A Hybrid Linear Programming and Relaxed Plan Heuristic for Partial Satisfaction Planning Problems

J. Benton
Dept. of Computer Science and Engineering
Arizona State University
Tempe, AZ 85287, USA
j.benton@asu.edu

Menkes van den Briel
Dept. of Industrial Engineering
Arizona State University
Tempe, AZ 85287, USA
menkes@asu.edu

Subbarao Kambhampati
Dept. of Computer Science and Engineering
Arizona State University
Tempe, AZ 85287
rao@asu.edu

Abstract

The availability of informed (but inadmissible) planning heuristics has enabled the development of highly scalable planning systems. Due to this success, a body of work has grown around modifying these heuristics to handle extensions to classical planning. Most recently, there has been an interest in addressing partial satisfaction planning problems, but existing heuristics fail to address the complex interactions that occur in these problems between action and goal selection. In this paper we provide a unique heuristic based on linear programming that we use to solve a relaxed version of the partial satisfaction planning problem. We incorporate this heuristic in conjunction with a lookahead strategy in a branch and bound algorithm to solve a class of over-subscribed planning problems.

Introduction

A popular and successful method for solving planning problems has been to use heuristic search with relaxed plan heuristics that based on ignoring delete lists. Typically these heuristics, first introduced by the planner FF (Hoffmann and Nebel 2001), are based on finding relaxed solutions over a planning graph. In classical planning and many of its variants these heuristics allow best-first search frameworks to find feasible solutions. However, finding such solutions is not enough in partial satisfaction planning (PSP). In these types of problems, actions are given costs and goals are given utility. Additionally, only a subset of the goals need to be satisfied such that we can achieve the maximum difference between the utility of the goals and cost for achieving them, or *net benefit*. Any sound plan can represent a solution to such problems. As such, simply reaching a feasible goal state is trivial. Instead, we must find the state with the best quality.

While some attempts have been made towards adapting relaxed plan heuristics to PSP problems (van den Briel *et al.* 2004; Do *et al.* 2007), there is a fundamental mismatch. Relaxed plan heuristics are good at estimating the set of actions (and their cost) for achieving a given set of top level goals. In PSP problems, we do not up front know the goals that will be supported in the eventual optimal plan. The actions and the goals need to be selected together so as to optimize the net benefit. This requires a heuristic estimate (relaxation) with a more global "optimization" perspective.

A standard way of setting up a relaxation that is sensitive to global optimization perspective involves (i) setting up an integer programming (IP) encoding for the PSP problem and (ii) computing a linear programming (LP) relaxation of this encoding. In addition to being sensitive to the objectives of the optimization, such a relaxation is also sensitive to the negative interactions between the actions–something that is notoriously missing in the standard relaxed plan heuristics. One challenge in adopting this approach involves deciding on the exact type of IP encoding for the PSP problem. Although IP encodings for PSP problems have been proposed in the literature (Do *et al.* 2007), such encodings are made for bounded horizons. The normal idea in bounded horizon planning is to put a bound on the number of plan steps. While this idea works for finding feasible plans, it does not work for finding optimal plans since it is not clear what step bound is required to guarantee optimality.

In this paper, we adopt an encoding that is *not dependent on the horizon bound*. In particular, we describe a causal encoding for action selection that accounts for the delete effects of the actions but ignores action ordering. While we use its solution value as a heuristic to guide search, the encoding also presents opportunities to exploit its unique structure. Namely, because it is informed of negative interactions, its action selection can bring greater insight as to what operators are required to reach the goal than the typical relaxed plan heuristic. We use the action selection to our advantage, by performing lookahead based on the actions that it selects, similar to what is done in the planner YAHSP (Vidal 2004). This helps offset the cost of solving the encoding by finding high-quality states and giving candidate solutions in a branch and bound search.

To perform the lookahead procedure effectively, we must have an ordered set of actions. One way to find such an ordering is to use a typical relaxed plan. We propose a novel way of combining the output of the linear programming relaxation of the action encoding with relaxed plan extraction.[1] Specifically, the relaxed plan extraction is started with the goals "selected" by the LP solution, and is biased to choose actions that appear in the LP solution.[2]

[1] In fact, this technique can be used in combination with any other goal and action selection technique.

[2] Given that the LP solution will be fractional, "selected by LP

Problem Representation and Notation

Classical planning problems can be described by a set of fluents F of predicate symbols, an initial state I defined by the predicates of F, a goal set G specified by a partial set of F and a set of actions A. Each action $a \in A$ consists of a tuple $\langle pre(a), add(a), delete(a)\rangle$ where $pre(a)$ is the set of preconditions of a, $add(a)$ is the add list of a and $delete(a)$ is the delete list of a. Applying an action a is possible when all of the preconditions are met in a state S. Given a state S, the application of an action a is defined as $Apply(a, S) = (S \bigcup add(a))\ del(a)$ iff $pre(a) \subseteq S$.

Partial satisfaction planning with goal utility dependencies PSP^{UD} (Do *et al.* 2007) extends classical planning by assigning a utility value to sets of goals using k local utility functions, $f^u(G_k) \in \mathbb{R}$ on $G_k \subseteq G$, where any goal subset $G' \subseteq G$ has an evaluated utility value of $u(G') = \sum_{G_k \subseteq G'} f^u(G_k)$. This follows the *general additive independence*(GAI) model for specifying utility (Bacchus and Grove 1995). In this way, any set of goals may be assigned a real valued utility. Additionally, each action $a \in A$ has an associated cost $cost(a)$, such that $cost(a) \geq 0$.

