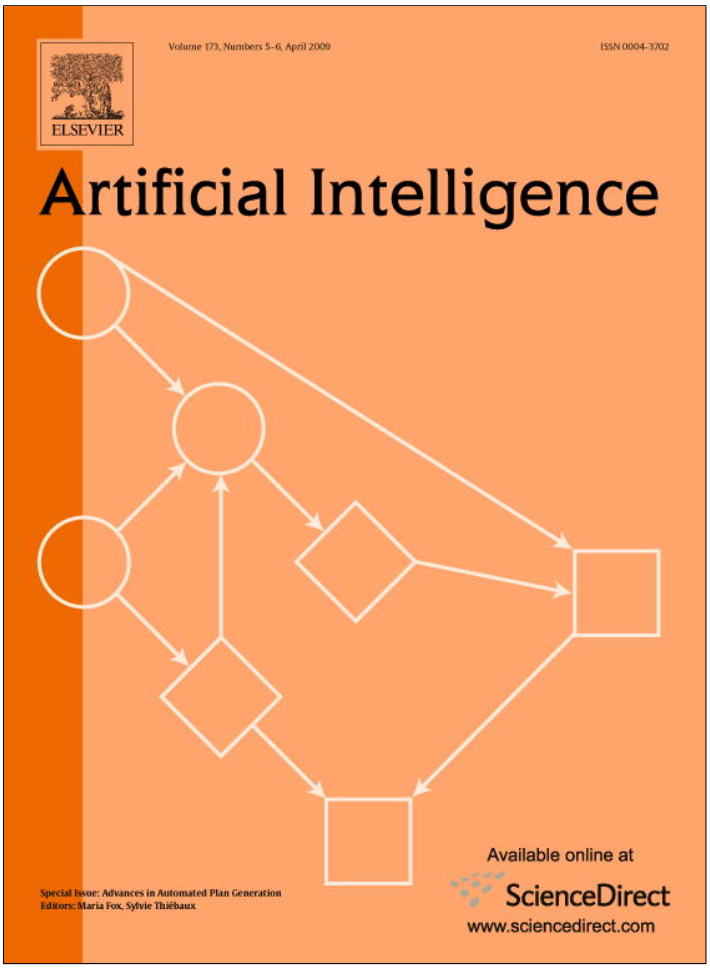


Artificial Intelligence 173 (2009) 562–592

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

Anytime heuristic search for partial satisfaction planning

J. Benton [a,*], Minh Do [b], Subbarao Kambhampati [a]

[a] Arizona State University, Department of Computer Science and Engineering Brickyard Suite 501, 699 South Mill Avenue, Tempe, AZ 85281, USA
[b] Embedded Reasoning Area, Palo Alto Research Center 3333 Coyote Hill Road, Palo Alto, CA 94304, USA

ARTICLE INFO

Article history:
Received 28 October 2007
Received in revised form 6 November 2008
Accepted 6 November 2008
Available online 28 November 2008

Keywords:
Planning
Heuristics
Partial satisfaction
Search

ABSTRACT

We present a heuristic search approach to solve partial satisfaction planning (PSP) problems. In these problems, goals are modeled as soft constraints with utility values, and actions have costs. Goal utility represents the value of each goal to the user and action cost represents the total resource cost (e.g., time, fuel cost) needed to execute each action. The objective is to find the plan that maximizes the trade-off between the total achieved utility and the total incurred cost; we call this problem PSP Net Benefit. Previous approaches to solving this problem heuristically convert PSP Net Benefit into STRIPS planning with action cost by pre-selecting a subset of goals. In contrast, we provide a novel anytime search algorithm that handles soft goals directly. Our new search algorithm has an anytime property that keeps returning better quality solutions until the termination criteria are met. We have implemented this search algorithm, along with relaxed plan heuristics adapted to PSP Net Benefit problems, in a forward state-space planner called $Sapa^{PS}$. An adaptation of $Sapa^{PS}$, called $Yochan^{PS}$, received a "distinguished performance" award in the "simple preferences" track of the 5th International Planning Competition.

1. Introduction

In classical planning, the aim is to find a sequence of actions that transforms a given initial state $\mathcal{I}$ to some state satisfying goals $\mathcal{G}$, where $\mathcal{G} = g_1 \wedge g_2 \wedge \cdots \wedge g_n$ 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. In many real world scenarios, however, the best plan for the agent may only satisfy a subset of the goals. The need for such partial satisfaction planning might arise in some cases because the set of goal conjuncts may contain logically conflicting fluents, or there may not be enough time or resources to achieve all of the goal conjuncts, or achieving all goals may prove to be too costly.[1]

Despite their ubiquity, PSP problems have only recently garnered attention. Effective handling of 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. In this paper, we focus on one of the more general PSP problems, called PSP Net Benefit. In this problem, each goal conjunct has

* Corresponding author.
E-mail address: j.benton@asu.edu (J. Benton).

[1] Smith [40] first introduced the term Over-Subscription Planning (OSP), and later van den Briel et al. [44] used the term Partial Satisfaction Planning (PSP) to describe the type of planning problem where goals are modeled as soft constraints and the planner aims to find a good quality plan achieving only a subset of goals. While the problem definition is the same, OSP and PSP have different perspectives: OSP emphasizes the resource limitations as the root of partial goal satisfaction (i.e., the planner *cannot* achieve all goals), and PSP concentrates on the tradeoff between goal achievement costs and overall achieved goal utility (i.e., even when possible, the achievement of all goals is not the best solution). We will use the term PSP throughout this paper because our planning algorithm is targeted to find the plan with the best tradeoff between goal utility and action cost.

0004-3702/$ – see front matter
doi:10.1016/j.artint.2008.11.010

a fixed utility and each ground action has a fixed cost. All goal utilities and action costs are independent of one another.[2] The objective is to find a plan with the best "net benefit" (i.e., cumulative utility minus cumulative cost). Hence the name PSP NET BENEFIT.

One obvious way of solving the PSP NET BENEFIT problem is to model it as a deterministic Markov Decision Process (MDP) with action cost and goal reward [9]. Each state that achieves one or more of the goal fluents is a terminal state in which the reward equals the sum of the utilities of the goals in that state. The optimal solution of the PSP NET BENEFIT problem can be obtained by extracting a plan from the optimal policy for the corresponding MDP. However, our earlier work [44] showed that the resultant approaches are often too inefficient (even with state-of-the-art MDP solvers). Consequently, we investigate an approach of modeling PSP in terms of heuristic search with cost-sensitive reachability heuristics. In this paper, we introduce $Sapa^{\mathcal{PS}}$, an extension of the forward state-space planner *Sapa* [13], to solve PSP NET BENEFIT problems. $Sapa^{\mathcal{PS}}$ adapts powerful relaxed plan-based heuristic search techniques, which are commonly used to find satisficing plans for classical planning problems [31], to PSP NET BENEFIT. Expanding heuristic search to find such plans poses several challenges of its own:

- The planning search termination criteria must change because goals are now soft constraints (i.e., disjunctive goal sets).
- Heuristics guiding planners to achieve a fixed set of goals with uniform action costs need to be adjusted to actions with non-uniform cost and goals with different utilities.

$Sapa^{\mathcal{PS}}$ develops and uses a variation of an anytime A^* search algorithm in which the most beneficial subset of goals S_G and the lowest cost plan P achieving them are estimated for each search node. The trade-off between the total utility of S_G and the total cost of P is then used as the heuristic to guide the search. The anytime aspect of the $Sapa^{\mathcal{PS}}$ search algorithm comes naturally with the switch from goals as hard constraints in classical planning to goals as soft constraints in PSP NET BENEFIT. In classical planning, there are differences between valid plans leading to states satisfying all goals and invalid plans leading to states where at least one goal is not satisfied. Therefore, the path from the initial state I to any node generated during the heuristic search process before a goal node is visited cannot be returned as a valid plan. In contrast, when goals are soft constraints, a path from I to any node in PSP NET BENEFIT represents a valid plan and plans are only differentiated by their qualities (i.e., net benefit). Therefore, the search algorithm can take advantage of that distinction by continually returning better quality plans as it explores the search space, until the optimal plan (if an admissible heuristic is used) is found or the planner runs out of time or memory. This is the approach taken in the $Sapa^{\mathcal{PS}}$ search algorithm: starting from the empty plan, whenever a search node representing a better quality plan is visited, it is recorded as the current best plan found and returned to the user when the algorithm terminates.[3]

The organizers of the 5th International Planning Competition (IPC-5), also realizing the importance of soft goals, introduced soft constraints with violation cost into PDDL3.0 [23]. Although this model is syntactically different from PSP NET BENEFIT, we derive a novel compilation technique to transform "simple preferences" in PDDL3.0 to PSP NET BENEFIT. The result of the conversion is then solved by our search framework in an adaptation of $Sapa^{\mathcal{PS}}$ called $Yochan^{\mathcal{PS}}$, which received a "distinguished performance" award in the IPC-5 "simple preferences" track.

In addition to describing our algorithms and their performance in the competition, we also provide critical analysis on two important design decisions:

- Does it make sense to compile the simple preference problems of PDDL3.0 into PSP NET BENEFIT problems (rather than compile them to cost-sensitive classical planning problems as is done by some competing approaches, such as [18,33])?
- Does it make sense to solve PSP NET BENEFIT problems with anytime heuristic search (rather than by selecting objectives up-front, as advocated by approaches such as [40,44])?

To analyze the first we have compared $Yochan^{\mathcal{PS}}$ with a version called $Yochan^{COST}$, which compiles the PDDL3.0 simple preference problems into pure cost-sensitive planning problems. Our comparison shows that $Yochan^{\mathcal{PS}}$ is superior to $Yochan^{COST}$ on the competition benchmarks.

Regarding the second point, in contrast to $Sapa^{\mathcal{PS}}$ (and $Yochan^{\mathcal{PS}}$), several other systems (e.g., $AltAlt^{\mathcal{PS}}$ [44] and the orienteering planner [40]) solve PSP problems by first selecting objectives (goals) that the planner should work on, and then simply supporting them with the cheapest plan. While such an approach can be quite convenient, a complication is that objective selection can, in the worst case, be as hard as the overall planning problem. To begin with, heuristic up-front selection of objectives automatically removes any guarantees of optimality of the resulting solution. Second, and perhaps more important, the goals selected may be infeasible together thus making it impossible to find even a *satisficing* solution. To be sure, there has been work done to make the objective selection more sensitive to mutual exclusions between the goals. An example of such work is *AltWlt* [38] which aims to improve the objective selection phase of $AltAlt^{\mathcal{PS}}$ with some

[2] In [12], we discuss the problem definition and solving approaches for a variation of PSP NET BENEFIT where there are dependencies between goal utilities.

[3] Note that the anytime behavior is dependent on the PSP problem at hand. If we have many goals that are achievable with low costs, then we will likely observe the anytime behavior as plans with increasing quality are found. However, if there are only a few goals and they are difficult to achieve or mostly unbeneficial (i.e., too costly), then we will not likely see many incrementally better solutions.

limited mutual exclusion analysis. We shall show, however, that this is not enough to capture the type of n-ary mutual exclusions that result when competition benchmark problems are compiled into PSP NET BENEFIT problems.

In summary, our main contributions in this paper are:

- Introducing and analyzing an anytime search algorithm for solving PSP NET BENEFIT problems. Specifically we analyze the algorithm properties, termination criteria and how to manage the search space.
- An approach to adapt the relaxed plan heuristic in classical planning to PSP NET BENEFIT; this involves taking into account action costs and soft goal utility in extracting the relaxed plan from the planning graph.
- A novel approach to convert "simple preferences" in PDDL3.0 to PSP NET BENEFIT; how to generate new goals and actions and how to use them during the search process.

The rest of this paper is organized into three main parts. To begin, we discuss background information on heuristic search for planning using relaxed plan based heuristics in Section 2. Next, we discuss $Sapa^{\mathcal{PS}}$ in Section 3. This section includes descriptions of the anytime algorithm and the heuristic guiding $Sapa^{\mathcal{PS}}$. In Section 4, we show how to compile PDDL3.0 "simple preferences" to PSP NET BENEFIT in $Yochan^{\mathcal{PS}}$. In Section 5, we show the performance of $Yochan^{\mathcal{PS}}$ against other state-of-the-art planners in the IPC-5 [22]. We finish the paper with the related work in Section 6, and our conclusions and future work in Section 7.

2. Background

2.1. PSP net benefit problem

In partial satisfaction planning with net benefit (PSP NET BENEFIT) [40,44], each goal $g \in G$ has a utility value $u_g \geqslant 0$, representing how much each goal is worth to a user; each action $a \in A$ has an associated execution cost $c_a \geqslant 0$, representing how costly it is to execute each action (e.g., the amount of time or resources consumed). All goals are *soft constraints* in the sense that any plan achieving a subset of goals (even the empty set) is a valid plan. Let $\mathbb{P}$ be the set of all valid plans and let $G_P \subseteq G$ be the set of goals achieved by a plan $P \in \mathbb{P}$. The objective is to find a plan P that maximizes the difference between total achieved utility $u(G_P)$ and total cost of all actions $a \in P$:

$$\underset{P \in \mathbb{P}}{\operatorname{argmax}} \sum_{g \in G_P} u_g - \sum_{a \in P} c_a \tag{1}$$

Example. In Fig. 1, we introduce a logistics example that we will use throughout the paper. In this example, there is a truck loaded with three packages that initially resides at location A. The (soft) goals are to deliver one package each to the other four locations: B, C, D, and E. The respective goal utilities are shown in Fig. 1. For example: $g_1 = HavePackage(B)$ and $u_{g_1} = 50$. To simplify the problem, we assume that there is a single $Move(X, Y)$ action that moves the truck from location X to location Y. The truck will always drop a package in Y if there is no package at Y already. Thus, if there is a package in the truck, it will drop it at Y. If there is no package in the truck, but one at X, then the truck will pick it up and drop it off at Y when executing $Move(X, Y)$. The costs of moving between different locations are also shown in Fig. 1. For example, with $a_1 = Move(A, B)$, $c_{a_1} = 40$ (note that $c_{Move(X,Y)} = c_{Move(Y,X)}$). The objective function is to find a plan that gives the best trade-off between the achieved total goal utility and moving cost. In this example, the optimal plan is $P = \{Move(A, C), Move(C, D), Move(D, E)\}$ that achieves the last three goals g_2, g_3, g_4 and ignores the first goal $g_1 = HavePackage(B)$.

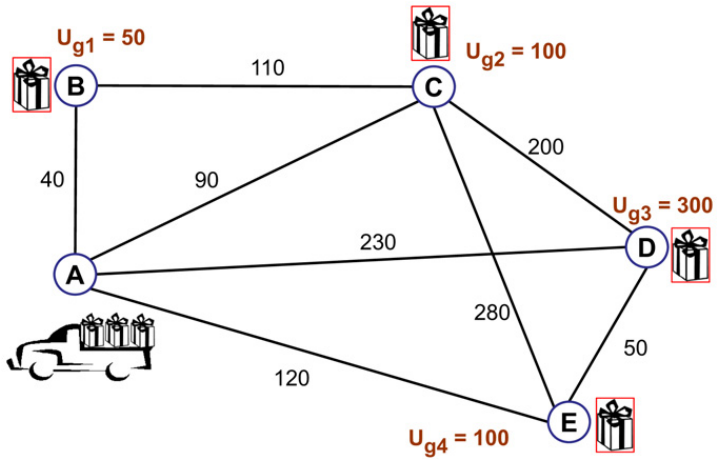


Fig. 1. A logistics example.

We call this utility and cost trade-off value "net benefit" and thus this class of PSP problem PSP NET BENEFIT. As investigated in [44], though it is considered harder due to the additional complications of *soft* goals with utility and action costs, PSP NET BENEFIT still falls into the same complexity class as classical planning (PSPACE). However, existing search techniques typically do not handle soft goals directly and instead either choose a subset of goals up-front or compile all soft goals into hard goals [18,32,33,44]. These approaches have several disadvantages. For instance, when selecting goals up-front, one runs the risk of choosing goals that cannot be achieved together because they are mutually exclusive. In our example, the set of all four goals: $\{g_1, g_2, g_3, g_4\}$ are mutual exclusive because we only have three packages to be delivered to four locations. Any planner that does not recognize this, and selects all four goals and converts them to hard goals is doomed to fail. Compiling to hard goals increases the number of actions to be handled and can fail to allow easy changes in goal selection (e.g., in cases where soft goals do not remain true after achievement such as in our example where a delivered package may need to be picked up again and delivered to another location). Instead, the two progression planners *Sapa*$^{\mathcal{PS}}$ and *Yochan*$^{\mathcal{PS}}$ discussed in this paper handle soft goals directly by heuristically selecting different goal sets for different search nodes. Those planners extend the relaxed plan heuristic used in the *Sapa* planner to the PSP NET BENEFIT setting.

2.2. Relaxed plan heuristics

Before exploring the details of this approach, we give an overview of solving classical planning problems with forward state-space search using relaxed plan based heuristics. This is the same approach used in our planners.

Forward state space search is used by many state-of-the-art planners such as HSP [7], FF [31], and Fast Downward [29]. They build plans incrementally by adding one action at a time to the plan prefix starting from the initial state I until reaching a state containing all goals. An action a is applicable (executable) in state s if $Pre(a) \subseteq s$ and applying a to s results in a new state $s' = (s \setminus Delete(a)) \cup Add(a)$. State s is a goal node (satisfies all goals) if: $G \subseteq s$. Algorithm 1 depicts the basic steps of this forward state space search algorithm. At each search step, let A_s be the set of actions applicable in s. Different search algorithms choose to apply one or more actions $a \in A_s$ to generate new search nodes. For example, the enforced hill-climbing search algorithm in the FF planner chooses one successor node while the best-first search algorithm used in *Sapa* [13] and HSP [7] applies all actions in A_s to generate new search nodes. Generated states are stored in the search queue and the most common sorting function (Line 6) is in the decreasing order of $f(s) = g(s) + w \cdot h(s)$ values where $g(s)$ is the distance from I to s, $h(s)$ is the expected distance from s to the goal state G, and w is a "weight" factor. The search stops when the first node taken from the queue satisfies all the pre-defined conjunctive goals.

