plans that were on average 27% better than those found when using h_{max} (when using h_{max} found a plan at all).

We found that the ILP encoding increases the time spent per search node by 3 to 200 times with the highest increase occurring on the larger problems. The advantage gained by *SPUDS* in heuristic guidance offsets this additional computation cost. This behavior is shown in Figure 5 through a plot of a run on problem 11 of ZenoTravel. For this case, *SPUDS* finds a better plan than *Sapa*$^{\mathcal{PS}}$ at 533 milliseconds and this trend continues; *SPUDS* retains its net benefit superiority.

Of the problems solved by *SPUDS*, the average number of actions in the plans was 14.5 for ZenoTravel and 126.2 in Satellite. For Satellite, this number excludes a single outlier problem that included 1850 actions. For *Sapa*$^{\mathcal{PS}}$, the average number of actions included in ZenoTravel plans was 13.75 and in Satellite plans was 121.1. These averages exclude the same outlier problem in Satellite. Our results underscore the fact that *SPUDS* finds higher net benefit plans even though shorter plans are found using *Sapa*$^{\mathcal{PS}}$. In fact, most of the plans found by *Sapa*$^{\mathcal{PS}}$ are shorter because they achieve fewer goals and therefore lose out on the utility that those goals give.

In conclusion, *SPUDS* is significantly better than *Sapa*$^{\mathcal{PS}}$ in both the ZenoTravel and Satellite domains. This shows that our declarative heuristic technique of using an ILP encoding to extract the best part of the relaxed plan regarding the utility dependencies pays off despite higher overhead. We are in the process of more creating test problems based on other benchmark problem sets. We also hope to enable other PSP planners such as *AltAlt*^{ps} and *OptiPlan* (van den Briel *et al.* 2004) to handle *PSP*$^{\mathcal{UD}}$ problems and compare these with *SPUDS*.

Related Work

There has been work on PSP problems using *orienteering* to select goal subsets by David Smith (2004). Also, van den Briel et. al. (2004) introduced several planners such as *AltAlt*^{ps}, *Sapa*$^{\mathcal{PS}}$, and $OptiPlan$ that tackle PSP by either greedily selecting goals up-front, heuristically searching for solutions, or compiling the problem into ILP. None of those planners deal with utility dependencies as described in this paper.

PrefPlan (Brafman & Chernyavsky 2005) can find optimal plans with preferences between goals specified in CP-Net. Both PSP planners and PrefPlan can handle soft-goals; however, PSP planners explicitly reason about quantitative goal utility and action costs, while PrefPlan handles qualitative goal preferences.

Besides the GAI model that we used to represent the utility dependencies, there are several other attractive models such as UCP-Net (Boutilier et. al. 2001) and the graphical model (Bacchus & Grove 1995). While both provide a graphical representation that can make it easier for users to understand the dependencies, the GAI has advantages in that it is more general than UCP-Net while still being simple and intuitive.

In combinatorial auctions, the utility for a set of items up for bid are normally non-additive and share many similarities with reasoning about sets of goals that are dependent in PSP. While a bidding process is different from planning, the bidding language (Nisan 2005) can be used to represent utility dependencies in $PSP^{\mathcal{UD}}$.

Conclusion & Future Work

In this paper, we discussed a framework of solving partial satisfaction planning (PSP) problems with utility dependencies. We show how to represent various types of dependencies using the GAI framework. We also introduced an admissible heuristic, h_{max}, and an inadmissible heuristic, h_{relax}, that when used with the A^*_{PSP} search algorithm, will find optimal or inoptimal solutions respectively. We empirically demonstrated the effectiveness of the new heuristic framework on two benchmark planning domains.

We plan to extend this work to combine both quantitative preferences as in PSP with qualitative preference model as handled in PrefPlan. To improve the performance, we plan on investigating more effective admissible heuristics and more aggressively take into account negative information, such as residual cost as described in AltWlt (Sanchez & Kambhampati 2005) to improve the heuristic quality. We are also looking at the possibility of converting the utility dependencies into dummy goals to simplify the problems.

References

Bacchus, F. and Grove, A. 1995. Graphical Models for Preference and Utility In *Proc. of UAI-95*.

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

Boutilier, C., Bacchus, F., and Brafman, R. 2001. UCP-Networks: A Directed Graphical Representation of Conditional Utilities. In *Proc. of UAI-2001*.

Boutilier, C., Brafman, R., Hoos, H., and Poole, D. 1999. Reasoning with conditional ceteris paribus preference statements. In *Proc. of UAI-2001*.

Brafman, R., and Chernyavsky, Y. 2005. Planning with Goal Preferences and Constraints. In *Proc. of ICAPS-2005*.

Brafman, R. 2004. Eliciting, Modeling, and Reasoning about Preference using CP-nets *Tutorial at UAI-2004*

Dasgupta, P., Sen, A., Nandy, S., and Bhattacharya, B. 2001. Searching Networks with Unrestricted Edge Costs. *IEEE Transactions on Systems, Man, and Cybernetics*

Dechter, R. 2003. Constraint Processing. *Morgan Kaufmann Publishers.*

Do, M. and Kambhampati, S. 2001. Sapa: A Domain-Independent Heuristic Metric Temporal Planner. In *Proc. of ECP-01.*

Long, D. and Fox, M. 2003. The 3rd International Planning Competition: Results and Analysis. *Journal of AI Research.* 20:1-59.

Nisan, N. 2000. Bidding and Allocation in Combinatorial Auctions In *Proc. of ACM conf. on Electronic Commerce*.

Sanchez, R. and Kambhampati, S. 2005. Planning Graph Heuristics for Selecting Objectives in Over-subscription Planning Problems. In *Proc. of ICAPS-05.*.

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

van den Briel, M., Sanchez, R., Do, M, and Kambhampati, S. 2004. Effective Approaches for Partial Satisfaction (Over-Subscription) Planning. In *Proc. of AAAI-04*

Williamson, M., and Hanks, S. 1994. Optimal Planning with a Goal-Directed Utility Model. In *Proc. of AIPS-94*.