LP Heuristic

We present a novel admissible heuristic that solves a relaxation of the original PSP^{UD} problem by using the LP-relaxation of an IP formulation. We build on the heuristic discussed in (van den Briel *et al.* 2007) for classical planning. While most heuristics ignore the delete effects of the actions, this heuristic accounts for the delete effects, but ignores action orderings instead. The formulation that we describe is based on the SAS+ planning formalism (Bäckström and Nebel 1995), where a SAS+ planning task is a tuple $\Pi = \langle V, A, s_0, s_*\rangle$ such that $V = \{v_1, \ldots, v_n\}$ represents a set of state variables, A is a finite set of actions, s_0 indicates the initial state and s_* denotes the goal variable assignments. Each $v \in V$ has a domain D_v and takes a single value f from it in each state s, stated as $s[v] = f$. Each action $a \in A$ includes a set of preconditions, $pre(a)$, post-conditions, $post(a)$, and prevail conditions, $prev(a)$.

Previous work has shown that we can translate classical (STRIPS) planning problems into SAS+ planning problems (Edelkamp and Helmert 1999).

IP Encoding

Our formulation is based on the domain transition graphs. Each of the graphs represents a variable in the SAS+ formalism with a value of a variable existing as a vector and effects as arcs between them. We define a network flow problem over each of them. Side constraints are introduced to handle pre-, post-, and prevail-conditions of actions. Additionally, we incorporate parameters, variables, and constraints to handle aspects of PSP^{UD} problems. Unlike previous integer programming formulations ours does not use a step-based encoding. In a step-based encoding the idea is to set up a formulation for a given plan length and increment it if no solution can be found. Such an encoding may become impractically large, even for medium sized planning tasks and cannot guarantee global cost optimality.

The variables in our formulation indicate how many times an action is executed, and the constraints ensure that all the action pre- and post-conditions must be respected. We also include variables for achievement of goal utility dependencies, where the achievement of specified sets of goals forces modifications in the final *net benefit* values.

In order to describe our formulation, we introduce the following parameters:

- $cost(a)$: the cost of action $a \in A$.
- $utility(v, f)$: the utility of achieving the value f in state variable v in the goal state.
- $utility(k)$: the utility of achieving the goal utility dependency G_k in the goal state.

and the following variables:

- $action(a) \in \mathbb{Z}^+$: the number of times action $a \in A$ is executed.
- $\mathit{effect}(a, v, e) \in \mathbb{Z}^+$: the number of times that effect e in state variable v is caused by action a.
- $prevail(a, v, f) \in \mathbb{Z}^+$: the number of times that the prevail condition f in state variable v is required by action a.
- $endvalue(v, f) \in \{0, 1\}$: is equal to 1 if value f in state variable v is achieved at the end of the solution plan, 0 otherwise.
- $goaldep(k) \in \{0, 1\}$: is equal to 1 if goal utility dependency G_k is satisfied, 0 otherwise.

The objective is to find a plan that maximizes the difference between the total utility that is accrued and the total cost that is incurred.

$$\text{MAX} \sum_{v \in V, f \in D_v} utility(v, f) endvalue(v, f)$$
$$+ \sum_{k \in K} utility(k) goaldep(k) - \sum_{a \in A} cost(a) action(a)$$

The constraints ensure that the action pre- and post-conditions are respected, and link the utility dependencies with their respective state variable values.

- Action implication constraints for each $a \in A$ and $v \in V$. The SAS+ formalism allows the pre-conditions of an action to be undefined (Bäckström and Nebel 1995). We model this by using a separate effect variable for each possible pre-condition that the effect may have in the state variable. We must ensure that the number of times that an action is executed equals the number of effects and prevail conditions that the action imposes on each state variable. Hence, if an action is executed twice, then all its effects and prevail conditions are required twice.

$$action(a) = \sum_{\text{effects of } a \text{ in } v} \mathit{effect}(a, v, e)$$
$$+ \sum_{\text{prevails of } a \text{ in } v} prevail(a, v, f)$$

solution" is interpreted as goals (actions) that have values above a threshold.

- Effect implication constraints for each $v \in V$, $f \in D_v$. In order to execute an action effect its pre-condition must be satisfied. Hence, if we want to execute an effect that deletes some value multiple times, then we must ensure that the value is added multiple times.

$$1\{\text{if } f \in s_0[v]\} + \sum_{\text{effects that add } f} \textit{effect}(a, v, e) = \sum_{\text{effects that delete } f} \textit{effect}(a, v, e) + endvalaue(v, f)$$

- Prevail implication constraints for each $a \in A$, $v \in V$, $f \in D_v$. In order to execute an action prevail condition it must be satisfied at least once. Hence, if there is a prevail condition on some value, then that value must be added. We set M to an arbitrarily large value.

$$1\{\text{if } f \in s_0[v]\} + \sum_{\text{effects that add } f} \textit{effect}(a, v, e) \geq prevail(a, v, f)/M$$