For forward planners using a best-first search algorithm that sorts nodes according to the $f = g + w \cdot h$ value, the "informedness" of the heuristic value h is critical to the planner's performance. The h and g values are measured according to the user's objective function. Let $P_{I \to s}$ be the partial plan leading from the initial state I to a state s and $P_{s \to G}$ be a plan leading from a state s to a goal state G. If the objective function is to minimize the number of actions in the final plan, then g measures the number of actions in $P_{I \to s}$ and h measures the expected number of actions in the shortest $P_{s \to G}$.

Measuring g is easy because by reaching s we already know $P_{I \to s}$. However, measuring h exactly (i.e., h^*) is as hard as solving the remaining problem of finding a plan from s to G. Therefore, finding a good approximation of h^* in a short time is critical. In recent years, many of the most effective domain-independent planners such as FF [31], LPG [25], and *Sapa* [13] have used the relaxed plan to approximate h. For the rest of this section, we will briefly describe a general approach of extracting the relaxed plan to approximate the shortest length plan (i.e., least number of actions) for achieving the goals.

One way to *relax* a planning problem is to ignore all of the negative effects of actions. A solution P^+ to a relaxed planning problem (F, A^+, I, G), where A^+ is built from A by removing all negative effects $Delete(a)$ from all actions $a \in A$,

```
1  Input: A planning problem: ⟨F, I, G, A⟩;
2  Output: A valid plan;
3  begin
4      OPEN ← {I};
5      while OPEN ≠ ∅ do
6          s ← argmin_{x∈OPEN} f(x);
7          OPEN ← OPEN \ {s};
8          if s ⊨ G then
9              return plan leading to s and stop search;
10         else
11             foreach a applicable in s do
12                 OPEN ← OPEN ∪ {Apply(a, s)}
13             end
14         end
15     end
16 end
```

Algorithm 1. Forward state space planning search algorithm.

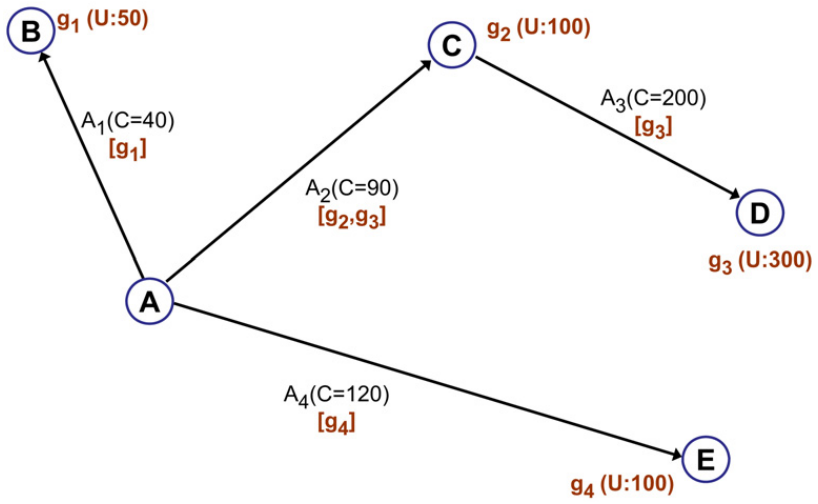


Fig. 2. The relaxed planning graph.

can be found in polynomial time in terms of the number of actions and facts. P^+ is called a "relaxed plan". The common approach to extract P^+ involves two steps:

(1) Build the relaxed planning graph PG^+ using A^+ forward from I.
(2) Extract the relaxed plan $P+$ from PG^+ backward from G.

In the first step, action and fact levels are interleaved starting from $I = F_0$ as the first fact level. The i^{th} action and fact levels A_i and F_i are built iteratively using the following rules:

$$A_i = \bigcup \{a:\ Pre(a) \subseteq F_{i-1}\},$$

$$F_i = F_{i-1} \cup \left(\bigcup_{a \in A_i} Add(a) \right)$$

Using these equations, we generate a procedure where the expansion stops when $G \subseteq F_i$. We start with the subgoal set $SG = G$ at the last level i where the expansion stopped and the relaxed plan $P^+ = \emptyset$. For each $g_i \in SG$ (i.e., goal g at level i) we select a supporting action a_i (i.e., action a at level i) such that: $g \in Effect(a)$ and update $P^+ \leftarrow P^+ \cup \{a_i\}$, $SG \leftarrow SG \cup \{p_{i-1}:\ p \in Pre(a)\}$. Note that subgoal g_i can also be supported by a *noop* action at level i if g appears in the graph at level $i-1$. In this case, we just replace g_i in SG by g_{i-1}. We repeat the procedure until $SG \subseteq F_0$.[4] The details of the original relaxed plan finding algorithm can be found in the description of the FF planner [31]. Note also that the relaxed plan greedily extracted as described above is not optimal.

Fig. 2 shows the relaxed planning graph for our ongoing example. The first fact level represents the initial state in which only $at(A)$ is true. The first action level contains actions applicable in the initial state, which are four *Move* actions from A to the other four locations. The second fact level contains effects of actions in the first level. The second action level contains actions with preconditions satisfied by the second fact level and thus contains all actions in this problem.[5] One example of a relaxed plan extracted from this graph (highlighted in Fig. 2) would be $P^+ = \{Move(A,B), Move(A,C), Move(A,E), Move(C,D)\}$. We can see that this relaxed plan is not consistent due to the mutual exclusion relation between $Move(A,C)$, $Move(A,B)$, and $Move(A,E)$. However, it can be used as an estimate for a real consistent plan. Thus, if the objective function is to find the minimum cost plan, then the estimated cost based on the relaxed plan extracted in Fig. 2 for the initial state is $h(I) = c_{Move(A,B)} + c_{Move(A,C)} + c_{Move(A,E)} + c_{Move(C,D)} = 40+90+120+200 = 450$. If the objective function is to find the shortest length plan then $|P^+| = 4$ can be used as the heuristic value.

3. *Sapa*$^{\mathcal{PS}}$: Forward state-space heuristic planner for PSP

In this section, we will discuss our approach for extending the forward state space search framework to handle PSP Net Benefit problems. Our search and heuristic techniques have been implemented in a planner called *Sapa*$^{\mathcal{PS}}$. We start in Section 3.1 by introducing a variation of the best-first search algorithm that solves the PSP Net Benefit problem. We then move on to the relaxed plan based heuristic for PSP Net Benefit in Section 3.2.

[4] In our implementation, besides the level at which each action appears, we also record the causal links $a \xrightarrow{g} a'$ if a is selected (at the lower level) to support the precondition g of a' at the later level. These causal links in effect represent P^+ as a partial-order plan.

[5] We exclude actions that have already appeared in the first level, as well as several *Move* actions from D and E to simplify the figure.

3.1. Anytime best-first search algorithm for PSP

One of the most popular methods for solving planning problems is to cast them as the problem of searching for a minimum cost path in a graph, then use a heuristic search to find a solution. Many of the most successful heuristic planners [7,13,31,37,41] employ this approach and use variations of best-first graph search (BFS) algorithms to find plans. We also use this approach to solve PSP Net Benefit problems. In particular, we use a variation of A^* with modifications to handle some special properties of PSP Net Benefit (e.g., any state can be a goal state). For the remainder of this section, we will outline them and discuss our search algorithm in detail.

Standard shortest-path graph search algorithms search for a minimum-cost path from a start node to a goal node. Forward state space search for solving classical planning problems can be cast as a graph search problem as follows: (1) each search node n represents a complete planning state s; (2) if applying action a to a state s leads to another state s' then action a represents a directed edge $e = s \xrightarrow{a} s'$ from s to s' with the edge cost $c_e = c_a$; (3) the start node represents the initial state I; (4) a goal node is any state s_G satisfying all goals $g \in G$. In our ongoing example, at the initial state $I = \{at(A)\}$, there are four applicable actions $a_1 = Move(A, B)$, $a_2 = Move(A, C)$, $a_3 = Move(A, D)$, and $a_4 = Move(A, E)$ that lead to four states $s_1 = \{at(B), g_1\}$, $s_2 = \{at(C), g_2\}$, $s_3 = \{at(D), g_3\}$, and $s_4 = \{at(E), g_4\}$. The edge costs will represent action costs in this planning state-transition graph[6] and the shortest path in this graph represents the lowest cost plan. Compared to the classical planning problem, the PSP Net Benefit problem differs in the following ways:

- Not all goals need to be accomplished in the final plan. In the general case where all goals are *soft*, any executable sequence of actions is a candidate plan (i.e., any node can be a valid goal node).
- Goals are not uniform and have different utility values. The plan quality is not measured by the total action cost but by the difference between the cumulative utility of the goals achieved and the cumulative cost of the actions used. Thus, the objective function shifts from *minimizing* total action cost to *maximizing* net benefit.

To cast PSP Net Benefit as a graph search problem, we need to make some modifications to (1) the edge weight representing the change in plan benefit by going from a search node to its successors and (2) the criteria for terminating the search process. We will first discuss the modifications, then present a variation of the A^* search algorithm for solving the graph search problem for PSP. To simplify the discussion and to facilitate proofs of certain properties of this algorithm, we will make the following assumptions: (1) all goals are soft constraints; (2) the heuristic is admissible. Later, we will provide discussions on relaxing one or more of those assumptions.

g value: A^* uses the value $f(s) = g(s) + h(s)$ to rank generated states s for expansion with g representing the "value" of the (known) path leading from the start state I to s, and h estimating the (unknown) path leading from s to a goal node that will optimize a given objective function. In PSP Net Benefit, g represents the additional benefit gained by traveling the path from I to s. For a given state s, let $G_s \subseteq G$ be the set of goals accomplished in s, then:

$$g(s) = \big(U(s) - U(I)\big) - C(P_{I\to s}) \tag{2}$$

where $U(s) = \sum_{g\in G_s} u_g$ and $U(I) = \sum_{g\in G_I} u_g$ are the total utility of goals satisfied in s and I. $C(P_{I\to s}) = \sum_{a\in P_{I\to s}} c_a$ is the total cost of actions in $P_{I\to s}$. For example: $U(s_2) = u_{g_2} = 100$, and $C(P_{I\to s_2}) = c_{a_2} = 90$ and thus $g(s_2) = 100 - 90 = 10$.

In other words, $g(s)$ as defined in Eq. (2) represents the additional benefit gained when plan $P_{I\to s}$ is executed in I to reach s. To facilitate the later discussion, we will introduce a new notation to represent the benefit of a plan P leading from a state s to another state s':

$$B(P|s) = \big(U(s') - U(s)\big) - \sum_{a\in P} c_a \tag{3}$$

Thus, we have $g(s) = B(P_{I\to s}|I)$.

h value: In graph search, the heuristic value $h(s)$ estimates the path from s to the "best" goal node. In PSP Net Benefit, the "best" goal node is the node s_g such that traveling from s to s_g will give the most additional benefit. In general, the closer that h estimates the real optimal h^* value, the better in terms of the amount of search effort. Therefore, we will first provide the definition of h^*.

Best beneficial plan. For a given state s, a best beneficial plan P_s^B is a plan executable in s and there is no other plan P executable in s such that: $B(P|s) > B(P_s^B|s)$.

Notice that an empty plan $P_\emptyset$ containing no actions is applicable in all states and $B(P_\emptyset|s) = 0$. Therefore, $B(P_s^B|s) \geqslant 0$ for any state s. The optimal additional achievable benefit of a given state s is calculated as follows:

$$h^*(s) = B\big(P_s^B|s\big) \tag{4}$$

[6] In the simplest case where actions have no cost and the objective function is to minimize the number of actions in the plan, we can consider all actions having uniform positive cost.

```
1  Input: A PSP problem: ⟨F, I, G, A⟩;
2  Output: A valid plan P_B;
3  begin
4      g(I) ← Σ_{g∈I} u_g;
5      f(I) ← g(I) + h(I);
6      B_B ← g(I);
7      P_B ← ∅;
8      OPEN ← {I};
9      while OPEN ≠ ∅ and not interrupted do
10         s ← argmax_{x∈OPEN} f(x);
11         OPEN ← OPEN \ {s};
12         if h(s) = 0 then
13             stop search;
14         else
15             foreach s' ∈ Successors(s) do
16                 if g(s') > B_B then
17                     P_B ← plan leading from I to s';
18                     B_B ← g(s');
19                     OPEN ← OPEN \ {s_i : f(s_i) ≤ B_B};
20                 end
21                 if f(s') > B_B then
22                     OPEN ← OPEN ∪ {s'}
23                 end
24             end
25         end
26     end
27     Return P_B;
28 end
```

Algorithm 2. Anytime A* search algorithm for finding maximum beneficial plan for PSP (without duplicate detection).

In our ongoing example, from state s_2, the most beneficial plan is $P^B_{s_2} = \{Move(C, D), Move(D, E)\}$, and $h^*(s_2) = B(P^B_{s_2}|s_2) = U(\{g_3, g_2, g_4\}) - U(\{g_2\}) - (c_{Move(C,D)} + c_{Move(D,E)}) = ((300 + 100 + 100) - 100) - (200 + 50) = 400 - 250 = 150$. Computing h^* directly is impractical as we need to search for P^B_s in the space of all potential plans and this is as hard as solving the PSP NET BENEFIT problem for the current search state. Therefore, a good approximation of h^* is needed to effectively guide the heuristic search algorithm. In the next section, we will discuss a heuristic approach to approximating P^B_s using a relaxed plan.

Algorithm 2 describes the anytime variation of the A^* algorithm that we used to solve the PSP NET BENEFIT problems. Like A^*, this algorithm uses the value $f = g + h$ to rank nodes to expand, with the successors generator and g and h values described above. We assume that the heuristic used is *admissible*. Because we try to find a plan that maximizes *net benefit*, admissibility means over-estimating additional achievable benefits; thus, $h(s) \geqslant h^*(s)$ with $h^*(s)$ defined above. Like other anytime algorithms, we keep one incumbent value B_B to indicate the quality of the best found solution at any given moment (i.e., highest net benefit).[7]

The search algorithm starts with the initial state I and keeps expanding the most promising node s (i.e., one with highest f value) picked from the *OPEN* list (Line 7). If $h(s) = 0$ (i.e., the heuristic estimate indicates that there is no additional benefit gained by expanding s) then we stop the search. This is true for the termination criteria of the A^* algorithm (i.e., where the goal node gives $h(s) = 0$). If $h(s) > 0$, then we expand s by applying applicable actions a to s to generate all successors.[8] If the newly generated node s' has a better $g(s')$ value than the best node visited so far (i.e., $g(s') > B_B$), then we record $P_{s'}$ leading to s' as the new best found plan. Finally, if $f(s') \leqslant B_B$ (i.e., the heuristic estimate indicates that expanding s' will never achieve as much additional benefit to improve the current best found solution), we will discard s' from future consideration. Otherwise s' is added to the *OPEN* list. Whenever a better solution is found (i.e., the value of B_B increases), we will also remove all nodes $s_i \in OPEN$ such that $f(s_i) \leqslant B_B$. When the algorithm is interrupted (either by reaching the time or memory limit) before the node with $h(s) = 0$ is expanded, it will return the best plan P_B recorded so far (the alternative approach is to return a new best plan P_B whenever the best benefit value B_B is improved). Thus, compared

[7] Algorithm 2, as implemented in our planners, does not include duplicate detection (i.e., no *CLOSED* list). However, it is quite straightforward to add duplicate detection to the base algorithm similar to the way *CLOSED* list is used in A^*.

[8] Note that with the assumption of $h(s)$ being admissible, we have $h(s) \geqslant 0$ because it overestimates $B(P^B_s|s) \geqslant 0$.

to A^*, this variation is an "anytime" algorithm and always returns some solution plan regardless of the time or memory limit.

Like any search algorithm, one desired property is preserving optimality. We will show that if the heuristic is admissible, then the algorithm will find an optimal solution if given enough time and memory.[9]

Proposition 1. *If h is admissible and bounded, then Algorithm 2 always terminates and the returned solution is optimal.*

Proof. Given that all actions a have constant cost $c_a > 0$, there is a finite number of sequences of actions (plans) P such that $\sum_{a\in P} c_a \leqslant U_G$. Any state s generated by plan P such that $\sum_{a\in P} c_a > 2 \times U_G$ will be discarded and will not be put in the *OPEN* list because $f(s) < 0 \leqslant B_B$. Given that there is a finite number of states that can be generated and put in the *OPEN* list, the algorithm will exhaust the *OPEN* list given enough time. Thus, it will terminate.

Algorithm 2 terminates when either the *OPEN* list is empty or a node s with $h(s) = 0$ is picked from the *OPEN* list for expansion. We will first prove that if the algorithm terminates when $OPEN = \emptyset$, then the plan returned is the optimal solution. If $f(s)$ overestimates the real maximum achievable benefit, then the discarded nodes s due to the cutoff comparison $f(s) \leqslant B_B$ cannot lead to nodes with higher benefit value than the current best found solution represented by B_B. Therefore, our algorithm does not discard any node that can lead to an optimal solution. For any node s that is picked from the *OPEN* list for expansion, we also have $g(s) \leqslant B_B$ because B_B always represents the highest g value of all nodes that have ever been generated. Combining the fact that no expanded node represents a better solution than the latest B_B with the fact that no node that was discarded from expansion (i.e., not put in or filtered out from the *OPEN* list) may lead to a better solution than B_B, we can conclude that if the algorithm terminates with an empty *OPEN* list then the final B_B value represents the optimal solution.

If Algorithm 2 does not terminate when $OPEN = \emptyset$, then it terminates when a node s with $h(s) = 0$ was picked from the *OPEN* list. We prove that s represents the optimal solution and the plan leading to s was the last one output by the algorithm. When s with $h(s) = 0$ is picked from the *OPEN* list, given that $\forall s' \in OPEN$: $f(s) = g(s) \geqslant f(s')$, all nodes in the *OPEN* list cannot lead to a solution with higher benefit value than $g(s)$. Moreover, let s_B represent the state for which the plan leading to s_B was last output by the algorithm; thus $B_B = g(s_B)$. If s_B was generated before s, then because $f(s) = g(s) < g(s_B)$, s should have been discarded and was not added to the *OPEN* list, which is a contradiction. If s_B was generated after s, then because $g(s_B) \geqslant g(s) = f(s)$, s should have been discarded from the *OPEN* list when s_B was added to the *OPEN* list and thus s should not have been picked for expansion. Given that s was not discarded, we have $s = s_B$ and thus P_s represents the last solution output by the algorithm. As shown above, none of the discarded nodes or nodes still in the *OPEN* list when s is picked can lead to better solution than s, where s represents the optimal solution. □

Discussion. Proposition 1 assumes that the heuristic estimate h is bounded and this can always be done. For any given state s, Eq. (4) indicates that $h^*(s) = B(P_s^B|s) = (U(s') - U(s)) - \sum_{a\in P_s^B} c_a \leqslant U(s') = \sum_{g\in s'} u_g \leqslant \sum_{g\in G} u_g = U_G$. Therefore, we can safely assume that any heuristic estimate can be bounded so that $\forall s$: $h(s) \leqslant U_G$.