- Goal dependency constraints for each goal utility dependency k. All values of the goal utility dependency are achieved at the end of the solution plan if and only if the goal utility dependency is satisfied.

$$goaldep(k) \geq \sum_{f \text{ in dependency } k} endvalue(v, f) - (|G_k| - 1)$$

$$goaldep(k) \leq endvalue(v, f) \quad \forall f \text{ in dependency } k$$

The solution to our formulation is a relaxation because it ignores action ordering. We further use the linear programming (LP) relaxation of this formulation as an admissible heuristic in our branch and bound framework,*and in this sense our heuristic is doubly relaxed*.[3] At every node in the branch and bound search we solve the corresponding LP, however, we avoid instantiating a new LP at every node by adjusting the initial state and updating the respective coefficients in the constraints. This allows us to quickly re-solve the LP as the LP solver will use current solution LP to optimize over the updated coefficients. We call the heuristic h_{LP}. Given the non-relaxed version of this heuristic, h_{IP}, and the optimal heuristic, h_{opt}, we have the relationship $h_{LP} \geq h_{IP} \geq h_{opt}$.

Example: To illustrate the heuristic, let us consider a transportation problem where we must deliver a person, *per1* to a location, *loc2* using a plane, *p1*, and must end with the plan at *loc3*. The cost of flying from *loc1* to *loc2* is 150, from *loc1* to *loc3* is 100, from *loc3* to *loc2* is 200, and from *loc2* to *loc3* is 100. To keep the example simple, we start *per1* in the plane. There is a cost of 1 for dropping the person off. Having the person and plane at their respective destinations each give us a utility of 1000 (for a total of 2000). Figure 1 shows an illustration of the example.

[3]Note that when we subtract the cost to a node, we find the node's f-value, $f(S) = g(S) + h(S)$ (the combination of the heuristic and node cost).

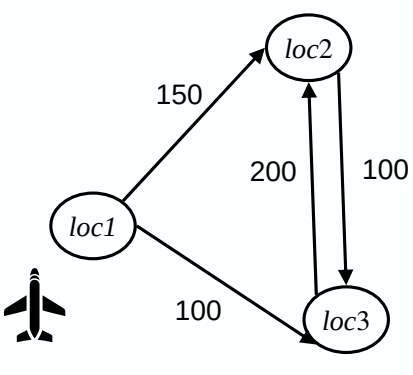

Figure 1: A transportation domain example with each edge labelled with the cost of travelling the indicated direction. Not shown are the utility values for achieving the goal of having person 1, *per1*, at *loc2* and the plane, *p1*, at *loc3* (1000 utility for each goal).

The optimal plan for this problem is apparent. With a total cost of 251, we can fly from *loc1* to *loc2*, drop off *per1*, then fly to *loc3*. Recall that the LP heuristic, while it relaxes action ordering, works over SAS+ multi-valued fluents. The translation to SAS+ captures the fact that the plane, *p1*, can be assigned to only a single location. This is in contrast to planning graph based heuristics that ignore delete lists. Such heuristics consider the possibility that objects can exist in more than one location at a given step in the relaxed problem. Therefore, at the initial state, a planning graph based heuristic would return a relaxed plan (RP) that allowed the plane *p1* to fly from *loc1* to *loc2*, and *loc1* to *loc3*, putting it in multiple places at once.

In contrast, the solution from the LP-based heuristic for this problem at the initial state includes every action in the optimal plan. In fact, "1.0" is the value returned for these actions.[4] Though this is a small example, the behavior is indicative of the fact that the LP, through the encoding of multi-valued fluents, is aware that a plane cannot be wholey in more than one place at a time. In this case, the value returned (the *net benefit*, or $2000 - 251 = 1749$) gives us the perfect heuristic.

To use this solution as a candidate in the branch and bound search described in the next section, we would like to be able to simulate the execution of the relaxed plan. For the example problem, this would allow us to reach the goal optimally. But because our encoding provides no action ordering, we cannot expect to properly execute actions given to us by the LP. For this example, it appears that a greedy approach might work. That is, we could iterate through the available actions and execute them as they become applicable. Indeed, we eventually follow a greedy procedure. However, blindly going through the unordered actions leads us to situations where we may "skip" operations necessary to reach to goals. Additionally, the LP may return values other than "1.0" for actions. Therefore, we have two issues to handle when considering the simulation of action execution to bring us to a better state. Namely, we must deal with cases

[4]The equivalant to what is given by h_{IP}.

where the LP returns non-integer values on the action variables and simultaneously consider how to order the actions given to us.

Using a Planning Graph for Action Order

Though standard relaxed planning graph based heuristics ignore negative effects, they also have the well-established virtue of giving some causal relationships between actions. We exploit this fact and present a method of using the LP to guide relaxed plan extraction in a planning graph that is created by ignoring delete lists. This gives us a set of ordered actions that we may simulate in an effort to reach higher-quality states during search. Note that, though we use the solution to the LP for guiding relaxed plan extraction, this method can be combined with any action and goal selection technique.