To simplify the discussion of the search algorithm described above, we made several assumptions at the beginning of this section: all goals are soft, the heuristic used is admissible, the planner is forward state space, and there are no constraints beyond classical planning (e.g., no metric or temporal constraints). If any of those assumptions is violated, then some adjustments to the main search algorithm are necessary or beneficial. First, if some goals are "hard goals", then only nodes satisfying all hard goals can be termination nodes. Therefore, the condition for outputting the new best found plan needs to be changed from $g(s') > B_B$ to $(g(s') > B_B) \wedge (G_h \in s)$ where G_h is the set of all hard goals.

Second, if the heuristic is inadmissible, then the final solution is not guaranteed to be optimal. To preserve optimality, we can place all generated nodes in the OPEN list. The next section discusses this idea in more detail. Suffice it to say that if the $h(s)$ value of a given node s is guaranteed to be within ϵ of the optimal solution, then we can use $(1 + \epsilon) \times B_B$ as a cutoff bound and still preserve the optimal solution.

Lastly, if there are constraints beyond classical planning such as metric resources or temporal constraints, then we only need to expand the state representation, successor generator, and goal check condition to include those additional constraints.

3.2. Relaxed plan heuristics for PSP Net Benefit

Given a search node corresponding to a state s, the heuristic needs to estimate the net benefit-to-go of state s. In Section 2.2, we discussed a class of heuristics for classical planning that estimate the "cost-to-go" of an intermediate state

[9] Given that there are both positive and negative edge benefits in the state transition graph, it is desirable to show that there is no positive cycle (any plan involving positive cycles will have infinite achievable benefit value). Positive cycles do not exist in our state transition graph because traversing over any cycle does not achieve any additional utility but always incurs positive cost. This is because the utility of a search node s is calculated based on the world state encoded in s (not what accumulated along the plan trajectory leading to s), which does not change when going through a cycle c. However, the total cost of visiting s is calculated based on the sum of action costs of the plan trajectory leading to s, which increases when traversing c. Therefore, all cycles have non-positive net benefit (utility/cost trade-off).

in the search based on the idea of extracting relaxed plans. In this section, we will discuss our approach for adapting this class of heuristics to the PSP NET BENEFIT problem. Before getting into the details, it is instructive to understand the broad challenges involved in such an adaptation.

Our broad approach will involve coming up with a (relaxed) plan P^+ that solves the relaxed version our PSP problem for the same goals, but starting at state s (recall that the relaxed version of the problem is one where negative interactions between actions are ignored). We start by noting that in order to ensure that the anytime search finds the optimal solution, the estimate of the net benefit-to-go must be an upperbound on the true net benefit-to-go of s. To ensure this, we will have to find the "optimal" plan P^+ for solving the relaxed version of the original problem from state s. This is going to be NP-hard, since even for classical planning, finding the optimal relaxed plan is known to be NP-hard. So, we will sacrifice admissibility of the heuristic and focus instead on finding "good" relaxed plans in polynomial time.

For classical planning problems, the greedy backward sweep approach described in Section 2.2 has become the standard way to extract reasonable relaxed plans. This approach does not, however, directly generalize to PSP NET BENEFIT problems since we do not know which of the goals should be supported by the relaxed plan. There are two general greedy approaches to handle this problem:

Agglomerative Relaxed Plan Construction: In the agglomerative approach, we compute the greedy relaxed plan by starting with a null plan, and extending it incrementally to support one additional goal at a time, until the net benefit of the relaxed plan starts to reduce. The critical issue is the order in which the goals are considered. It is reasonable to select the first goal by sorting goals in the order of individual net benefit. To do this, for each goal $g \in G$, we compute the "greedy" relaxed plan P^+_g to support g (using the backward sweep approach in Section 2.2). The estimated net benefit of this goal g is then given as $u_g - C(P^+_g)$. In selecting the subsequent goals, however, we should take into account the fact that the actions already in the current relaxed plan will not contribute any more cost. By doing this, we can capture the positive interactions between the goals (note that while there are no negative interactions in the relaxed problem, there still exist positive interactions). We investigated this approach, and its variants in [38,44].

Pruned Relaxed Plan Construction: In the pruned approach, we first develop a relaxed plan to support *all* the top level goals (using the backward sweep algorithm). We then focus on pruning the relaxed plan by getting rid of non-beneficial goals (and the actions that are solely used to support those goals). We note at the outset that the problem of finding a subplan of a (relaxed) plan P^+ that has the best net benefit is a generalization (to PSP NET BENEFIT) of the problem of minimizing a (classical) plan by removing all redundant actions from it, and is thus NP-hard in the worst case ([19]). However, our intention is not to focus on the optimal pruning of P^+, but rather to improve its quality with a greedy approach.[10]

Although we have investigated both agglomerative and pruned relaxed plan construction approaches for PSP NET BENEFIT, in this paper we focus on the details of the pruned relaxed plan approach. The main reason is that we have found in our subsequent work ([5,12]) that pruned relaxed plan approaches lend themselves more easily to handling more general PSP NET BENEFIT problems where there are dependencies between goal utilities (i.e., the utility of achieving a set of goals is not equal to the sum of the utilities of individual goals).

Details of Pruned Relaxed Plan Generation in $Sapa^{\mathcal{PS}}$. Let us first formalize the problem of finding an optimal subplan of a plan P (whether it is relaxed or not).

Definition *(Proper Subplan).* Given a plan P seen as a sequence of actions (strictly speaking "steps" of type $s_i : a_j$ since a ground action can appear multiple times in the plan) $a_1 \cdots a_n$, P' is considered a proper subplan of P if: (1) P' is a subsequence of P (arrived at by removing some of the elements of P), and (2) P' is a valid plan (can be executed).

Most-beneficial Proper Subplan. A proper subplan P_o of P is the most beneficial subplan of a plan P if there is no other proper subplan P' of P that has a higher benefit than P_o.

Proposition 2. *Finding the most-beneficial Proper Subplan of a plan of a plan is NP-hard.*

Proof. The problem of minimizing a "classical" plan (i.e., removing actions while still keeping it correct) can be reduced to the problem of finding the subplan with the best net benefit. The former is NP-hard [19]. □

While we do not intend to find the most-beneficial proper subplan, it is nevertheless instructive to understand how it can be done. Below, we provide a criterion for identifying and removing actions from a plan P without reducing its net benefit.

[10] This makes sense as optimal pruning does not give any optimality guarantees on the underlying search since we are starting with a greedy relaxed plan supporting all goals.

Sole-supporter Action Set. Let G_P be the goals achieved by P, $A_{G'} \subseteq P$ is a sole-supporter action set of $G' \subseteq G_P$ if:

(1) The remaining plan $P_r = P \setminus A_{G'}$ is a valid plan.
(2) P_r achieves all goals in $G_P \setminus G'$.
(3) P_r does not achieve any goal in G'.
(4) There is no subset of action $A' \subseteq P_r$ that can be removed from P_r such that $P_r \setminus A'$ is a valid plan achieving all goals in $G_P \setminus G'$.

Note that there can be extra actions in P that do not contribute to the achievement of any goal and those actions can be considered as the sole-supporter for the empty goal set $G' = \emptyset$. The last condition above ensures that those actions are included in any supporter action set $A_{G'}$ and thus the remaining plan P_r is as beneficial as possible. We define the unbeneficial goal set and supporting actions as follows:

Unbeneficial Goal Set. For a given plan P that achieves the goal set G_P, the goal set $G' \subseteq G_P$ is an *unbeneficial* goal set if it has a sole-supporter action set $A_{G'}$ and the total action cost in $A_{G'}$ is higher than the total goal utility in G'.

It is quite obvious that removing the unbeneficial goal set G' and its sole-supporter action set $A_{G'}$ can only improve the total benefit of the plan. Thus, if $P_r = P \setminus A_{G'}$ then:

$$\sum_{g \in (G_p \setminus G')} u_g - \sum_{a \in P_r} c_a \geqslant \sum_{g \in G_P} u_g - \sum_{a \in P} c_a \tag{5}$$

Given the plan P achieving goals G_P, the best way to prune it is to remove the most unbeneficial goal set G' that can lead to the remaining plan with the highest benefit. Thus, we want to find and remove the unbeneficial goal set $G' \subseteq G_P$ and its sole-supporter action set $A_{G'}$ such that:

$$G' = \operatorname*{argmin}_{G_x \in G_P} \sum_{g \in G_x} u_g - \sum_{a \in A_{G_x}} c_a \tag{6}$$

Proposition 3. *Given a plan P, if G' with its sole-supporter action set $A_{G'}$ is the most unbeneficial goal set as specified by Eq. (6), then the plan $P' = P \setminus A_{G'}$ is the most beneficial proper subplan of P.*

Proof. Let P_o subplan of P with the most benefit, and B, B_o, B' be the net benefit of P, P_o, and P' respectively. Given that P_o is optimal, we have: $B_o \geqslant B'$. Let G_o be the set of goals achieved by P_o, we define $A_o = P \setminus P_o$ and $G'_o = G \setminus G_o$. We want to show that A_o is the sole-supporter set of G'_o. Given that P_o is a valid plan that achieves all goals in G_o and none of the goals in G'_o, the first three conditions for A_o to be the sole-supporter set of G'_o are clearly satisfied. Given that P_o is the most beneficial subplan of P, there should not be any subplan of P_o that can achieve all goals in G_o. Therefore, the fourth and last condition is also satisfied. Thus, A_o is the sole-supporter set of G'_o. Given that G' is the most unbeneficial goal set, Eq. (6) indicates that $B_1 = \sum_{g \in G'} u_g - \sum_{a \in A_{G'}} c_a \geqslant B_2 = \sum_{g \in G'_o} u_g - \sum_{a \in A_o} c_a$. Therefore, $B' = B + B_1 \geqslant B_o = B + B_2$. Combining this with $B_o \geqslant B'$ that we get above, we have $B' = B_o$ and thus P' is the most beneficial subplan of P. □

We implemented in our planner an incremental algorithm to approximate the identification and removal of the most unbeneficial goal set and its associated sole-supporter action set from the relaxed plan. Because we apply the procedure to the relaxed plan instead of the real plan, the only condition that needs to be changed is that the remaining plan $P' = P_G \setminus A_{G'}$ is still a valid relaxed plan (i.e., can execute when ignoring the delete list).

This scan and filter process contains two main steps:

(1) *Identification*: Using the causal-structure of the relaxed plan, identify the set of top level goals that each action supports. This step is illustrated by Algorithm 3.
(2) *Incremental removal*: Using the supported goal sets calculated in step 1, heuristically remove actions and goals that are not beneficial. This step is illustrated in Algorithm 4.

The first step uses the *supported goals* list *GS* that is defined for each action a and fact p as follows:

$$GS(a) = \bigcup_{p \in Effect(a)} GS(p), \tag{7}$$

$$GS(p) = \begin{cases} p \cup \left(\bigcup_{p \in Prec(a)} GS(a)\right) & \text{if } p \in G, \\ \bigcup_{p \in Prec(a)} GS(a) & \text{if } p \notin G \end{cases} \tag{8}$$

In our implementation, beginning with $\forall a \in P^+, \forall p \notin G$: $GS(a) = GS(p) = \emptyset$ and $\forall g \in G : GS(g) = \{g\}$, the two rules listed above are applied repeatedly starting from the top-level goals until no change can be made to either the $GS(a)$ or $GS(p)$ set.

Input: A relaxed plan P^+ achieving the goal set G_P;
Output: The supported goal set for each action $a \in P^+$ and fact;
begin
  $\forall a \in P:\ GS(a) \leftarrow \emptyset$;
  $\forall g \in G_P:\ GS(g) \leftarrow \{g\}$;
  $\forall p \notin G_P:\ GS(p) \leftarrow \emptyset$;
  DONE $\leftarrow$ *false*;
  while *not* DONE **do**
    DONE $\leftarrow$ *true*;
    forall $a \in P^+$ **do**
      if $GS(a) \neq \bigcup_{p \in Effect(a)} GS(p)$ **then**
        $GS(a) \leftarrow \bigcup_{p \in Effect(a)} GS(p)$;
        DONE $\leftarrow$ *false*;
      end
    end
    forall *proposition* p **do**
      if $GS(p) \neq GS(p) \cup (\bigcup_{p \in Precond(a)} GS(a))$ **then**
        $GS(p) \leftarrow GS(p) \cup (\bigcup_{p \in Precond(a)} GS(a))$;
        DONE $\leftarrow$ *false*;
      end
    end
  end
end

Algorithm 3. Algorithm to find the supported goals list GS for all actions and facts.

Input: A relaxed plan P^+ and parameter $N > 0$;
Output: A relaxed plan $P' \subseteq P^+$ with higher or equal benefit compared to P^+;
begin
  $DONE \leftarrow false$;
  while *not* DONE **do**
    Use Algorithm 3 to compute all $GS(a)$, $GS(p)$ using P^+;
    forall $G' \subseteq G_{P^+}$ *and* $|G'| \leq N$ **do**
      $SA(G') \leftarrow \bigcup_{GS(a) \subseteq G'} a$;
    end
    $G_x \leftarrow \operatorname{argmin}_{G' \subseteq G_{P^+},\ |G'| \leq N} (\sum_{g \in G'} u(g) - \sum_{a \in SA(G')} c(a))$;
    if $(\sum_{g \in G_x} u(g) - \sum_{a \in SA(G_x)} c(a)) \leq 0$ **then**
      $P^+ \leftarrow (P^+ \setminus SA(G_x))$;
    else
      $DONE \leftarrow true$;
    end
  end
end

Algorithm 4. Pruning the relaxed plan using the supported goal set found in Algorithm 3.

Thus, for each action a, we know which goal it supports. Fig. 3 shows the extracted relaxed plan $P^+ = \{a_1 : Move(A, B), a_2 : Move(A, C), a_3 : Move(C, D), a_4 : Move(A, E)\}$ along with the goals each action supports. By going backward from the top level goals, actions a_1, a_3, and a_4 support only goals g_1, g_3, and g_4 so the goal support list for those actions will be $GS(a_1) = \{g_1\}$, $GS(a_3) = \{g_3\}$, and $GS(a_4) = \{g_4\}$. $at(C)$, the precondition of the action a_3 would in turn contribute to goals g_3, thus we have: $GS(at(C)) = \{g_3\}$. Finally, because a_2 supports both g_2 and $at(C)$, we have: $GS(a_2) = GS(g_2) \cup GS(at(C)) = \{g_3, g_2\}$.

In the second step, using the supported goal set of each action, we can identify the subset $SA(G')$ of actions that contributes *only* to the goals in a given subset of goals $G' \subseteq G$:

$$SA(G') = \bigcup_{GS(a) \subseteq G'} a \tag{9}$$

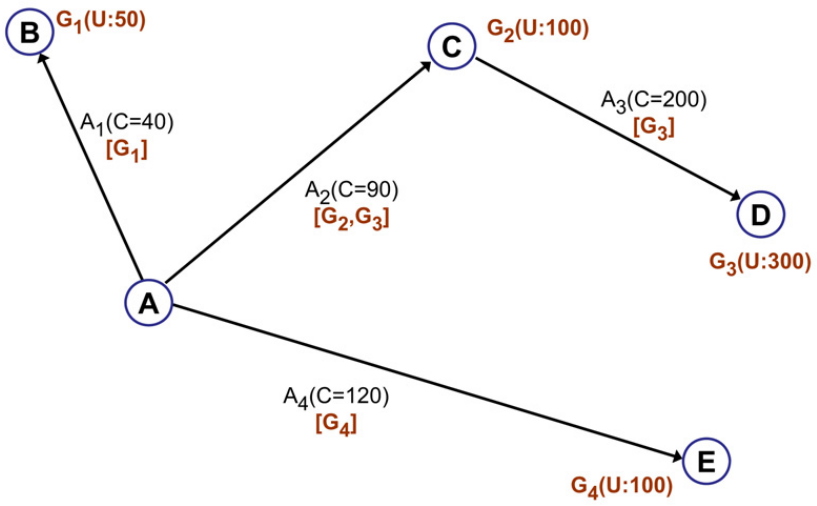


Fig. 3. The relaxed plan.

If the total cost of actions in $SA(G')$ exceeds the sum of utilities of goals in G' (i.e., $\sum_{a\in SA(G')} c(a) \geqslant \sum_{g\in G'} u(g)$), then we can remove G' and $SA(G')$ from the relaxed plan. We call those subgoal sets *unbeneficial*. In our example, action a_4 is the only one that contributes to the achievement of g_4. Since $c(a_4) \geqslant u(g_4)$ and a_4 does not contribute to any other goal, we can remove a_4 and g_4 from consideration. The remaining three actions a_1, a_2, and a_3 in P^+ and the goals g_1, g_2, and g_3 all appear beneficial. In general, it will be costly if we consider all $2^{|G|}$ possible subsets of goals for potential removal. Therefore, in Algorithm 4, we only consider the subsets G' of goals such that $|G'| \leqslant N$, with N as the pre-defined value. If there is no unbeneficial goal set G' with $|G'| \leqslant N$, then we terminate the pruning algorithm. If there are beneficial goal sets, then we select the most unbeneficial one G_x, and remove the actions that solely support G_x from the current relaxed plan P^+, then start the pruning process again. In our current implementation, we only consider $N = 0$, $N = 1$, and $N = 2$. However, as we discuss later, considering only $N = 0$ and $N = 1$ is often sufficient.

After removing unbeneficial goals and the actions (solely) supporting them, the cost of the remaining relaxed plan and the utility of the goals that it achieves are used to compute the h value. Thus, in our ongoing example, $h(I) = (u(g_3) + u(g_2) + u(g_1)) - (c(a_2) + c(a_3) + c(a_1)) = (100 + 300 + 50) - (90 + 200 + 40) = 120$. While this heuristic is effective, it is not guaranteed to be admissible because the original relaxed plan was heuristically extracted and may not be optimal. Moreover, we only consider removing subset G' of goals such that $|G'| \leqslant 2$. Therefore, if we use this heuristic in the algorithm discussed in the previous section (Algorithm 2), then there is no guarantee that the plan returned is optimal. This is because (i) a pruned node may actually lead to an optimal plan (i.e., expandable to reach node s with $g(s) > B_B$); and (ii) the last B_B may not represent an optimal plan. To handle the first issue, we made adjustments in the implementation so that even though weight $w = 1$ is used in equation $f = g + w \cdot h$ to sort nodes in the queue, another value $w = 2$ is used for pruning (unpromising) nodes with $f = g + w \cdot h \leqslant B_B$. For the second issue, we continue the search for a better plan after a node with $h(s) = 0$ is found until some resource limits are reached (e.g., we have reached a certain number of search nodes or a given planning time limit).

Using relaxed plans for lookahead search. In addition to using the final pruned relaxed plan for heuristic-computation purposes, we have also implemented a rudimentary lookahead technique that takes the relaxed plans given to us and simulates their execution in the actual planning problem as much as possible (i.e., the planner attempts each action, in the order defined by the causal structure of the relaxed plan, and repeats this process until no actions can be executed; the resulting state is then evaluated and placed into the *OPEN* list). This technique is inspired by the results of the planner YAHSP [45], which used a similar but more sophisticated lookahead strategy. We found that this method helps to find better quality plans in less time, but also causes the search to reach a termination search node more quickly.

4. Handling PDDL3.0 simple preferences

The organizers of the 5th International Planning Competition (IPC-5) introduced PDDL3.0 [23], which can express variations of partial satisfaction planning problems. One track named "simple preferences" (PDDL3-SP) has qualities analogous to PSP Net Benefit.

In PDDL3-SP, each preference p_i includes a variable v_{p_i} that counts the number of times p_i is violated and c_i representing the violation cost when p_i is not satisfied. Preferences can be associated with more than one action *precondition* or *goal*. Additionally, they can include conjunctive and disjunctive formulas on fluents. The objective function is:

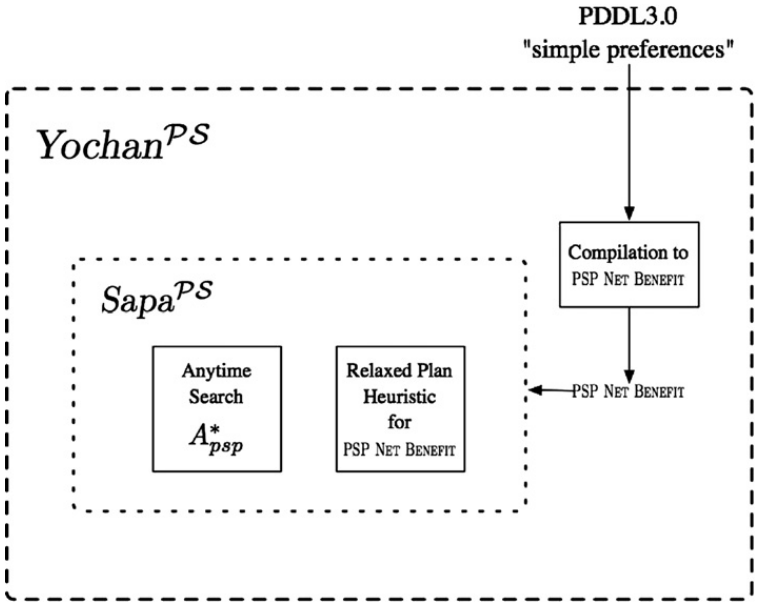


Fig. 4. Relationship between techniques defined in this paper.

$$\text{minimize } c_1 \cdot v_{p_1} + c_2 \cdot v_{p_2} + \cdots + c_n \cdot v_{p_n} \tag{10}$$

where violation costs $c_i \in \mathbb{R}$ are multiplied by the number of times p_i is violated.

In this section we show a method of compiling PDDL3-SP problems into PSP Net Benefit problems. This involves converting the goal violation cost (i.e., failing to satisfy goal preferences) into goal utility (i.e., successfully achieving goals) and the action precondition violation cost into action costs. The compilation takes advantage of the similarities between the two problems, and the main challenge is in handling the differences between PDDL3-SP and PSP Net Benefit. The compilation is solved by *Sapa*$^{\mathcal{PS}}$, as shown in Fig. 4, in an adaptation of that planner called *Yochan*$^{\mathcal{PS}}$.

Other compilation methods for handling the constraints in PDDL3.0 were also introduced in the IPC-5. For instance, the planner MIPS-XXL [18] used a transformation from PDDL3.0 that involved a compilation into hard goals and numeric fluents. *Yochan*$^{\mathcal{PS}}$ and other compilation approaches proved competitive in the competition. In fact, both *Yochan*$^{\mathcal{PS}}$ and MIPS-XXL participated in the "simple preferences" track and received a "distinguished performance" award. However, the compilation used by MIPS-XXL did not allow the planner to directly handle the soft goal preferences present in PDDL3.0. To assist in determining whether considering soft goals directly during the planning process is helpful, in this section we also introduce a separate compilation from PDDL3.0 that completely eliminates soft goals, resulting in a classical planning problem with action costs. The problem is then solved by the anytime A^* search variation implemented in *Sapa*$^{\mathcal{PS}}$. We call the resulting planner *Yochan*^{COST}.

4.1. *Yochan*^{COST}: PDDL3-SP to hard goals

Recently, approaches to compiling planning problems with *soft* goals to those with *hard* goals have been proposed [18]. In fact, Keyder & Geffner [33] directly handle PSP Net Benefit by compiling the problem into one with hard goals. While we explicitly address soft goals in *Yochan*$^{\mathcal{PS}}$, to evaluate the advantage of this approach we explore the possibility of planning for PDDL3-SP by compiling to problems with only hard goals. We call the planner that uses this compilation strategy *Yochan*^{COST}. It uses the anytime A^* search variation from *Sapa*$^{\mathcal{PS}}$ but reverts back to the original relaxed plan heuristic of *Sapa* [13].[11]

Algorithm 5 shows the algorithm for compiling PDDL3-SP goal preferences into a planning problem with hard goals and actions with cost. Precondition preferences are compiled using the same approach as in *Yochan*$^{\mathcal{PS}}$, which is discussed later. The algorithm works by transforming a "simple preference" goal into an equivalent hard goal with dummy actions that give that goal. Specifically, we compile a goal preference $pref(G') \mid G' \subseteq G$ to two actions: action a_1 takes G' as a condition and action a_2 takes $\neg G'$ as a condition (foregoing goal achievement). Action a_1 has cost *zero* and action a_2 has cost equal to the violation cost of not achieving G'. Both a_1 and a_2 have a single dummy effect to achieve a newly created hard goal that indicates we "have handled the preference" $pref(G')$. At least one of these actions, a_1 or a_2, is always included in the final plan, and every other non-preference action deletes the new goal (thereby forcing the planner to again decide whether to re-achieve the hard goal, and again include the necessary achievement actions). After the compilation to hard goals, we will have actions with disjunctive preconditions. We convert these into STRIPS with cost by calling Algorithm 7.

[11] This is done so we may compare the compilation in our anytime framework.

Input: a PDDL3-SP problem;
Output: a PDDL problem with new set of hard goals and an extended action set;
$B := \emptyset$;
forall $pref(G') \mid G' \subseteq G$ **do**
  create two new actions a_1 and a_2;
  $pre(a_1) := G'$;
  $g_{G'} := name(pref(G'))$;
  $eff(a_1) := g_{G'}$;
  $C(a_1) := 0$;
  $B := B \cup \{a_1\}$;
  $G := (G \cup \{g_{G'}\}) \setminus \{G'\}$;

  $pre(a_2) := \neg G'$;
  $eff(a_2) := g_{G'}$;
  $C(a_2) := c(pref(G'))$;
  $B := B \cup \{a_2\}$;
  $G := (G \cup \{g_{pref}\}) \setminus \{G'\}$;
end
$A := B \cup A$;

Algorithm 5. PDDL3-SP goal preferences to hard goals.

```
(:goal  (preference P0A
          (stored goods1 level1)))
```

(a) Goal preferences in PDDL3-SP

```
(:action p0a-0
    :parameters ()
    :cost 0.0
    :precondition (and (stored goods1 level1))
    :effect (and (hasPref-p0a)))

(:action p0a-1
    :parameters ()
    :cost 500.0
    :precondition (and
        (not (stored goods1 level1)))
    :effect (and (hasPref-p0a)))

With new goal:  (hasPref-p0a)
```

(b) Actions with cost

Fig. 5. Compiling goal preferences from PDDL3-SP to cost-based planning.

After the compilation, we can solve the problem using any planner capable of handling hard goals and action costs. In our case, we use *Sapa*$^{\mathcal{PS}}$ with the heuristic used in the non-PSP planner *Sapa* to generate *Yochan*^{COST}. We are now *minimizing* cost instead of *maximizing* net benefit (and hence take the negative of the heuristic for search). In this way, we are performing an anytime search algorithm to compare with *Yochan*$^{\mathcal{PS}}$. As in *Yochan*$^{\mathcal{PS}}$, which we will explain in the next section, we assign unit cost to all non-preference actions and increase preference cost by a factor of 100. This serves two related purposes. First, the heuristic computation uses cost propagation such that actions with zero cost will essentially look "free" in terms of computational effort. Secondly, and similarly, actions that move the search toward goals take some amount of computational effort which is left uncounted when action costs are zero. In other words, the search node evaluation completely neglects tree depth when actions have zero cost.

Example. Consider an example taken from the IPC-5 TPP domain shown in Fig. 5 and Fig. 6. On the left side of these two figures we show examples of PDDL3-SP action and goal preferences. On the right side, we show the newly created actions and goals resulting from the compilation to classical planning (with action costs) using our approach described above.

In this example, the preferred goal `(stored goods1 level1)` has a violation cost of 5 (defined in Fig. 6). We add a new goal `(hasPref-p0a)` and assign the cost of achieving it with action `p0a-1` (i.e., not having the goal) to 500.

```
(:action drive
    :parameters
        (?t - truck ?from ?to - place)
    :precondition
        (and
        (at ?t ?from)
            (connected ?from ?to)
            (preference p-drive
                (and
                (ready-to-load
                    goods1 ?from level0)
                (ready-to-load
                    goods2 ?from level0)
                (ready-to-load
                    goods3 ?from level0))))
    :effect (and (not (at ?t ?from))
            (at ?t ?to)))

Weight assigned to preferences:
(:metric
    (+ (× 10 (is-violated p-drive) )
      (× 5 (is-violated P0A) )))
```

(a) Action preferences in PDDL3-SP

```
(:action drive-0
    :parameters
            (?t - truck ?from ?to - place)
    :cost 100
    :precondition (and
            (at ?t ?from) (connected
                ?from ?to)
            (ready-to-load
                goods1 ?from level0)
            (ready-to-load
                goods2 ?from level0)
            (ready-to-load
                goods3 ?from level0)))
    :effect (and (not (at ?t ?from))
            (at ?t ?to)))

(:action drive-1
    :parameters
            (?t - truck ?from ?to - place)
    :cost 1000
    :precondition (and
            (at ?t ?from) (connected
                ?from ?to))
    :effect (and (not (at ?t ?from))
            (at ?t ?to)))
```

(b) Actions with cost

Fig. 6. Compiling action preferences from PDDL3-SP to cost-based planning.

4.2. Yochan$^{\mathcal{PS}}$: PDDL3-SP to PSP

When all soft goals in PDDL3-SP are compiled to hard goals, it is always easiest (in terms of search depth) to do nothing. That is, simply executing the higher cost preference avoidance actions will achieve the goal of having "handled" the preference. Consequentially, the relaxed plan based heuristic may be misleading because it is uninformed of the mutual exclusion between the preference evaluation actions. That is, the heuristic may see what appears to be a "quick" path to a goal, where in fact that path requires the undesirable consequence of violating a preference. Instead, viewing preferences as goals that are desirable to achieve (i.e., attaching reward to achieving them) allows the relaxed plan heuristic to be directed to them. As such, we introduce a method of converting PDDL3-SP problems into PSP problems, which gives the preferences a reward for achievement rather than a cost for violation, thus giving better direction for the relaxed planning graph heuristic. There are two main differences between how PDDL3-SP and PSP Net Benefit define *soft* goals. First, in PDDL3-SP, soft goal preferences are associated with a preference name which allows them to be given a violation cost. Second, goal preferences can consist of a disjunctive or conjunctive goal formula. This is opposed to PSP Net Benefit problems where individual goals are given utility. Despite these differences, the similarities are abundant:

- The *violation cost* for failing to achieve an individual goal in PDDL3-SP and *achievement utility* in PSP Net Benefit are semantically equivalent. Thus, if there is a goal g with a violation cost of $c(g)$ for *not* achieving it in PDDL3-SP, then it is equivalent to having this goal with utility of $u_g = c(g)$ for achieving it in PSP.
- PDDL3-SP and PSP Net Benefit both have a notion of plan quality based on a quantitative metric. PDDL3-SP bases a plan's quality on how well it reduces the goal preference violation cost. On the other hand, PSP Net Benefit views cost as a monotonically increasing value that measures the resources consumed by actions. In PDDL3-SP we have a plan metric ρ and a plan P_1 has a higher quality than a plan P_2 if and only if $\rho(P_1) < \rho(P_2)$. A plan's quality in PSP Net Benefit deals with the trade-off between the utility of the goals achieved and the cost of the actions to reach the goals. Therefore, a plan P_1 has a higher quality than a plan P_2 in PSP Net Benefit if and only if $U(P_1) - C(P_1) > U(P_2) - C(P_2)$, where $U(P)$ represents the utility of a plan P and $C(P)$ represents the cost of a plan P.
- Preferences on action conditions in PDDL3-SP can be viewed as a *conditional cost* in PSP Net Benefit. The cost models on actions differ only in that PDDL3-SP provides a *preference* which acts as a condition for applying action cost. Like violation costs for goal preferences, action condition violation cost is incurred if a given action is applied to a state where that condition is not satisfied.

```
Input: A PDDL3-SP problem;
Output: A PSP NET BENEFIT problem;
B := ∅;
forall pref(G') | G' ⊆ G do
    pre(a) := G';
    g_G' := name(pref(G'));
    eff(a) := g_G';
    B := B ∪ {a};
    U(g_G') := c(pref(G'));
    G := (G ∪ {g_G'}) \ {G'};
    forall b ∈ A do
        eff(b) := eff(b) ∪ ¬{g_G'};
    end
end
A := B ∪ A;
```

Algorithm 6. Compiling goal preferences to PSP NET BENEFIT goals.

```
i := 0;
forall a ∈ A do
    foreach precSet ∈ P(pref(a)) do
        pre(a_i) := pre(a) ∪ precSet;
        eff(a_i) := eff(a);
        c_{a_i} := 100 × c(pref(a) \ precSet);
        A := A ∪ {a_i};
        i := i + 1;
    end
    A := A \ {a};
end
```

Algorithm 7. Compiling preference preconditions to actions with cost. Note that, though not shown here, disjunctive preconditions are compiled away using the method described in [21].

As part of our compilation, we first transform "simple preference" goals to equivalent goals with utility equal to the cost produced for not satisfying them in the PDDL3-SP problem. Specifically, we can compile a goal preference $pref(G') \mid G' \subseteq G$ to an action that takes G' as a condition. The effect of the action is a newly created goal representing the fact that we "have the preference" $pref(G')$.

Both PDDL3-SP and PSP NET BENEFIT have a notion of cost on actions, though their view differs on how to define cost. PSP NET BENEFIT defines cost directly on each action, while PDDL3-SP uses a less direct approach by defining the penalty for not meeting an execution condition. Therefore, PDDL3-SP can be viewed as considering action cost as a conditional effect on an action where cost is incurred on the preference condition's negation. From this observation, we can compile PDDL3.0 "simple preferences" on actions in a manner that is similar to how conditional effects are compiled [21].

Goal Compilation. The goal compilation process converts goal preferences into additional soft goals and actions achieving them in PSP. Algorithm 6 illustrates the compilation of goals. We begin by creating a new action a for every preference $pref(G') \mid G' \subseteq G$ in the goals. The action a has G' as a set of preconditions, and a new effect, $g_{G'}$. We then add $g_{G'}$ to the original goal set G, and give it utility equal to the cost $c(pref(G'))$ of violating the preference $pref(G')$. We remove the preference $pref(G')$ from the resulting problem and also force every non-compiled action that destroys G' to remove $g_{G'}$ (by adding $g_{G'}$ to the delete list of these actions).

Action Compilation. To convert precondition action preferences, for each action $a \in A$ we generate $P(pref(a))$ as the power set of $pref(a)$ (i.e.,$P(pref(a))$ containing all possible subsets of $pref(a)$). As Algorithm 7 shows, for each combination of preference $s \in P(pref(a))$, we create an action a_s derived from a. The cost of the new action a_s equals the cost of failing to satisfy all preferences in $pref(a) \setminus s$. We remove a from the domain after all of its compiled actions a_s are created. Since some preferences contain disjunctive clauses, we compile them away using the method introduced in [21] for converting disjunctive preconditions in ADL to STRIPS. Notice that because we use the power set of preferences, this could potentially result in a large number of newly formed actions. Since this increase is related to number of preferences, the number of actions that need to be considered during search may seem unwieldy. However, we found that in practice this increase is usually minimal. After completion of the planning process, we apply Eq. (11) to determine the PDDL3-SP total violation cost evaluation:

$$\text{TOTALCOST} = \sum_{g \in G} u_g - \sum_{g' \in G'} u_{g'} + \sum_{a \in P} c_a \tag{11}$$

```
(:goal  (preference P0A (stored goods1
    level1)))
```

(a) Goal preferences in PDDL3-SP

```
(:action p0a
    :parameters ()
    :cost 100
    :precondition (and
      (stored goods1 level1))
    :effect (and (hasPref-p0a)))

With new goal:  ((hasPref-p0a) 5.0)
```

(b) Action with cost in PSP

Fig. 7. Compiling goal preferences from PDDL3-SP to PSP.