Recall that a relaxed planning graph is created by iteratively applying all possible applicable actions given the propositions available, thereby generating a union of the previously available propositions with the ones added by applying the actions. This can provide a cost estimate on reaching a particular proposition by summing the cost of each action applied to reach it, always keeping the minimum summed cost (i.e., we always keep the cheapest cost to reach any proposition). This process is called *cost propagation*. After this, we can extract a relaxed plan from the planning graph by finding the supporting actions for the set of goals. The heuristic value is typically taken from the sum of the cost of all actions in the relaxed plan. If we could extract an optimal relaxed plan the heuristic would be admissible. However, due to the difficulty of this task (which is NP-complete) greedier approaches are generally used (such as preferring to select the cheapest supporting action at each step).

In over-subscription planning we have additional considerations. In particular, we should only extract plans for sets of goals that appear to be beneficial (i.e., provide a high *net benefit*). We can use the LP for this, as it returns a choice of goals. Given that the LP can produce real number values on each variable (in this case a goal variable), we give a threshold, θ_G on their value. For every goal g, there is a value assignment given by the LP, $Value(g)$. If $Value(g) \geq \theta_G$ then we select that goal to be used in the plan extraction process.

The idea for extracting a relaxed plan using the LP as guidance is to prefer those actions that are selected in the LP. When extracting a relaxed plan, we first look at actions supporting propositions that are of the least propagated cost and part of the LP solution. If no such actions support the proposition, we default to the procedure of taking the action with the least propagated cost. Again, since the LP can produce fractional values, we place a threshold on action selection, θ_A. If an action variable $Action(a)$, is greater than the threshold, $action(a) \geq \theta_A$, then that action is preferred in the relaxed plan extraction process given the described procedure. The complete algorithm is shown in Algorithm 1.The key difference between a typical relaxed plan extraction process and our algorithm are lines 11-15, which cause a bias for actions that are in the LP.

To see why the LP makes an impact on the relaxed plans we extract, let us revisit our ongoing example. Figure 2 shows the relaxed planning graph with each action and proposition labelled with the minimum cost for reaching it (using a summing cost propagation procedure). Recall that we want to bias our relaxed plan extraction process toward the actions in the LP because it contains information that the planning graph lacks–namely, negative interactions.

We now see what happens when we follow the algorithm using the example. The LP returns the action set {*fly(loc1, loc2)*, *fly(loc2, loc3)*, *drop(p1, loc2)*}. Also, both goals are chosen by the LP, causing us to place both goals into the set of open conditions (line 5). We have three layers in the graph, and so we progress backward from layer 3 to 1 (line 8). We begin with the least expensive goal and find its cheapest action, *fly(loc1,loc3)*. Since this action is not part of the LP solution (i.e., its value is 0), we move on to the next least expensive supporting action, *fly(loc2,loc3)*. This action is in LP's returned list of actions and therefore it is chosen to satisfy the goal *at(p1,loc3)*. Next, we support the open condition *at(per1,loc2)* with *drop(per1,loc2)*. This action is in the LP. We add the new open condition *at(p1,loc2)* then satisfy it with the action *fly(loc1,loc2)*. We now have the final relaxed plan by reversing the order that the actions were added. Note that without the LP bias we would have the plan {*fly(loc1,loc2)*, *fly(loc1,loc3)*, *drop(per1,loc2)*}, which

Algorithm 1: $ExtractRelaxedPlan_{LP}$, an LP-guided relaxed plan extraction. effect$^+(a)$ represents the set of positive effects of an action a.

Input: Set of all actions above threshold θ_A in LP, A_{LP}; set of all goal assignments above threshold θ_G in LP, G_{LP}; propagated relaxed planning graph with action layers $A_1 \ldots A_n$

1 Initialize RP = {};
2 **for** $i := 1 \ldots \|G\|$ **do**
3 $\quad g := i^{th}$ cheapest goal achieved in planning graph;
4 $\quad$ **if** $g \in G_{LP}$ **then**
5 $\quad\quad OpenConditions := OpenConditions + g$;
6 $\quad$ **end**
7 **end**
8 **for** $i := n \ldots 1$ **do**
9 $\quad$ **forall** $p \in OpenConditions$ **do**
10 $\quad\quad$ p := first(OpenConditions);
11 $\quad\quad$ **if** $\exists a \in A_{i-1} \bigcap A_{LP}$ *such that* $p \in$ *effect*$^+(a)$ **then**
12 $\quad\quad\quad$ Find minimum cost action a in $A_{i-1} \bigcap A_{LP}$ such that $p \in$ *effect*$^+(a)$;
13 $\quad\quad$ **else**
14 $\quad\quad\quad$ Find minimum cost action a in A_{i-1} such that $p \in$ *effect*$^+(a)$;
15 $\quad\quad$ **end**
16 $\quad\quad RP := RP + a$;
17 $\quad$ **end**
18 $\quad$ **forall** $a \in A, p \in pre(a)$ **do**
19 $\quad\quad OpenConditions := OpenConditions + p$;
20 $\quad$ **end**
21 **end**
22 **return** RP

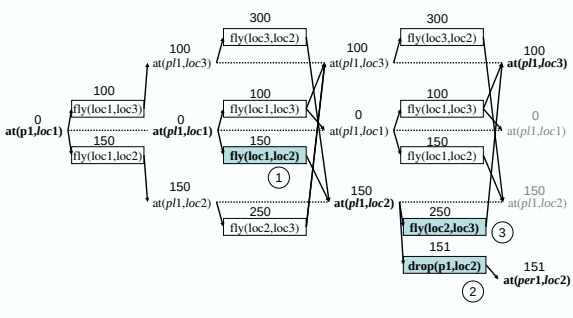

Figure 2: The planning graph for the running example showing LP-biased relaxed plan extraction. Numbers above the actions are their propagated costs. The order of the final relaxed plan is indicated by the circled numbers next to the actions.

is only partially executable in the original planning problem.

Branch and Bound Search

A common method for solving combinatorial problems is to use a branch and bound search where a global bound is kept that represents the objective value of the best feasible solution found so far. In this type of search it is well known that quickly finding a good, close to optimal bound allows for more efficient pruning of the search space. An admissible heuristic is typically used at each search state, s, to determine the potential value that may be found by branching there. Note that our lookahead procedure may give us states that, although high in *net benefit*, do not lead to better solutions (i.e., they are locally optimal but not globally optimal). Branch and bound search allows for situations where we want to continue searching despite having found such a solution and so the algorithm has an *anytime* behavior. This is in contrast to best-first search, which stops after finding a feasible solution. The algorithm, shown in Algorithm 2, requires that we keep a global *lower bound* that provides the value of the currently best-valued node (in terms of total *net benefit* or g-value). This allows us to maintain optimality despite choosing paths that may potentially lead to in-optimal states.

The variable f_{bound} represents the candidate solution with the best *net benefit* found so far by the search. If a search state ever has an f-value that is less than this bound, we can discard the node without looking at it further. Note that for each state, we attach the ordered relaxed plan to it after we calculate the LP-based heuristic.

By simulating the execution of relaxed plans we have the potential to quickly achieve better lower bounds. Algorithm 4 shows how the relaxed plan simulation is done. The procedure works greedily, continually trying to execute relaxed plans in the order that they are given until either the actions in the RP are exhausted (i.e., completely executed) or we cannot apply any more actions from the relaxed plan. We continue to apply the relaxed plans of the resulting state. Similar to what is done in (Yoon *et al.* 2007), we add each state found after performing the lookahead procedure.

This method works with the search by always first attempting to execute the relaxed plan of a node pulled off the queue. In our example we have the relaxed plan

Algorithm 2: Branch and bound search.

```
1  f_bound := net benefit of initial state;
2  LP := LP encoding of the initial state;
3  S := initialstate;
4  S.RP := ExtractRelaxedPlan_lp(LP);
5  SQ := Priority queue initially {S};
6  while SQ ≠ {} do
7      i := 0;
8      S := dequeue(SQ);
9      if f(S) < f_bound then
10         discard(S);
11         continue;
12     end
13     if i = 0 then
14         while S' ≠ S do
15             S' := SimulateRelaxedPlan(S);
16             ProcessState(S');
17             enqueue(SQ, S');
18         end
19     else
           /* Note:  Helpful actions first  */
20         select an action a ∈ A;
21         S' := Apply(a,S);
22     end
23     ProcessState(S');
24     if net_benefit(S') > f_bound then
25         Output_Best_Node(S');
26         f_bound := net_benefit(S');
27     end
28     enqueue(SQ, S);
29     i = i + 1;
30 end
```

{*fly(loc1, loc2)*, *fly(loc2, loc3)*, *drop(p1, loc2)*}. Given the simulation of this relaxed plan from the initial state, the first exploration gives us a high quality search state that achieves the goals (in this case optimally) with a value of 1749 (the *net benefit*). Search continues generating states for each applicable action at the initial state. When we subsequently pull the nodes off of the queue we can discard them. Though this example gives a best case scenario, it serves to show the strength of combining the heuristic, relaxed plan simulation, and search technique. Note that, during the selection of actions, we first attempt so-called *helpful actions*, as used in the planner FF (Hoffmann and Nebel 2001).

With the admissible heuristic and search strategy, the algorithm will, given enough time, return an optimal solution. Otherwise it will return the *best found* solution (which has the same value as f_{bound}). Additionally, the heuristic provides an upper bound measure on the *best possible total net benefit* at the initial state. That is, the heuristic value returned at the initial state provides us with some measure of "how close to optimal" we may be.

Any solution that we are given can be checked against the bound found at the beginning of search (from the initial state). As we shall see in the empirical analysis, this bound provides insight into the quality of the solution found for any

Algorithm 3: *ProcessState*

Input: A state S

updateInitialConstraints(S, LP);
$h_{LP}(S) := solve(LP)$;
$S.f := h_{LP}(S) - cost(S)$;
$S.RP := ExtractRelaxedPlan_{lp}(LP)$;
return *S*;

Algorithm 4: *SimulateRelaxedPlan*, greedily lookahead using a relaxed plan.

Input: A state S with an associated relaxed plan, RP

$executed :=$ boolean array set to false with size $|RP|$;
while *not* $done$ **do**
  $forward := false$;
  for $j = 1 \ldots RP\_size$ **do**
    if *not* $executed[j]$ *and* applicable$(S', RP[j])$ **then**
      $S' := Apply(RP[j], S')$;
      $executed[j] := true$;
      $forward := true$;
      if *net_benefit(S')* $> f_{bound}$ **then**
        Output_Best_Node(S');
        $S'' := S'$;
        f_{bound} := *net_benefit*(S');
      end
    end
  end
  if *not* $forward$ **then**
    $done := true$;
  end
  return S'';
end

given problem.