Action Selection. The compilation algorithm will generate a set of actions A_a from an original action a with $|A_a| = 2^{|pref(a)|}$. Given that actions in A_a appear as separate operators to a planner, this can result in multiple action instances from A_a being included in the plan. Therefore, a planner could produce plans with superfluous actions. One way to fix this issue is to explicitly add negations of the preference conditions that are not included in the new action preconditions (i.e., we can use a negation of the precondition formula in the actions rather than removing the whole condition). This is similar to the approach taken by [21] when compiling away conditional effects. This compilation approach, however, may result in several disjunctive preconditions (from negating the original conjunctive preference formula), which will result in even more actions being included in the problem. To overcome this, we use a simple criterion on the plan that removes the need to include the negation of clauses in the disjunctive preferences. Given that all actions in A_a have the same effect, we enforce that for every action generated from a, only the *least cost* applicable action $a_i \in A_a$ can be included in P at a given forward search step. This criterion is already included in *Sapa*$^{\mathcal{PS}}$.

Example. Consider the examples found in Figs. 6 and 7. Fig. 6 shows the compilation for the TPP domain action: `drive` and Fig. 7 shows a TPP domain PDDL3-SP goal preference that has been compiled into PSP Net Benefit.

For the action compilation, Fig. 6 shows the preference `p-drive` has a cost of $10 \times 100 = 1000$ for failing to have all goods ready to load at level 0 of a particular location at the time `drive` is executed. We translate this idea into one where we either (1) have all goods ready to load at level 0 (as in the new action `drive-0` with cost 100) or (2) do not have all goods ready to load at level 1 (as in the new action `drive-1` with cost 1000).

To convert the goal condition from PDDL3-SP into PSP Net Benefit we generate a single action named for the preference, as shown in Fig. 7. The new action takes the preference goal as a precondition and we again introduce the new goal `(hasPref-p0a)`. However, with this compilation process we give it a utility value of 5.0. This is the same as the cost for being unable to achieve `(stored goods1 level1)`.

As for implementation details, *Yochan*$^{\mathcal{PS}}$ multiplies the original preference costs by 100 and uses that to direct the forward search. All actions that do not include a preference are given a default unit cost. Again, we do this so the heuristic can direct search toward short-length plans to reduce planning time. An alternative to this method of artificial scale-up would be to increase the preference cost based on some function derived from the original problem. In our initial experiments, we took the number of actions required in a relaxed plan to reach all the goals at the initial state and used this value to generate a scale-up factor. However, our preliminary observations using this approach yielded worse results in terms of plan quality.

After the compilation process is done, *Sapa*$^{\mathcal{PS}}$ is called to solve the new PSP Net Benefit problem with the normal objective of maximizing the net benefit. When a plan P is found, newly introduced actions resulting from the compilations of goal and action preferences are removed before returning P to the user.

5. Empirical evaluation

Most of the problems in the "simple preferences" track of IPC-5 consist of groups of preferred disjunctive goals. These goals involve various aspects of the problems (e.g., a deadline to deliver a package in the *trucks* domain). The *Yochan*$^{\mathcal{PS}}$ compilation converts each preference p into a series of actions that have the preference condition as a precondition and an effect that indicates that p is satisfied. The utility of a preferred goal is gained if we have obtained the preference at the end of the plan (where the utility is based on the penalty cost of not satisfying the preference in PDDL3-SP). In this way, the planner is more likely to try to achieve preferences that have a higher penalty violation value.

In the competition, *Yochan*$^{\mathcal{PS}}$ was able to solve problems in five of the domains in the "simple preferences" track. Unfortunately, many of the problems in several domains were large and *Yochan*$^{\mathcal{PS}}$ ran out of memory due to its action grounding process. This occurred in the *pathways*, *TPP*, *storage* and *trucks* domains. Also, some aspects of several domains (such as conditional effects and quantification) could not be handled by our planner directly and needed to be compiled to STRIPS. The competition organizers could not compile the *openstacks* domain to STRIPS, and so *Yochan*$^{\mathcal{PS}}$ did not participate

in solving it. Additionally, the *pipesworld* domain did not provide a "simple preferences" category. *Yochan*$^{\mathcal{PS}}$ also handles hard goals, which were present in some of the problems, by only outputting plans when such goals are satisfied. The *Sapa*$^{\mathcal{PS}}$ heuristic was also slightly modified such that hard goals could never be removed from a relaxed plan [3].

To test whether varying goal set sizes for the heuristic goal removal process affects our results, we compared running the planner with removing goal set sizes in each iteration of at most 1 and at most 2. It turns out that in almost all of the problems from the competition, there is no change in the quality of the plans found when looking at individual goals (as against individual goals and pairs of goals) during the goal removal process of the heuristic. Only in two problems in the *rovers* domain does there exist a minor difference in plan quality (one in favor of looking at only single goals, and one in favor of looking at set sizes of one and two). There is also an insignificant difference in the amount of time taken to find plans.

In conclusion, *Yochan*$^{\mathcal{PS}}$ entered the 5th International Planning Competition (IPC-5), where it performed competitively in several of the domains given by the organizers. Its performance was particularly good in "logistics" style domains. The quality of the plans found by *Yochan*$^{\mathcal{PS}}$ earned it a "distinguished performance" award in the "simple preferences" track. For comparison, we solved the IPC-5 problems with *Yochan*^{COST} and show that compiling directly to classical planning with action cost performs worse than compiling to a PSP NET BENEFIT problem in the competition domains.

For the rest of this section we evaluate the performance of *Yochan*$^{\mathcal{PS}}$ in each of the five "simple preferences" domains for which the planner participated. For all problems, we show the results from the competition (which can also be found on the competition website [24]). We focus our discussion on plan quality rather than solving time, as this was emphasized by the IPC-5 organizers. To compare *Yochan*$^{\mathcal{PS}}$ and *Yochan*^{COST}, we re-ran the results (with a small bug fix) using a 3.16 Ghz Intel Core 2 Duo with 4 GB of RAM, 1.5 GB of which was allocated to the planners using Java 1.5.

5.1. *The* trucks *domain*

The *trucks* domain consists of trucks that move packages to a variety of locations. It is a logistics-type domain with the constraint that certain storage areas of the trucks must be free before loading can take place into other storage areas. In the "simple preferences" version of this domain, packages must be delivered at or before a certain time to avoid incurring a preference violation penalty.

Fig. 8(a) shows the results for the *trucks* domain in the competition. Over all, *Yochan*$^{\mathcal{PS}}$ performed well in this domain compared to the other planners in the competition. It scaled somewhat better than both MIPS-XXL [18] and MIPS-BDD [18], though the competition winner, SGPlan [32] solved more problems, often with a better or equal quality. Notably, in problems 7 through 9, *Yochan*$^{\mathcal{PS}}$ had difficulty finding good quality plans. Examining the differences between the generated problems provides some insight into this behavior. In the first ten problems of this domain, the number of preferences (i.e., soft goals) increased as part of the increase in problem size. These all included existential quantification to handle deadlines for package delivery, where a package must be delivered before a particular encoded time step in the plan (time increases by one unit when driving or delivering packages). For example, *package1* may need to be delivered sometime before a time step t_3. Because this criterion was defined using a predicate, this caused the number of grounded, soft disjunctive goal sets to increase.[12] This in turn caused more goals to be considered at each time step. The planning graph's cost propagation and goal selection processes would take more time in these circumstances. In contrast, the second set of problems (problems 11 through 20) contained absolute package delivery times on goal preferences (e.g., *package1* must be delivered at exactly time t_5) thereby avoiding the need for disjunctive preferences. The planner solved four instances of these harder problems.[13]

A seeming advantage to *Yochan*^{COST} in this domain is that it is attempting to find the *least costly* way of achieving the goal set and does not rely on pruning away goals as *Yochan*$^{\mathcal{PS}}$ does. In *trucks*, the violation cost for failing to satisfy goal preferences turns out to be low for many of the goals, and so the *Sapa*$^{\mathcal{PS}}$ heuristic used by *Yochan*$^{\mathcal{PS}}$ may prune away some of the lower valued goals if the number of actions required for achievement is deemed too high. However, this advantage seems not to help the planner too much here. Also note that *Yochan*^{COST} has great difficulty with problems 8 and 9. Again, this is largely due to compilation of goals to actions, as the large number of actions that were generated caused the planner's branching factor to increase such that many states with equal heuristic values were generated. When large numbers of preferences exist *Yochan*^{COST} must "decide" to ignore them by adding the appropriate actions.

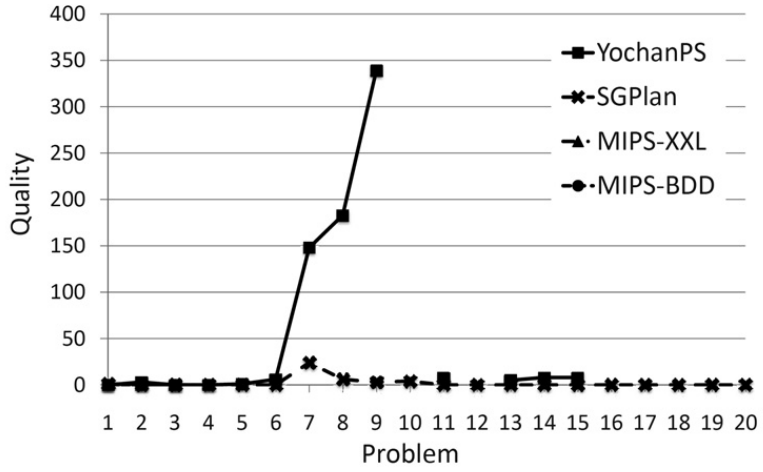
5.2. *The* pathways *domain*

This domain has its roots in molecular biology. It models chemical reactions via actions and includes other actions that choose initial substrates. Goals in the "simple preferences" track for this domain give a preference on the substances that must be produced by a pathway.

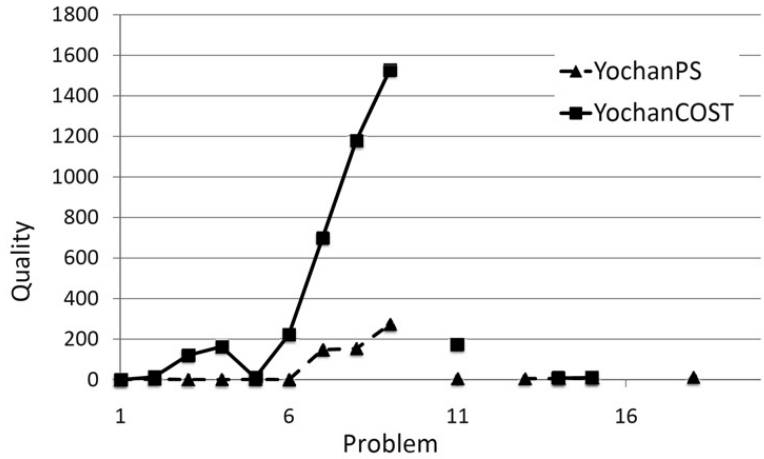
Fig. 9(a) shows that *Yochan*$^{\mathcal{PS}}$ tends to scale poorly in this domain, though this largely is due to the planner running out of memory during the grounding process. For instance, the number of objects declared in problem 5 caused our grounding

[12] Recall that the compilation to PSP NET BENEFIT generates a new action for each clause of a disjunctive goal formula.

[13] Note that *Yochan*$^{\mathcal{PS}}$ solved more problems than in the competition on the new runs, as the CPU was faster.



(a) IPC-5 results. $Yochan^{\mathcal{PS}}$ solved 13; MIPS-XXL solved 3; MIPS-BDD solved 4; SGPlan solved 20



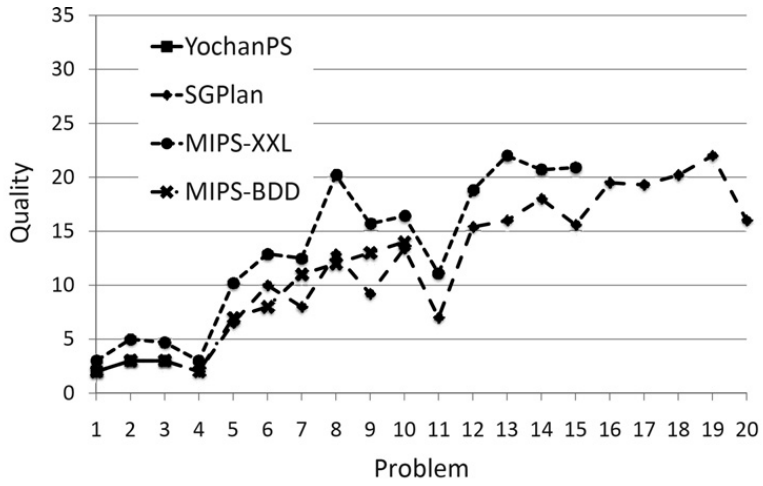
(b) $Yochan^{\mathcal{PS}}$ vs. $Yochan^{COST}$. $Yochan^{\mathcal{PS}}$ solved 14; $Yochan^{COST}$ solved 12

Fig. 8. IPC-5 *trucks* "simple preferences".

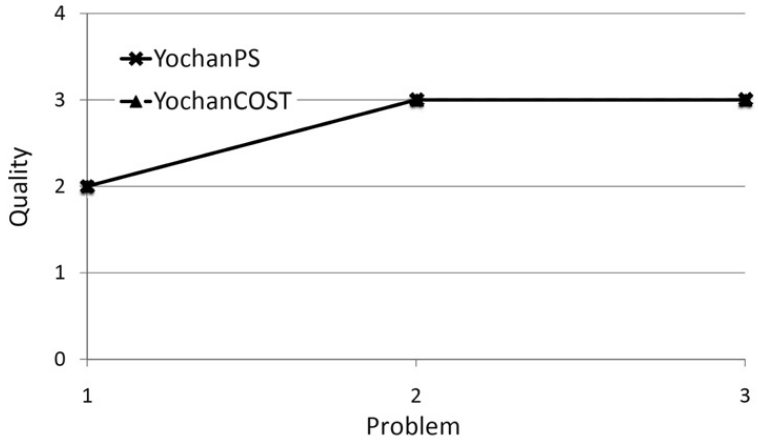
procedure to attempt to produce well over 10^6 actions. On most of its solved problems $Yochan^{\mathcal{PS}}$ provided equal quality in comparison to the other planners. Fig. 9(b) shows that both $Yochan^{\mathcal{PS}}$ and $Yochan^{COST}$ found plans of equal quality. Note that fixing a small search bug since the competition in $Yochan^{\mathcal{PS}}$ and $Yochan^{COST}$ caused the planners, in this domain, to fail to find a solution in problem 4 on the new runs (though $Yochan^{\mathcal{PS}}$ was able to find a solution during the competition and this is the only problem in which $Yochan^{\mathcal{PS}}$ performs worse).

5.3. *The* rovers *domain*

The *rovers* domain initially was introduced at the Third International Planning Competition (IPC-3). For the "simple preferences" version used in IPC-5, we must minimize the summed cost of actions in the plan while simultaneously minimizing violation costs. Each action has a cost associated with it through a numeric variable specified in the plan metric. The goals from IPC-3 of communicating rock samples, soil samples and image data are made into preferences, each with varying violation cost. Interestingly, this version of the domain mimics the PSP Net Benefit problem in the sense that the cost of moving from place to place causes a numeric variable to increase monotonically. Each problem specifies this variable as part of its problem metric, thereby allowing the variable to act as the cost of traversing between locations. Note that the problems in this domain are not precisely the PSP Net Benefit problem but are semantically equivalent. Additionally, none of the preferences in the competition problems for this domain contains disjunctive clauses, so the number of additional actions generated by the compilation to PSP Net Benefit is small.



(a) IPC-5 results. *Yochan*$^{\mathcal{PS}}$ solved 4; MIPS-XXL solved 15; MIPS-BDD solved 10; SGPlan solved 30



(b) *Yochan*$^{\mathcal{PS}}$ vs. *Yochan*^{COST}. *Yochan*$^{\mathcal{PS}}$ solved 3; *Yochan*^{COST} solved 3

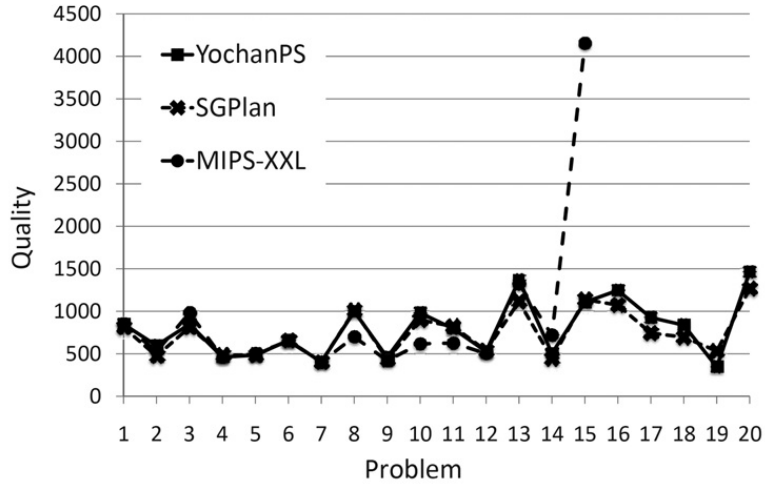
Fig. 9. IPC-5 *pathways* "simple preferences".

As shown in Fig. 10(a), *Yochan*$^{\mathcal{PS}}$ is able to solve each of the problems with quality that is competitive with the other IPC-5 participants. For this domain, the heuristic in *Yochan*$^{\mathcal{PS}}$ guides the search well, as it is made to discriminate between goals based on the cost of the actions to reach them. On the other hand, as shown in Fig. 10(b), *Yochan*^{COST} attempts to satisfy the goals in the cheapest way possible and, in the harder problems, always returns an empty plan and then fails to find a better one in the allotted time. Thus, *Yochan*^{COST} tends to find plans that trivially satisfy the newly introduced hard goals.

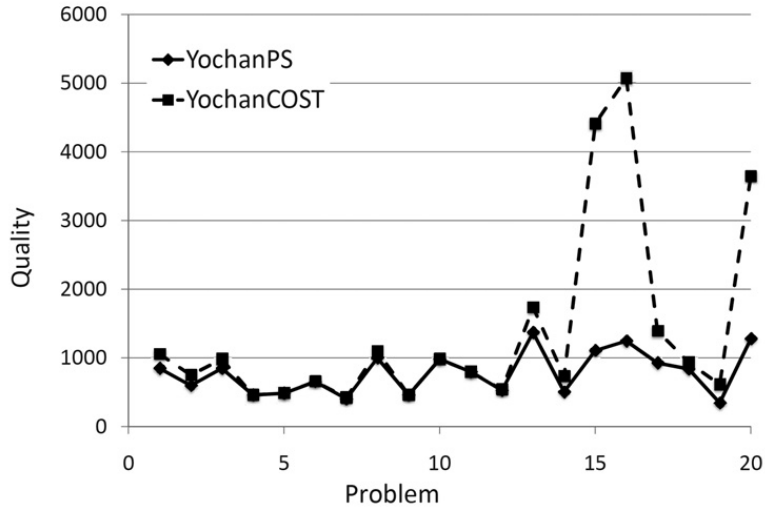
5.4. *The* storage *domain*

Here a planner must discover how to move crates from containers to different depots. Each depot has specific spatial characteristics that must be taken into account. Several hoists exist to perform the moving, and goals involve preferences for storing compatible crates together in the same depot. Incompatible crates must not be located adjacent to one another. Preferences also exist about where the hoists end up.

In this domain, both *Yochan*$^{\mathcal{PS}}$ and *Yochan*^{COST} failed in their grounding process beyond problem 5. Fig. 11(a) shows that, of the problems solved, *Yochan*$^{\mathcal{PS}}$ found solutions with better quality than MIPS-XXL. Fig. 11(b) shows that both *Yochan*$^{\mathcal{PS}}$ and *Yochan*^{COST} solved versions of *storage* that had universal and existential quantification compiled away from the goal preferences and produced plans of equal quality. Of the problems solved by both planners, the longest plan found in this domain by the two planners contain 11 actions (the same plan found by both planners).



(a) IPC-5 results. *Yochan*$^{\mathcal{PS}}$ solves 20; MIPS-XXL solves 15; SGPlan solves 20



(b) *Yochan*$^{\mathcal{PS}}$ vs. *Yochan*^{COST}. *Yochan*$^{\mathcal{PS}}$ solves 20; *Yochan*^{COST} solves 20

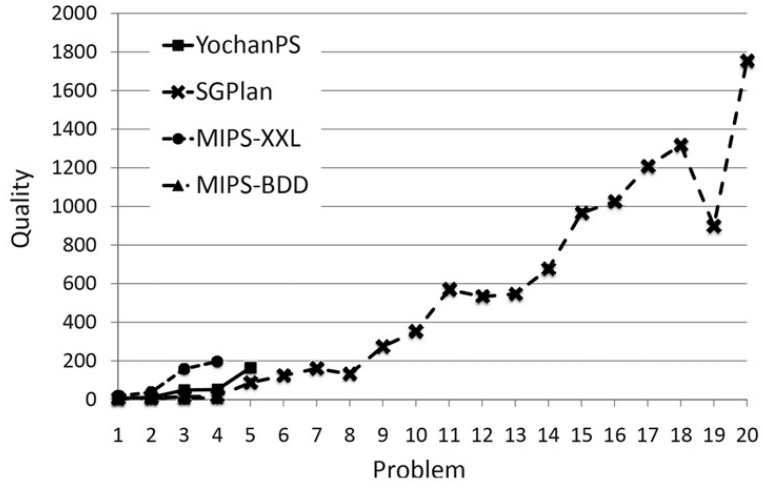
Fig. 10. IPC-5 *rovers* "simple preferences".

5.5. *The* TPP *domain*

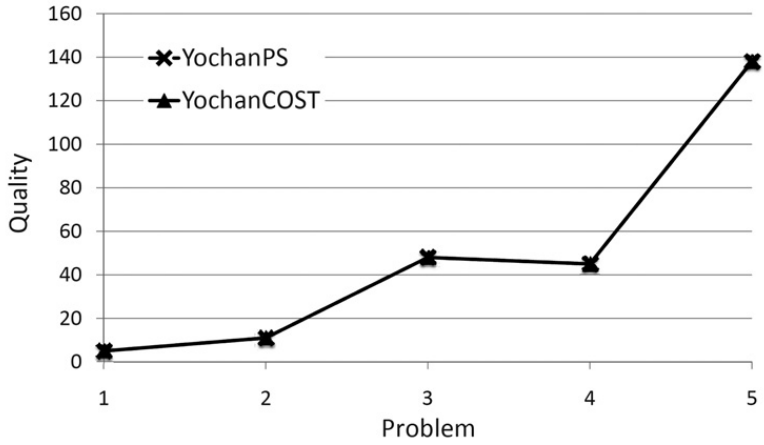
This is the traveling purchaser problem (TPP), a generalization of the traveling salesman problem. In this domain, several goods exist at various market locations. The object of the planning problem is to purchase some amount of each product while minimizing the cost of travel (i.e., driving a truck) and while also satisfying goal preferences. The *TPP* domain is unique in that it is the only one in the "simple preferences" track to have preference over action preconditions. When driving a truck away from a market, we always prefer to have all of the goods emptied at that market. Cost is added to the action if we fail to satisfy this condition. Like the trucks domain, this is a logistics-like domain. Goal preferences typically involve having a certain number of the various goods stored.

As we can see in Fig. 12(a), *Yochan*$^{\mathcal{PS}}$ finds plans of competitive quality in the problems that were solved. This domain has soft goals that are mutually exclusive from one another (i.e., storing various amounts of goods). Though the heuristic used in *Yochan*$^{\mathcal{PS}}$ does not identify this, it does focus on finding goals to achieve that may be of the highest quality. It turns out that, in *TPP*, this is enough. As the planner searches for a solution, it identifies this fact and looks for plans that can achieve the highest quality. It is interesting to note that *Yochan*$^{\mathcal{PS}}$ solves more problems than MIPS-XXL and MIPS-BDD. Also, when both find solutions, plans given by *Yochan*$^{\mathcal{PS}}$ are often of better quality.

As Fig. 12(b) shows, *Yochan*^{COST} has more difficulty finding solutions for this domain than *Yochan*$^{\mathcal{PS}}$. It attempts to minimize actions as well as cost (as does *Yochan*$^{\mathcal{PS}}$), but tends not to improve plan quality after finding a plan with a lower level of goods (involving fewer actions).



(a) IPC-5 results. *Yochan*$^{\mathcal{PS}}$ solves 5; MIPS-XXL solves 4; MIPS-BDD solves 4; SGPlan solves 20



(b) *Yochan*$^{\mathcal{PS}}$ vs. *Yochan*^{COST}. *Yochan*$^{\mathcal{PS}}$ solves 5; *Yochan*^{COST} solves 5

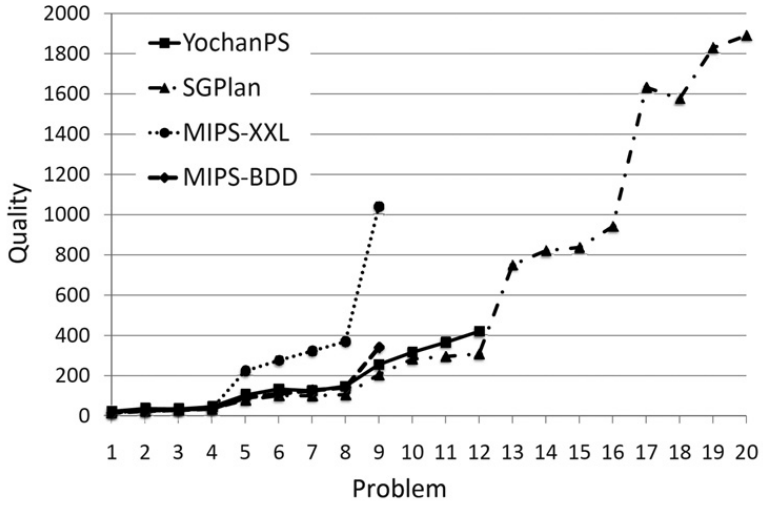
Fig. 11. IPC-5 *storage* "simple preferences".

Interestingly, a similarity exists between the anytime behavior of *Yochan*$^{\mathcal{PS}}$ and *Yochan*^{COST}. Typically, both planners discover initial plans at approximately the same rate, and when possible find incrementally better plans. In fact, only when *Yochan*$^{\mathcal{PS}}$ finds better solutions does the behavior significantly differ. And in these cases, *Yochan*$^{\mathcal{PS}}$ "reaches further" for more solutions. We largely attribute this to the heuristic. That is, by ignoring some of the goals in the relaxed plan, the planner essentially serializes the goals to focus on during search. At each search node *Yochan*$^{\mathcal{PS}}$ re-evaluates the reachability of each goal in terms of cost versus benefit. In this way, a goal can look more appealing at greater depths of the search.[14] This is especially noticeable in the *TPP* domain. In this domain, all of the higher-quality plans that *Yochan*$^{\mathcal{PS}}$ found were longer (in terms of number of actions) than those of *Yochan*^{COST} in terms of number of actions. This is likely because the relaxed plan heuristic in *Yochan*^{COST} believes preference goals are reachable when they are not.

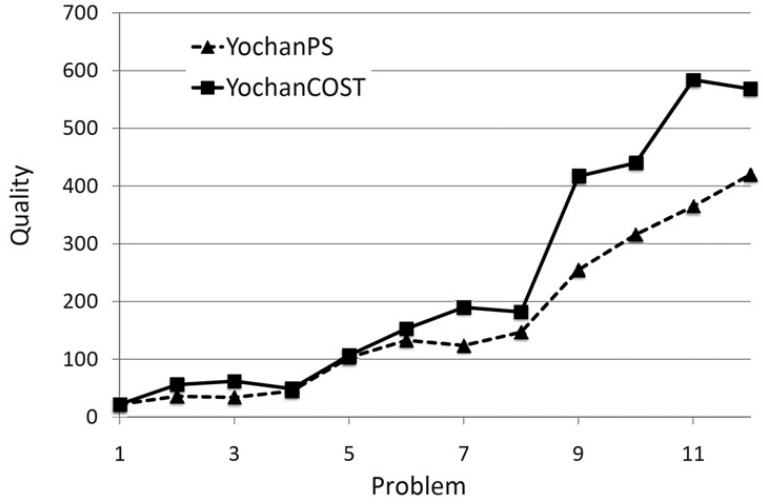
5.6. Other tracks

While *Yochan*$^{\mathcal{PS}}$ participated in the IPC-5 as a partial satisfaction planner capable of handling PDDL3.0, it is based on *Sapa* and therefore is capable of handling a wide variety of problem types. Because of this, the planner also participated in both the "metrictime" and "propositional" tracks. In the "metrictime" track, *Yochan*$^{\mathcal{PS}}$ performed quite well in terms of finding good quality (short makespan) plans, achieving 1st place in one domain (the "time" versions of *openstacks*) and 2nd place in three domains (the "time" version of *storage* and *trucks* and the "metrictime" version of *rovers*). The performance in

[14] We also note evidence of this exists by the fact that *Yochan*$^{\mathcal{PS}}$ tends to do better as problems scale-up.



(a) IPC-5 results. *Yochan*^{COST} solves 12; MIPS-XXL solves 9; MIPS-BDD solves 9; SGPlan solves 20



(b) *Yochan*$^{\mathcal{PS}}$ vs. *Yochan*^{COST}. *Yochan*$^{\mathcal{PS}}$ solves 12; *Yochan*^{COST} solves 12

Fig. 12. IPC-5 *TPP* "simple preferences" results.

these problems can be attributed to the action re-scheduling procedure of *Sapa*, which takes an original parallel, temporal plan and attempts to re-order its actions to shorten the makespan even more [14]. This especially holds for the *openstacks* problems, whose plans have a high amount of parallelism.

Looking at the results of *Yochan*$^{\mathcal{PS}}$ versus SGPlan for the temporal *openstacks* domain provides some further insight into this behavior. Even in the more difficult problems that *Yochan*$^{\mathcal{PS}}$ solves, the plans contained an equal or greater number of actions. However, *Yochan*$^{\mathcal{PS}}$ parallelized them to make better use of time using its action scheduling mechanism (which, again, was inherited from the planner *Sapa*).

Summary of IPC-5 results

Yochan$^{\mathcal{PS}}$ performs competitively in many domains. In the *trucks* domain *Yochan*$^{\mathcal{PS}}$ scaled better than MIPS-XXL and MIPS-BDD, but was outperformed overall in terms of number of problems solved by SGPlan, the winner of the competition.[15]

[15] The organizers of IPC-6 have alleged that the version of SGPlan that was entered in that competition seems to have code in it that could allow it to select among different planning strategies based on the name of the domain and other characteristics of the domain (e.g., number of actions, number of actions' preconditions, etc.). As of this writing, we do not know if the SGPlan version that participated in IPC-5 also had such domain customization, and how it might have affected the competition results.

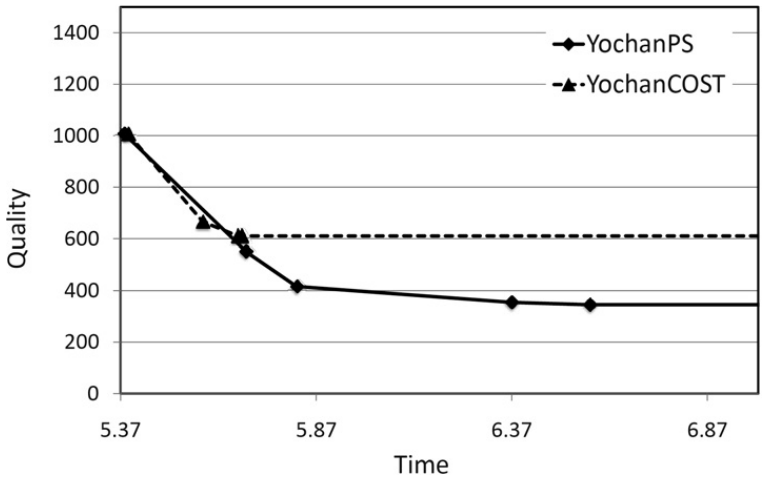


Fig. 13. IPC-5 Rovers, Problem 19 anytime search behavior.

There are several technical reasons for *Yochan*$^{\mathcal{PS}}$'s inability to solve large problems in many of the domains: *Yochan*$^{\mathcal{PS}}$'s parsing and grounding routine was quite slow and takes most if not all of the allocated 30 minutes time to parse large problems in many domains.

In several domains (*trucks*, *TPP*, and *rovers*), *Yochan*$^{\mathcal{PS}}$ predominately gave better quality plans than *Yochan*^{COST}. From the search behavior, in many cases the compilation to hard goals caused the planner to quickly choose naïve solutions (i.e., trivially achieving the hard goals without achieving the preference) despite the additional cost associated with doing so. This is attributed to the fact that the heuristic also minimizes the number of actions in the plan while minimizing cost (since the heuristic counts all non-preference actions with a cost 1). While this same quality exists in the heuristic used by *Yochan*$^{\mathcal{PS}}$, handling *soft* goals directly helps the planner by allowing it to completely avoid considering achievement of goals. In other words, the planner can focus on satisfying only those goals that it deems beneficial and can satisfy some subset of them without selecting actions that "grant permission" to waive their achievement.

A view into the behavior of the anytime search between the two planners helps illustrate what is happening. Fig. 13 shows the search behavior on problem 19 of the *rovers* domain from the competition. We can see that, while *Yochan*$^{\mathcal{PS}}$ and *Yochan*^{COST} find plans of similar quality initially, *Yochan*^{COST} stops finding better plans while *Yochan*$^{\mathcal{PS}}$ continues. This is typical throughout *rovers*, *TPP* and some of the *trucks* problems. In these domains, as the problems scale in size, *Yochan*^{COST} typically exhibits this behavior. In both planners, it is very often the case that initial solutions are quickly found, as single goals can often be quickly satisfied.[16]

Note that one issue with *Yochan*^{COST} is that the number of "dummy" actions that must be generated can effect its search. For every step, the actions to decide to "not achieve the goal" can be applicable, and therefore must be considered (such that a node is generated for each one). This can quickly clog the search space, and therefore results in a disadvantage to the planner as the scale of the problems increases. *Yochan*$^{\mathcal{PS}}$, on the other hand, by directly handling soft goals, can avoid inserting such search states into the space, thereby increasing its scalability over *Yochan*^{COST}.

5.7. Up-front goal selection in competition domains

While *Sapa*$^{\mathcal{PS}}$, and by extension *Yochan*$^{\mathcal{PS}}$, performs goal re-selection during search, one can also imagine dealing with soft goals by selecting them before the planning process begins. Afterward, a planner can treat the selected goals as *hard* and plan for them. The idea is that this two-step approach can reduce the complexities involved with constantly re-evaluating the given goal set, but it requires an adequate technique for the initial goal selection process. Of course, performing optimal goal selection is as difficult as finding an optimal plan to the original PSP Net Benefit problem. However, one can imagine attempting to find a feasible set of goals using heuristics to estimate how "good" a goal set is. But, again, proving the satisfiability of goals requires solving the entire planning problem or at least performing a provably complete analysis of the mutual exclusions between the goals (which is as hard as solving the planning problem).

Given that hard goals must be non-mutex, one may believe that in most domains mutually exclusive soft goals would be rare. However, users can quite easily specify soft goals with complex mutexes lingering among them. For instance, consider a blocks world-like domain in which the soft goals involve blocks stacked variously. If we have three blocks (*a*, *b*, and *c*) with the soft goals *(on a b)*, *(on b c)*, and *(on c a)*, we have a ternary mutual exclusion and we can at best achieve only two of the goals at a time. For any number of blocks, listing every stacking possibility will always generate *n*-ary mutexes, where *n* can be as large as the number of blocks in the problem.

[16] By "initial solution" we, of course, mean a plan other than the "null plan" when all goals are soft.

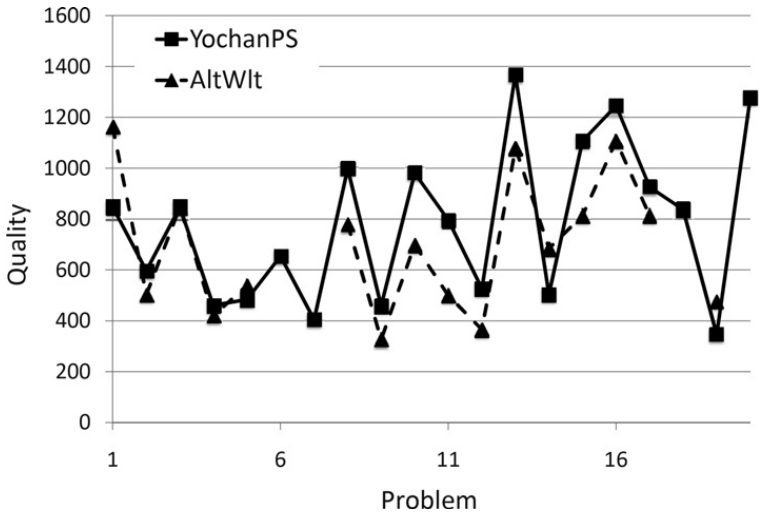


Fig. 14. Comparison with *AltWlt* on IPC-5 *rovers* domain.