Empirical Analysis

We created a planner called BBOP-LP (Branch and Bound Over-subscription Planning using Linear Programming, pronounced "bee-bop-a-loop") on top of the framework used for the planner SPUDS (Do *et al.* 2007), which is capable of solving the same type of planning problems and was written in Java 1.5. h_{LP} was implemented using the commercial solver CPLEX 10. All experiments were run on a 3.2 Ghz Pentium D with 1 GB of RAM allocated to the planners.

The system was compared against SPUDS (Do *et al.* 2007) and two of its heuristics, h^{GAI}_{relax} and h^{GAI}_{max}. The branch and bound search is in contrast to SPUDS, which implement inadmissible heuristics that could cause it to stop searching without finding an optimal solution. The heuristic h^{GAI}_{relax} greedily extracts a relaxed plan from its planning graph then uses an IP encoding of the relaxed plan to remove goals that look unpromising. Using this heuristic it also simulates the execution of the final relaxed plan. The other heuristic in SPUDS that we look at, h^{GAI}_{max}, is admissible and performs max cost propagation (i.e., it takes the maximum reachability cost among supporters of any predicate or action) on the planning graph but does not extract a relaxed plan. Instead it uses the propagated costs of the goals and tries to minimize the set using an IP encoding for the goal utility dependencies.

We use the BBOP-LP system using three separate options. Specifically, we use the h_{LP} heuristic without extracting a relaxed plan for simulation, the h_{LP} heuristic with the LP-based heuristic extraction process, and the h_{LP} heuristic with a cost-based heuristic extraction process. Since these methods are used within the BBOP-LP framework, they provide a search that can terminate only when a global optimal solution is found (or time runs out). A goal and action threshold of 0.01 was used.[5] We compare with SPUDS using the h^{GAI}_{relax} and h^{GAI}_{max} heuristics. SPUDS, using an anytime best-first search with the admissible h^{GAI}_{max} heuristic, will terminate when finding an optimal solution (or a timeout). It is possible that SPUDS using the inadmissible h^{GAI}_{relax} heuristic will terminate without having found an optimal solution. We have set SPUDS using h^{GAI}_{relax} to also simulate the execution of the relaxed plan. Each of the planners is run with a time limit of 10 minutes.

Problems

We tested using variants of three domains from the 3^{rd} International Planning Competition (Long and Fox 2002): *zenotravel*, *satellite*, and *rovers*. We generated modified versions of the competition problems such that utilties were added to sets of goals and costs were added to actions within reasonable bounds.

For the domain *zenotravel*, having a person or plane at a goal location always gives a positive utility value. However, having certain groups of people at different locations may generate additional negative or positive utility, with a greater likelihood that it would be negative. For the actions, the cost of boarding and debarking were minimial compared with the cost of flying, zooming, and refueling.

In *satellite*, goals involve having a satellite take images of particular locations. In this domain taking individual images gives large utility values. However, having certain images together could effectively negate the utility achieved from each goal individually (the idea being that we gain redundant information from these images). For this domain, the cost of most of the actions is minor compared to with the utility values. Therefore, this domain is largely about taking utility dependencies into account.

The modified *rovers* domain contains goals for taking measurements and communicating them to some outside location. In this domain an optimal solution may include all of the goals (if not too much navigating is required). However, as the problems grow in size the cost of navigation likely outweighs the utility of the goal sets.

Analysis

Figure 3 shows the results of running the planners in terms of the *net benefit* of the solutions found and the time it took to search for the given solution value. In 13 of the problems, the plans with the highest *net benefit* are found using h_{LP} heuristic with the LP-based relaxed plan lookahead

[5] In our experiments, this threshold provided overall better results over other, higher values for θ_A and θ_G that were tested.