Further, the IPC-5 "simple preferences" domains have many n-ary mutual exclusions between goals with sometimes complex interactions such that the satisfaction of one set of goals may be negatively dependent upon the satisfaction of another set of goals (i.e., some goal sets are mutex with other goal sets). It turns out that even when binary mutexes are taken into account, as is done with the planner *AltWlt* (which is an extension of the planner *AltAlt*$^{\mathcal{PS}}$), these complex interactions cannot be detected [38].

Specifically, the planner *AltWlt* uses a relaxed planning graph structure to "penalize" the selection of goals that appear to be binary mutually exclusive by solving for each goal individually, then adding cost to relaxed plans that interfere with already-chosen goals. In other words, given a relaxed plan for a selected goal g called r_g, and a relaxed plan for a candidate goal g', $r_{g'}$, we have a penalty cost c for the selection of g' if any action in $r_{g'}$ interferes with an action in r (i.e., the effects of actions in $r_{g'}$ delete the preconditions found in r_g in actions at the same step). A separate penalty is given if preconditions in the actions of $r_{g'}$ are binary and statically mutex with preconditions in the actions of r_g and the maximum of the two penalties is taken. This is then added to the cost propagated through the planning graph for the goal. *AltWlt* then greedily selects goals by processing each relaxed plan in turn, and selects the one that looks most beneficial.

To see if this approach is adequate for the competition benchmarks, we converted problems from each of the five domains into a format that can be read by *AltWlt*. We found that in *storage*, *TPP*, *trucks*, and *pathways*, *AltWlt* selects goals but indicates that there exists no solution for the set it selects. However, *AltWlt* found some success in *rovers*, a PSP NET BENEFIT domain where mutual exclusion between goals is minimum in the benchmark set. The planner was able to solve 16 of the 20 problems, while *Yochan*$^{\mathcal{PS}}$ was able to solve all 20. Of the ones *AltWlt* failed to solve, it explicitly ran out of memory or gave errors. Fig. 14 shows the results. In 12 of the 16 problems, *AltWlt* is capable of finding better solutions than *Yochan*$^{\mathcal{PS}}$. *AltWlt* also typically does this faster. As an extreme example, to find the eventual final solution to problem 12 of *rovers*, *Yochan*$^{\mathcal{PS}}$ took 172.53 seconds while *AltWlt* took 324 milliseconds.

We believe that the failure of *AltWlt* on the other competition domains is not just a bug, but rather a fundamental inability of its up-front objective selection approach to handle goals with complex mutual exclusion relations. To understand this, consider a slightly simplified version of the simple preferences *storage* domain from the IPC-5. In this domain we have crates, storage areas, depots, load areas, containers and hoists. Depots act to group storage areas into a single category (i.e., there are several storage areas within a single depot). Hoists can deliver a crate to a storage area adjacent to it. Additionally, hoists can move between storage areas within a depot, and through load areas (which connect depots). When a crate or hoist is in a storage area or load area, then no other hoist or crate may enter into the area. Crates begin by being inside of a container in a load area (hence the load area is initially passable, as no crates are actually inside of it).

Fig. 15 shows the layout in our example (which is a simplified version of problem 1 from the competition). In the problem there exists a hoist, a crate, a container, two depots ($depot_0$ and $depot_1$) and two storage areas in each depot (sa_{0-0}, sa_{0-1} in $depot_0$ and sa_{1-0}, sa_{1-1} in $depot_1$). The storage areas are connected to each other, and one in each depot is connected to the loading area. The crate begins inside of the container and the hoist begins at in $depot_1$ at sa_{1-0}. We have several preferences: (1) the hoist and crate should end up in different depots (with a violation penalty of 1), (2) the crate should be in $depot_0$ (violation penalty of 3), (3) the hoist should be in sa_{0-0} or sa_{0-1} (violation penalty of 3), (4) sa_{1-0} should be clear (i.e., contains neither the hoist nor the crate with a violation penalty of 2), and (5) sa_{0-1} should be clear (violation penalty of 2).

The (shortest) optimal plan for this problem involves only moving the hoist; specifically, moving the hoist from its current location, sa_{1-0}, to sa_{0-1} (using 3 moves). This satisfies preference (1) because the crate is in no depot (hence it will always be in a "different depot" than the hoist), (3) because the hoist is in sa_{0-1}, (4) because sa_{1-0} is clear and (5) because sa_{0-1} is clear. It violates the soft goal (2) with a penalty cost of 3. Of course, finding the optimal plan would be nice, but

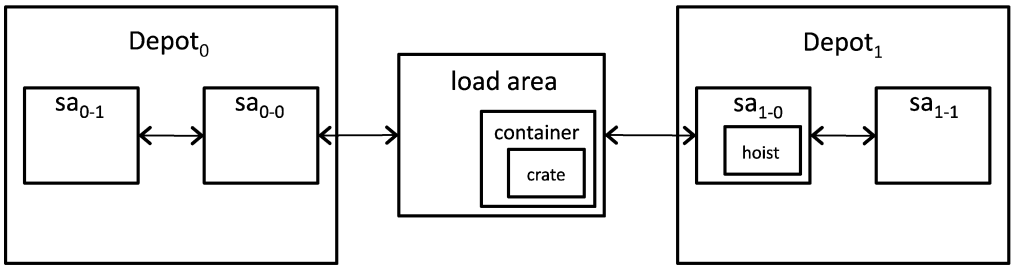


Fig. 15. An example of the "simple preferences" storage domain.

we would also be satisfied with a feasible plan. However, there is a heavy burden on the goal selection process to find a satisfiable, conjunctive set. In this problem the "simple preference" goals have complex, non-binary mutual exclusions.

Consider the *AltWlt* procedure for finding a set of goals for this domain. *AltWlt* selects goals greedily in a non-deterministic way. But the important aspect of *AltWlt* here is how it defines its penalty costs for noticing mutual exclusion between goals. Interference involves the effect of one action deleting the precondition of another action. However, there are often several ways of satisfying a preference, most of which do not interfere with satisfying another preference in the relaxed setting. For instance, consider preference (1), that we should have the create and hoist in different depots. A preference of this form essentially involves several discrete disjunctive clauses, (e.g., "do not have the hoist at sa_{1-1} *or* do not have the crate in $depot_1$"). Satisfying for one of these clauses is sufficient to believe that the preference can be achieved. If we achieve one of these (e.g., "do not have the hoist at sa_{1-1}"), the clause is satisfied. Of course even in the relaxed problem, we must satisfy each of the disjunctive clauses (e.g., we can have each of "do not have the hoist at sa_{x-y} where $x, y \in \{0, 1\}$" or "do not have the crate in $depot_x$ where $x \in \{0, 1\}$"). It turns out that these are satisfiable in the initial state, so this is a trivial feat. If we then choose goal preference (2), having the crate in $depot_0$, we can find a relaxed plan that moves the hoist to the load area, removes the crate from the container and places it in sa_{0-0} (which is in $depot_0$). Satisfying (3), having the hoist at sa_{0-0} or sa_{0-1} looks statically mutex with (1), but the competing needs or interference penalty costs apply only when a relaxed plan exists. Since none exists for (1), *AltWlt* finds a relaxed plan that moves the hoist to sa_{0-1}.[17] Satisfying preference goal (4) requires that we move a single step–easily satisfiable, and sharing an action with (2), and hence there exists no interference or competing needs. Preference goal (5) is satisfied at the initial state.

From this analysis, we can see that *AltWlt* selects each of the goals, as there exist no penalties to make them look unappealing. It will subsequently fail when attempting to find a solution for the goals–there exists no way to satisfy for all of the preferences. The complex mutual exclusions and disjunctive clauses cause *AltWlt* to select goal sets that are impossible to achieve. From the point of view of the competition, *AltWlt* suffers from similar issues in all but one of the "simple preference" domains (namely, the "simple preferences" version of *rovers*).

In summary, while up-front selection of objectives does make the PSP Net Benefit problem much easier to handle, as we have suspected, in complex domains the objective selection cannot even guarantee satisficing plans.

6. Related work

In the last few years, there has been considerable work on planning with goals as soft constraints or preferences. Problems tackled include those with either quantitative or qualitative goal preferences. The solving methods also range from various heuristic approaches to compilations for simplifying the soft goal constraints. In this section, we will compare $Sapa^{\mathcal{PS}}$ and $Yochan^{\mathcal{PS}}$ with them as well as explore the similarities and differences between our variation of A^* and other well-known search algorithms. A further overview on planning and scheduling with preferences and soft constraints can be found in [16].

6.1. Planners solving PSP and its close variations

There are several planners that solve the same PSP and closely related problems. Two recent heuristic planners that solve PSP Net Benefit are the orienteering-planner (OP) [40] and $AltAlt^{\mathcal{PS}}$ [44]. Both OP and $AltAlt^{\mathcal{PS}}$ use a two-step framework. In step 1, they heuristically select the subset S of soft goals. In step 2, they convert S into hard goals and use a non-PSP planner to find the lowest cost plan achieving S. For step 1, OP uses the solution of a simpler problem to select both the subset of goals and the order to achieve them. The abstract problem is built by first propagating the action costs on the planning graph and constructing the *orienteering* problem, which is a variation of the traveling salesman problem. Unlike the orienteering-planner, $AltAlt^{\mathcal{PS}}$ relies on the cost-sensitive planning graph and uses a different technique to analyze the graph to heuristically select the most beneficial subset of goals. After the goals are found, $AltAlt^{\mathcal{PS}}$ uses a variation of the regression search planner *AltAlt* to search for a low cost plan.

[17] Even if a relaxed plan were to exist for (1), the disjunctive clauses make interference difficult to identify–i.e., we can be satisfying for "do not have the crate in $depot_x$" which is not mutex with preference (3).

The main advantage of the two-step approach used by OP and *AltAlt*$^{\mathcal{PS}}$ is that up-front goal selection enables the reduction to a planning problem with hard goals (and action costs) which can be solved by any planner capable of handling such problems. The disadvantage of this approach is that if the heuristics in the first step do not select the right set of goals then the planner may either find a poor quality plan or can take a lot of time to discover that the problem is unsolvable before it can switch to another goal set. Therefore, if the first step does not select the *exact* optimal goal set, then the final plan is not guaranteed to be optimal. Moreover, if there is an unachievable goal selected, then the planner will return failure before trying to select another set of goals. Indeed, as shown in Section 5.7, *AltAlt*$^{\mathcal{PS}}$ and its improved version *AltWlt* never try to solve more than a single (hard) goal set and consistently select the set of goals containing non-obvious mutexes.[18]

Sapa$^{\mathcal{PS}}$ is different from those two planners in the sense that it does not rely on any pre-defined subset of goals and lets the A^* framework decide which goals are the most beneficial for a given node during search. Therefore, it can partially correct the mistakes in heuristically selecting a particular subset of goals at each search node as it goes deeper in the search tree. *Sapa*$^{\mathcal{PS}}$ also works in an *anytime* fashion and keeps improving its solution quality given more search time. Nevertheless, the two types of planners can complement each other. The heuristic framework used in the orienting-planner and *AltAlt*$^{\mathcal{PS}}$ can be employed in *Sapa*$^{\mathcal{PS}}$ to improve its heuristic evaluation at each search node. However, it can be quite expensive to do so for each search node.

As mentioned previously, Keyder and Geffner [33] introduced a heuristic planner that is able to avoid the two-step solving approach in *AltAlt*$^{\mathcal{PS}}$ and OP by compiling all soft goals into hard goals. The newly introduced actions and fluents guarantee that the lowest cost plan in the new problem corresponds to the highest-benefit plan in the original problem. This compilation approach shares a lot of similarities with *Yochan*^{COST}. However, the compilation approach in *Yochan*^{COST} is more complicated due to the more complex preference model in PDDL3.0, the existence of disjunctions on preference formulas, and the potential necessity to delete and re-achieve goals. As we have seen from our experiments, *Yochan*^{COST} tends to fare worse than *Yochan*$^{\mathcal{PS}}$. Because of this, we believe that handling soft goals directly provides better heuristic guidance.

OptiPlan [44] extends an integer linear programming (ILP) encoding for bounded parallel length classical planning to solve the PSP problem by adding action cost and goal utility. It also relaxes the hard goal constraints by moving those goals satisfying conditions into the ILP's objective function. This way, goals can be treated as soft constraints. The advantage of *OptiPlan*'s approach is that off-the-shelf ILP solvers can be used to find the final plan that is guaranteed to be optimal up to a bounded parallel plan length. The disadvantage of this approach is that it does not scale up well as compared with heuristic approaches (such as those used by *Sapa*$^{\mathcal{PS}}$ and *Yochan*$^{\mathcal{PS}}$).

There have been some recent extensions to the basic PSP problem definition. SPUDS and iPud [12], and BBOP-LP [5] have extended *Sapa*$^{\mathcal{PS}}$ and *OptiPlan* to solve PSP problems where there are utility-dependencies between goals. Thus, achieving a set of goals may have a higher or lower utility than the sum of the utilities of individual goals, depending on user-defined relations between them. The heuristics in *Sapa*$^{\mathcal{PS}}$ and the objective function in *OptiPlan* have been extended significantly in those planners to accommodate the new constraints representing dependencies between goals.

Bonet and Geffner [8] present a planner whose search is guided by several heuristics approximating the optimal relaxed plan using the rank of d-DNNF theory. While the search framework is very similar to *Sapa*$^{\mathcal{PS}}$ and the heuristic is also relaxed plan-based, the problem tackled is a variation of PSP where goal utilities are not associated with facts achieved at the end of the plan execution but achieved *sometime* during the plan execution. This way, it is a step in moving from the PSP definition of traditional "at end" goals to a more expressive set of goal constraints on the plan trajectory defined in PDDL3.0. While the heuristic estimate is likely to be more accurate than *Sapa*$^{\mathcal{PS}}$, the heuristic computation is more expensive due to the required step of compiling the problem to d-DNNF.

6.2. PDDL3.0 planners

Several competing planners (besides *Yochan*$^{\mathcal{PS}}$) were able to solve various subsets of PDDL3.0 in the IPC-5, specifically SGPlan [32], MIPS-XXL [18], MIPS-BDD [17] and HPlan-P [2]. Like *Yochan*$^{\mathcal{PS}}$, these planners use a forward heuristic search algorithm but none convert PDDL3-SP into PSP like *Yochan*$^{\mathcal{PS}}$. Besides SGPlan, each planner compiles PDDL3.0 preferences into another planning representation and then changes the heuristic approach to find good quality plans given the costs associated with preferences defined in PDDL3.0.

Baier et al. [2] compile trajectory preferences into additional predicates and actions by first representing them as a non-deterministic finite state automata (NFA). The heuristic is then adjusted to take into account that different preferences have different values so that the planner is guided toward finding overall good quality plans. The planner is then extended in [1] to have a more sophisticated search algorithm where conducting a planning search and monitoring the parametrized NFA are done closely together. MIPS-XXL [18] and MIPS-BDD [17] both compile plan trajectory preferences into Büchi automata and "simple preferences" into PDDL2.1 numerical fluents that are changed upon a preference violation. MIPS-XXL then uses Metric-FF with its enforced hill-climbing algorithm to find the final solution. On the other hand, MIPS-BDD stores the expanded search nodes in BDD form and uses a bounded-length cost-optimal BFS search for BDDs to solve the compiled problems. While compiling to NFA seems to allow those planners to handle a wider subset of PDDL3.0 preferences than

[18] However, the orienteering-planner has a strategy for avoiding this problem, by selecting one goal at a time to achieve.

Yochan$^{\mathcal{PS}}$, it is not clear if there is any performance gain from doing so. SGPlan [32] uses partition techniques to solve planning problems; it does not compile away the preferences but uses the costs associated with violating trajectory and simple preferences to evaluate partial plans.

There are planners that solve planning problems with trajectory preferences in PDDL3.0 by compiling them to satisfiability (SAT) [26] or ILP [43]. The SAT compilation can be done by first finding the maximally achievable plan quality value C, then $n = \lceil \log_2(C) + 1 \rceil$ ordered bits $b_1, \ldots, b_n$ are used to represent all possible plan quality values within the range of 0 to C. A SAT solver with modified branching rules over those b_i bits is then used to find a bounded-length plan with the maximum achievable plan quality value. Due to the limitation of SAT in only supporting binary variables, the SAT-compilation approach is arguably awkward.

It is easier to support quantitative preferences in ILP due to its natural ability to support real values and an objective function to optimize plan quality. van den Briel et. al. [43] have shown various examples of how to compile trajectory preferences into ILP constraints. The overall framework is to: (1) obtain the logical expression of the preferences; (2) transform those expressions into CNF constraints in SAT; (3) formulate the ILP constraints corresponding to the resulting SAT clauses; and (4) set up the objective function based on the preference violation cost of those ILP constraints. Both the SAT and ILP compilation approaches do not scale up well compared to the heuristic search approach used in *Sapa*$^{\mathcal{PS}}$ and *Yochan*$^{\mathcal{PS}}$. The advantage is that they can capitalize on state-of-the-art solvers in other fields to solve complex planning problems.