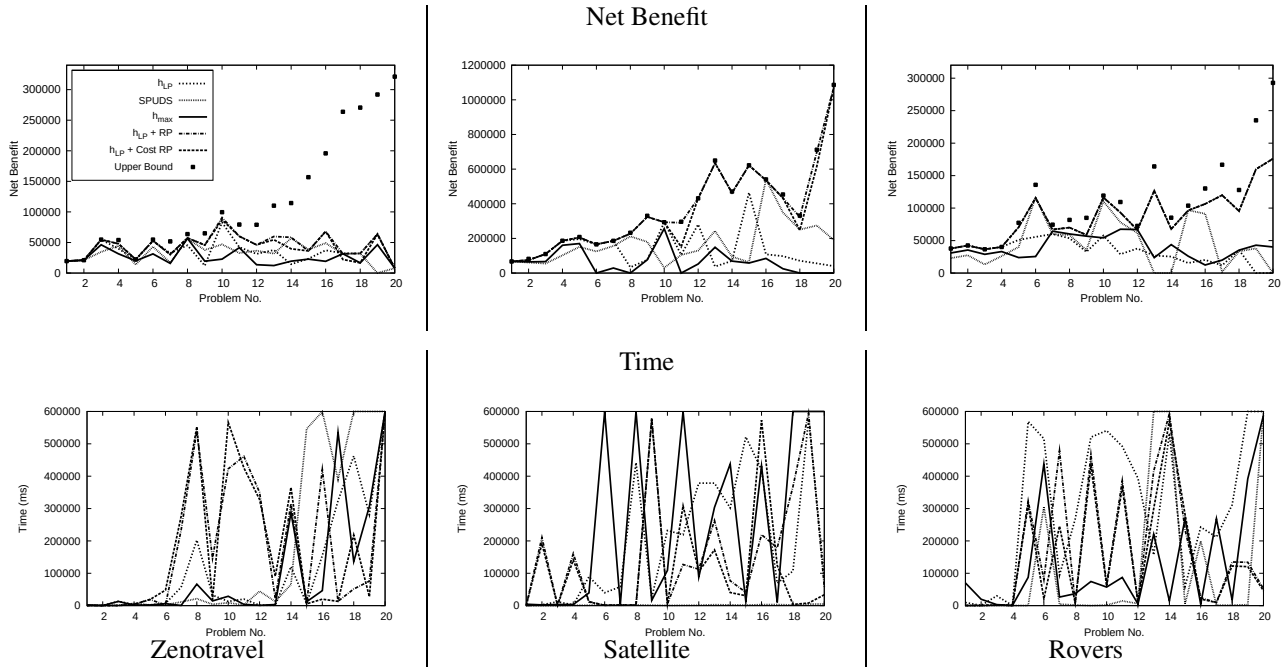

Figure 3: Results for the *zenotravel*, *satellite*, and *rovers* domains in terms of total net benefit. The upper bound value for each problem is found by the LP at the initial state. If a planner could not find a plan beyond the value at the initial state, we set its time to 10 minutes to indicate the wasted search effort.

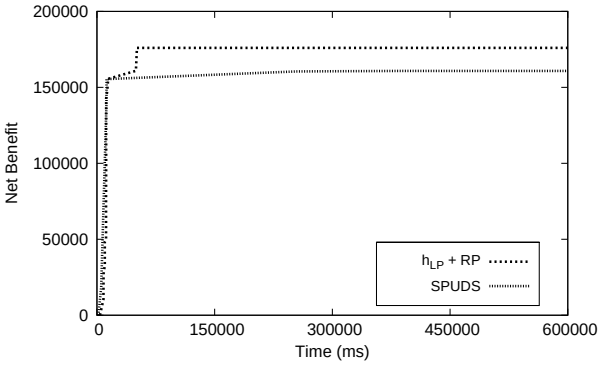

Figure 4: Values found in our branch and bound search using h_{LP} and the LP-based relaxed plan extraction compared with the anytime search of SPUDS for Rovers problem 20. h_{max} represents SPUDS with the h^{GAI}_{max} heuristic.

technique. In fact, in only four of the problem instances is this method returning *net benefit* value less than one of the other methods (*zenotravel* problems 14 through 17). Also, this technique was able find the optimal plan in 15 of the 60 problems (by exhausting the search space). We compare this with SPUDS, which could find the optimal solution in only 2 of the problems within the time limit.

One reason that the LP-based relaxed plan usually performs lookahead better than the cost-based relaxed plan is that it is more informed as to the obvious negative interactions that exist within the problem (e.g., a plane cannot be in more than one place at a time). The heuristic h_{LP} finds flow to the goals on SAS+ variables. As such, only the actions that contribute to achieving a goal will be chosen by the heuristic. In the ongoing example from the previous sections we saw a best-case scenario of this behavior.

The time taken to solve for the final values provides some insight into the search behavior of the planners. SPUDS tends to return plans quickly but then does not continue to find better plans. As can be seen in Figure 3, a consequence of this is that plan *net benefit* remains low, but the time required to achieve that *net benefit* is also low. To provide more insight, Figure 4 shows a search behavior comparison between BBOP-LP with LP-based relaxed plan extraction, BBOP-LP with cost-based relaxed plan extraction, and SPUDS with the h^{GAI}_{relax} heuristic. We can see that SPUDS initially reaches a higher quality plan more quickly, but then makes only incremental improvements. On the other hand, the planner using the LP-based relaxed plan for lookahead continues to find plans whose quality soon surpasses those found by SPUDS.

Another interesting result from this study is the disparity between the domains in terms of quality of the final plans found by BBOP-LP and the upper bound *net benefit* value found by h_{LP} at the initial state (see (Benton *et al.* 2007) for a discussion on the quality of these bounds). We cannot be sure exactly how close to optimal this bound is, but we hypothesize that as the problems scale up in size it becomes less accurate. Looking again at Figure 3, we can see that in *zenotravel* we are quite far from the upper bound found. This gets more pronounced as the problems scale up in difficulty. Another likely contributing factor to the wide margin is the time spent finding h_{LP} at each node. This calculation takes much longer on more difficult problems. In problem 20 of *zenotravel*, for instance, it takes 41 seconds to calculate this value at the initial state on our system. However, on

these problems the lookahead plan simulation process often provides us with solutions that have a fair quality eventually.

We have shown a novel heuristic, h_{LP}, used to guide search in a variant of over-subscription planning called PSP^{UD}. With this, we biased a traditional relaxed plan extraction toward actions included in the heuristic's action selection. This gave us an action ordering to use in a lookahead procedure in a branch and bound search framework. This method was shown to be superior for finding high quality plans over the other tested methods.

Related Work

Much recent work in solving partial satisfaction planning problems use heuristics taken from planning graphs. The planner $AltWlt$ selects goals using a planning graph before the planning process begins (Sanchez and Kambhampati 2005). The planner SPUDS, which introduced the concept of goal utility dependency, refines a relaxed plan found by the planning graph using an IP formulation of it to select goals dynamically during search (Do *et al.* 2007). An approach not using a planning graph is the *orienteering* planner (Smith 2004). Also, several planners that work with preferences as defined in PDDL3 (Gerevini and Long 2005), the language developed for the 5^{th} International Planning Competition (Gerevini *et al.* 2006) use planning graph based heuristics.

The first instance of using an LP-based heuristic in planning was done by Bylander 1997 in the planner Lplan. Also, some planners encode entire classical planning problems as an IP (Vossen *et al.* 1999; van den Briel *et al.* 2005). OptiPlan and iPud, two IP-based planners, solve partial satisfaction planning optimally given a bounded length (Do *et al.* 2007; van den Briel *et al.* 2004).

The idea of relaxed plan execution simulation to find good lower bound values in the branch and bound search was inspired by the great benefits it showed in the YAHSP planner (Vidal 2004). This planner uses the concept of "helpful actions" as used in FF (Hoffmann and Nebel 2001) and extends upon it to use the relaxed plan for performing a lookahead to more quickly feasible solutions to classical planning problems. Since we do this in conjunction with branch and bound search, we retain optimality (unlike YAHSP).

Future Work

An advantage to using LP-based heuristics is that they are malleable. We plan to add or change constraints in the LP encoding used for h_{LP} such that we can achieve better heuristic values more quickly. We also will explore new ways of using the LP with added constraints that give us orderings without the use of a planning graph. Further exploration will involve finding ways to encode PDDL3 temporal constraints.

Acknowledgments: We extend our thanks to Sungwook Yoon and William Cushing for their helpful suggestions during the development of this work. We also would like to thank the anonymous reviewers for their helpful feedback. This research is supported in part by the ONR grant N000140610058, a Lockheed Martin subcontract TT0687680 to ASU as part of the DARPA Integrated Learning program, and the NSF grant IIS-308139.

References

F. Bacchus and A. Grove. Graphical model for preference and utility. In *Proceedings of the 11th Conference on Uncertainty in Artifi cial Intelligence*, pages 3–10, 1995.

C. Bäckström and B. Nebel. Complexity results for SAS$^+$ planning. *Computational Intelligence*, 11(4):625–655, 1995.

J. Benton, M. van den Briel, and S. Kambhampati. Finding admissible bounds for over-subscription planning problems. In *Workshop on Heuristics for Domain-independent Planning: Progress, Ideas, Limitations, Challenges, ICAPS-2007*. submitted, 2007.

T. Bylander. A linear programming heuristic for optimal planning. In *Proceedings of the 14th National Conference on Artifi cial Intelligence (AAAI-97)*, pages 165–204, 1997.

M.B. Do, J. Benton, M. van den Briel, and S. Kambhampati. Planning with goal utility dependencies. In *Proceedings of the 20th International Joint Conference on Artifi cial Intelligence (IJCAI 2007)*, pages 1872–1878, 2007.

S. Edelkamp and M. Helmert. Exhibiting knowledge in planning problems to minimize state encoding length. In *5th European Conference on Planning*, volume 1809, 1999.

A. Gerevini and D. Long. Plan constraints and preferences in PDDL3: The language of the fifth international planning competition. Technical report, University of Brescia, Italy, 2005.

A. Gerevini, Y. Dimopoulos, P. Haslum, and A. Saetti. The Fifth International Planning Competition (ipc5). http://zeus.ing.unibs.it/ipc-5/, 2006.

J. Hoffmann and B. Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artifi cial Intelligence Research*, 14:253–302, 2001.

D. Long and M. Fox. The Third International Planning Competition (ipc3). http://planning.cis.strath.ac.uk/competition/, 2002.

R. Sanchez and S. Kambhampati. Planning graph heuristics for selecting objectives in over-subscription planning problems. In *Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS-2005)*, pages 192–201, 2005.

D. Smith. Choosing objectives in over-subscription planning. In *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS-2004)*, pages 393–401, 2004.

M. van den Briel, R. Sanchez, M.B. Do, and S. Kambhampati. Effective approaches for partial satisfaction (oversubscription) planning. In *Proceedings of the 19th National Conference on Artifi cial Intelligence (AAAI-2004)*, pages 562–569, 2004.

M. van den Briel, T. Vossen, and S. Kambhampati. Reviving integer programming approaches for AI planning. In *Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS-2005)*, pages 310–319, 2005.

M. van den Briel, J. Benton, and S. Kambhampati. An LP-based heuristic for optimal planning. In *International Conference on Principles and Practice of Constraint Programming (CP)*, 2007.

V. Vidal. A lookahead strategy for heuristic search planning. In *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS-2004)*, pages 150–159, 2004.

T. Vossen, M. Ball, A. Lotem, and D.S. Nau. On the use of integer programming models in AI planning. In *Proceedings of the 18th International Joint Conference on Artifi cial Intelligence (IJCAI-1999)*, pages 304–309, 1999.

S. Yoon, A. Fern, and R. Givan. Using learned policies in heuristic-search planning. In *International Joint Conference on Artifi cial Intelligence*, 2007.