6.3. Qualitative preference planners

There is another class of planners that also treats goals as soft constraints; however, goals are not quantitatively differentiated by their utility values, but their preferences are instead qualitatively represented. Qualitative preferences are normally easier to elicit from users, but they are less expressive and there can be many plans that are *incomparable*. Brafman and Chernyavsky [10] use TCP-Nets to represent the qualitative preferences between goals. Some examples are: (1) $g_1 \succ g_2$ means achieving g_1 is preferred to achieving g_2; (2) $g_1 \succ \neg g_1$ means achieving g_1 is better than not achieving it. Using the goal preferences, plan P_1 is considered better than plan P_2 if the goal set achieved by P_1 is preferred to the goal set achieved by P_2 according to the pre-defined preferences. A Pareto optimal plan P is the plan such that the goal set achieved by P is not dominated (i.e., preferred) by the goal set achieved by any other plan. A CSP-based planner is used to find the bounded-length optimal plan. This is accomplished by changing the branching rules in the CSP solver so that the most preferred goal and the most preferred value for each goal are always selected first. Thus the planner first branches on the goal set ordering according to goal preferences before branching on actions making up the plan. Like the extension from PSP to PDDL3.0 quantitative preference models on plan trajectories, there have also been extensions from qualitative goal preferences to qualitative plan trajectory preferences. Tran and Pontelli [42] introduced the PP language that can specify qualitative preferences on plan trajectories such as preferences over the states visited by the plan or over actions executed at different states. PP uses a nested subset of temporal logic (similar to PDDL3.0) to increase the set of possible preferences over a plan trajectory. PP is later extended with quantification and variables by Bienvenu et al. [6]. Both logic-based [42] and heuristic search based [6] planners have been used to solve planning with qualitative preferences represented in PP by using weighting functions to convert qualitative preferences to quantitative utility values. This is due to the fact that quantitative preferences such as PSP and PDDL3.0 fit better with the heuristic search approach that relies on a clear way to compute and compare g and h values. The weights are then used to compute the g and h values guiding the search for an optimal or good quality solution.

6.4. Other PSP work

The PYRRHUS planning system [46] considers an interesting variant of temporal partial satisfaction planning where goals have deadlines. In PYRRHUS, the quality of the plan is measured by the utilities of goals and the amount of resources consumed. Goals have deadlines, and utilities of goals decrease if they are achieved later than their deadlines. Unlike PSP and PDDL3.0 problems, all the logical goals still need to be achieved by PYRRHUS for the plan to be valid. In other words, the logical aspect of the goals (i.e., the atemporal aspect) are still hard constraints while the goal deadline constraints (i.e., the temporal aspect) are soft constraints and can be "partially satisfiable". For solving this problem, PYRRHUS uses a partial order planning framework guided by domain-dependent knowledge. Thus, it is not a domain-independent planner as are the other planners discussed in this paper.

One way of solving PSP problems is to model them directly as deterministic MDPs [30], where actions have different costs. Any state S in which any of the goals hold is a terminal state with the reward defined as the sum of the utilities of the goals that hold in S. The optimal solution to the PSP problem can then be extracted from the optimal policy of this MDP. Given this, *Sapa*$^{\mathcal{PS}}$ can be seen as an efficient way of directly computing the plan without computing the entire policy (in fact, $h^*(S)$ can be viewed as the optimal value of S). Our preliminary experiments with a state-of-the-art MDP solver show that while direct MDP approaches can guarantee optimality, they scale very poorly in practice and are unable to solve even small problems.

Over-subscription issues have received more attention in the scheduling community. Earlier work in over-subscription scheduling used "greedy" approaches, in which tasks of higher priorities are scheduled first [34,39]. More recent efforts have used stochastic greedy search algorithms on constraint-based intervals [20], genetic algorithms [27], and iterative repairing

techniques [35] to solve this problem more effectively. Some of those techniques can potentially help PSP planners to find good solutions. For example, scheduling tasks with higher priorities shares some similarity with the way $AltAlt^{ps}$ builds the initial goal set, and iterative repairing techniques may help local search planners such as LPG in solving PSP problems.

6.5. Our variation of A^* vs. other closely related search algorithms

For the rest of this section, we will discuss search algorithms closely related to our search algorithm, which was discussed in Section 3.1.

vs. variations of anytime A^*: The main difference between best-first heuristic search algorithms such as A^* and our algorithm is that one deals with minimizing path length in a graph with only *positive* edge costs and the other works to maximize the *path length* in a graph with both *positive* and *negative* edge benefits. The other difference is that any node can be a goal node in PSP Net Benefit problems. Nevertheless, only ones with higher net benefit than the initial state are interesting and can potentially provide better solutions than the empty plan. Turning from maximization to minimization can be done by negating the edge benefit to create "edge cost", resulting in a new graph G'. However, we cannot convert G' into an equivalent graph with only positive edges, which is a condition for A^*. Compare this search to other anytime variations of the A^* algorithm such as Anytime-WA^* [28] or ARA^* [36]. Both of these anytime variations are based on finding a non-optimal solution first using an *inadmissible* heuristic and later gradually improving the solution quality by branch and bound or gradually lowering the weight in the heuristic estimate $f = g + w \cdot h$. Given that there are both positive and negative edges in PSP Net Benefit, our algorithm may generate multiple solutions with gradually improving quality before the optimal solution is found regardless of the admissibility of the heuristic. This is due to the fact that there are potentially many solutions on the path to a better quality, or optimal solution. Perhaps the closest work to our algorithm is by Dasgupta et al. [11] which searches for the shortest path in a directed acyclic graph with negative edges. However, in that case a single goal node is still pre-defined. We also want to note that besides the *OPEN* list, there is no *CLOSED* list used in Algorithm 2, which is popular in most shortest-path graph search algorithms to facilitate duplicate detection. Our algorithm is implemented in a metric temporal planner with a complicated state representation and duplicate states are rarely generated. Therefore, we did not implement the *CLOSED* list. However, one can be added to our algorithm similar to the way a *CLOSED* list is used in A^*. Besides admissibility, another important property of the heuristic is *consistency* (i.e., $h(s) \leqslant h(s') + c_e$ with s' is a successor of s), which allows A^* with duplicate detection to expand each node at most once in graph search. It can be shown that our algorithm has a similar property. That is, if the heuristic is consistent (i.e., $h(s) \geqslant h(s') + b_e$), then our algorithm with duplicate detection will also expand each node at most once.

vs. Bellman–Ford Algorithm: The Bellman–Ford algorithm solves the single-source shortest path problem for a graph with both positive and negative edges (in a digraph). We can use this algorithm to solve PSP Net Benefit problems with the edge weights defined by the negation of the edge benefit. However, this algorithm requires enumerating through all plan states and can be very costly for planning problems which normally have a very large number of possible states. Moreover, we only need to find a single optimal solution in a digraph with no negative cycle and, thus, the additional benefit of the Bellman–Ford algorithm such as negative cycle detection and shortest path to all states are not needed. However, given the relations between the A^* and Dijkstra algorithms that can be used to prove some properties of A^* and the relations between the Dijkstra and Bellman–Ford algorithms (generalization from an undirected graph to a digraph), we can potentially prove similar properties of our algorithm by exploiting the relations between our algorithm and the Bellman–Ford algorithm.

vs. Branch and Bound: Many anytime algorithms share similarities with the branch and bound search framework in the sense that at any given time during the search, a best found solution s is kept and used as a bound to cutoff potential paths that lead to solutions proved worse than s. Our algorithm is no exception, as it uses the best found solution represented by B_B to filter nodes from the OPEN list and prevent newly generated nodes of lower quality from being added to the OPEN list as shown in Algorithm 2.

7. Conclusion and future work

In this paper, we present a heuristic search approach to solve partial satisfaction planning problems. In these problems, goals are modeled as soft constraints with utility values, and actions have costs. Goal utility represents the value of each goal to the user and action cost represents the total resource cost (e.g., time, fuel cost) needed to execute each action. The objective is to find the plan that maximizes the trade-off between the total achieved utility and the total incurred cost; we call this problem PSP Net Benefit. Previous PSP planning approaches heuristically convert PSP Net Benefit into STRIPS planning with action cost by pre-selecting a subset of goals. In contrast, we provide a novel anytime search algorithm that handles soft goals directly. Our new search algorithm has an anytime property that keeps returning better quality solutions until the termination criteria are met. This search algorithm, along with the relaxed plan heuristics adapted to PSP Net Benefit problems, were implemented in the forward state-space planner $Sapa^{\mathcal{PS}}$.

Besides $Sapa^{\mathcal{PS}}$, we also presented $Yochan^{\mathcal{PS}}$, a planner that converts "simple preferences" in the standard language PDDL3.0 into PSP Net Benefit and uses $Sapa^{\mathcal{PS}}$ to find good quality solutions. $Yochan^{\mathcal{PS}}$ recognizes the similarities between (1) goal and action precondition preference violation costs in PDDL3-SP, and (2) goal utility and action cost in PSP Net Benefit. It uses these similarities to create new goals and actions in PSP Net Benefit with appropriate utility and cost values to represent the original preferences. $Yochan^{\mathcal{PS}}$ participated in the 5^{th} International Planning Competition (IPC-5)

and was competitive with other planners that can handle PDDL3-SP, receiving a "distinguished performance" award. While SGPlan, the winner of the competition, solves many more problems than *Yochan*$^{\mathcal{PS}}$, our planner returns comparable quality solutions (which is the emphasis of the IPC) to SGPlan, in problems it can solve. There are several technical reasons for our planner's inability to solve many problems. *Yochan*$^{\mathcal{PS}}$'s parsing and grounding routine was quite slow and took most if not all of the allocated 30 minutes time to parse big problems in many domains. When *Yochan*$^{\mathcal{PS}}$ can ground the competition problems in a reasonable time, it typically can solve them.

We also introduce another planner called *Yochan*^{COST}. Like *Yochan*$^{\mathcal{PS}}$, it compiles away preferences in PDDL3-SP. However, the resulting problem is not PSP Net Benefit but a problem with hard goals and action costs. Our empirical results show that *Yochan*$^{\mathcal{PS}}$ performs better than *Yochan*^{COST} by handling soft goals directly.

We want to further explore our current approach of solving PSP problems in several directions. Even though we have used a forward planner, the anytime search algorithm presented in this paper can be used for other types of heuristic search planners such as regression or partial order causal link planners. It would be interesting to compare which is a better-suited planning framework. We have also expanded the basic PSP Net Benefit framework to include metric goals with variable goal utility in the planner *Sapa*$^{\mathcal{MPS}}$ [3], and logical goals with inter-dependent utilities in the planner SPUDS [12]. We are currently planning on extending it to handle goals whose utilities depend on their achievement time.

Acknowledgements

This article is in part edited and extended from Do and Kambhampati [15]; Benton, Kambhampati and Do [4]; and van den Briel, Sanchez, Do and Kambhampati [44]. We would like to thank Romeo Sanchez for help with his code and the experiments, and Menkes van den Briel and Rong Zhou for their discussions and helpful comments on this paper. We also greatly appreciate the assistance of William Cushing for discussions and help with experiments. Additionally, we express our gratitude to David Smith and Daniel Bryce for their suggestions and discussions in the initial stages of this work. And of course thanks go to Sylvie Thiébaux and the anonymous reviewers, who gave valuable comments that helped us to improve the article. This research is supported in part by the ONR grants N000140610058 and N0001407-1-1049 (MURI subcontract from Indiana University), a Lockheed Martin subcontract TT0687680 to ASU as part of the DARPA Integrated Learning program, and the NSF grant IIS-308139.

References

[1] J. Baier, F. Bacchus, S. McIllraith, A heuristic search approach to planning with temporally extended preferences, in: Proceedings of IJCAI-07, 2007.
[2] J. Baier, J. Hussell, F. Bacchus, S. McIllraith, Planning with temporally extended preferences by heuristic search, in: Proceedings of the ICAPS Booklet on the Fifth International Planning Competition, 2006.
[3] J. Benton, M.B. Do, S. Kambhampati, Over-subscription planning with numeric goals, in: Proceedings of IJCAI, 2005, pp. 1207–1213.
[4] J. Benton, S. Kambhampati, M. Do, YochanPS: PDDL3 simple preferences as partial satisfaction planning, in: Proceedings of the ICAPS Booklet on the Fifth International Planning Competition, 2006.
[5] J. Benton, M. van den Briel, S. Kambhampati, A hybrid linear programming and relaxed plan heuristic for partial satisfaction planning problems, in: Proceedings of ICAPS, 2007.
[6] M. Bienvenu, C. Fritz, S. McIllraith, Planning with qualitative temporal preferences, in: Proceedings of KR-06, 2006.
[7] B. Bonet, G. Loerincs, H. Geffner, A robust and fast action selection mechanism for planning, in: Proceedings of AAAI-97, 1997.
[8] B. Bonet, H. Geffner, Heuristics for planning with penalties and rewards using compiled knowledge, in: Proceedings of KR-06, 2006.
[9] C. Boutilier, T. Dean, S. Hanks, Decision-theoretic planning: Structural assumptions and computational leverage, Journal of Artificial Intelligence Research 11 (1999) 1–91.
[10] R.I. Brafman, Y. Chernyavsky, Planning with goal preferences and constraints, in: Proceeding of ICAPS-05, 2005.
[11] P. Dasgupta, A. Sen, S. Nandy, B. Bhattacharya, Searching networks with unrestricted edge costs, IEEE Transactions on Systems, Man, and Cybernetics.
[12] M.B. Do, J. Benton, S. Kambhampati, M. van den Briel, Heuristic planning with utility dependencies, in: Proceedings of IJCAI-07, 2007.
[13] M.B. Do, S. Kambhampati, Sapa: A multi-objective metric temporal planer, Journal of Artificial Intelligence Research 20 (2002) 155–194.
[14] M.B. Do, S. Kambhampati, Improving the temporal flexibility of position constrained metric temporal plans, in: Proc. of ICAPS-03, 2003.
[15] M.B. Do, S. Kambhampati, Partial satisfaction (over-subscription) planning as heuristic search, in: Proceedings of KBCS-04, 2004.
[16] M.B. Do, T. Zimmerman, S. Kambhampati, Planning and scheduling with over-subscribed resources, preferences, and soft constraints, in: Tutorial given at AAAI-07, 2007.
[17] S. Edelkamp, Optimal symbolic pddl3 planning with mips-bdd, in: Proceedings of the ICAPS Booklet on the Fifth International Planning Competition, 2006.
[18] S. Edelkamp, S. Jabbar, M. Nazih, Large-scale optimal pddl3 planning with mips-xxl, in: Proceedings of the ICAPS Booklet on the Fifth International Planning Competition, 2006.
[19] E. Fink, Q. Yang, A spectrum of plan justifications, in: Proceedings of the AAAI 1993 Spring Symposium, 1993, pp. 23–33.
[20] J. Frank, A. Jonsson, R. Morris, D. Smith, Planning and scheduling for fleets of earth observing satellites, in: Proceedings of Sixth Int. Symp. on Artificial Intelligence, Robotics, Automation & Space, 2001.
[21] B. Gazen, C. Knoblock, Combining the expressiveness of ucpop with the efficiency of graphplan, in: Fourth European Conference on Planning, 1997.
[22] A. Gerevini, B. Bonet, B. Givan, Fifth international planning competition, in: IPC06 Booklet, 2006.
[23] A. Gerevini, D. Long, Plan constraints and preferences in PDDL3: The language of the fifth international planning competition, Tech. rep., University of Brescia, Italy (August 2005).
[24] A. Gerevini, D. Long, IPC-5 website, http://zeus.ing.unibs.it/ipc-5/, 2006.
[25] A. Gerevini, A. Saetti, I. Serina, Planning through stochastic local search and temporal action graphs in lpg, Journal of Artificial Intelligence Research 20 (2003) 239–290.
[26] E. Giunchiglia, M. Maratea, Planning as satisfiability with preferences, in: Proceedings of AAAI-07, 2007.
[27] A. Globus, J. Crawford, J. Lohn, A. Pryor, Scheduling earth observing satellites with evolutionary algorithms, in: Proceedings of Int. Conf. on Space Mission Challenges for Infor. Tech., 2003.

[28] E. Hansen, R. Zhou, Anytime heuristic search, Journal of Artificial Intelligence Research 28 (2007) 267–297.
[29] M. Helmert, The fast downward planning system, Journal of Artificial Intelligence Research (2006) 191–246.
[30] J. Hoey, R. St-Aubin, A. Hu, C. Boutilier, Spudd: Stochastic planning using decision diagrams, in: Proceedings of UAI-99, 1999.
[31] J. Hoffmann, B. Nebel, The FF planning system: Fast plan generation through heuristic search, Journal of Artificial Intelligence Research 14 (2001) 253–302.
[32] C.-W. Hsu, B. Wah, R. Huang, Y. Chen, New features in sgplan for handling preferences and constraints in pddl3.0, in: Proceedings of the ICAPS Booklet on the Fifth International Planning Competition, 2006.
[33] E. Keyder, H. Geffner, Set-additive and tsp heuristics for planning with action costs and soft goals, in: Proceedings of the Workshop on Heuristics for Domain-Independent Planning, ICAPS-07, 2007.
[34] L. Kramer, L. Giuliano, Reasoning about and scheduling linked hst observations with spike, in: Proceedings of Int. Workshop on Planning and Scheduling for Space, 1997.
[35] L. Kramer, S. Smith, Maximizing flexibility: A retraction heuristic for oversubscribed scheduling problems, in: Proceedings of IJCAI-03, 2003.
[36] M. Likhachev, G. Gordon, S. Thrun, Ara*: Anytime a* with provable bounds on sub-optimality, in: Proceedings of NIPS-04, 2004.
[37] X. Nguyen, S. Kambhampati, R.S. Nigenda, Planning graph as the basis to derive heuristics for plan synthesis by state space and csp search, Artificial Intelligence 135 (1–2) (2002) 73–124.
[38] R.S. Nigenda, S. Kambhampati, Planning graph heuristics for selecting objectives in over-subscription planning problems, in: Proceedings of ICAPS-05, 2005.
[39] W. Potter, J. Gasch, A photo album of earth: Scheduling landsat 7 mission daily activities, in: Proceedings of SpaceOp, 1998.
[40] D.E. Smith, Choosing objectives in over-subscription planning, in: Proceedings of ICAPS-04, 2004.
[41] E. Stefan, Taming numbers and durations in the model checking integrated planning system, Journal of Artificial Intelligence Research 40 (2003) 195–238.
[42] S. Tran, E. Pontelli, Planning with preferences using logic programming, Theory and Practice of Logic Programming 6 (5) (2006) 559–608.
[43] M. van den Briel, S. Kambhampati, T. Vossen, Planning with preferences and trajectory constraints by integer programming, in: Proceedings of Workshop on Preferences and Soft Constraints at ICAPS-06, 2006.
[44] M. van den Briel, R.S. Nigenda, M.B. Do, S. Kambhampati, Effective approaches for partial satisfaction (over-subscription) planning, in: Proceedings of AAAI-04, 2004.
[45] V. Vidal, A lookahead strategy for heuristic search planning, in: Proceedings of ICAPS-04, 2004.
[46] M. Williamson, S. Hanks, Optimal planning with a goal-directed utility model, in: Proceedings of AIPS-94, 1994.