# *Yochan*$^{\mathcal{PS}}$: PDDL3 Simple Preferences as Partial Satisfaction Planning

**J. Benton & Subbarao Kambhampati**
Computer Sci. & Eng. Dept.
Arizona State University
Tempe, AZ 85287
{j.benton,rao}@asu.edu

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

## Introduction

*Yochan*$^{\mathcal{PS}}$ compiles a problem using PDDL3 "simple preferences" (PDDL3-SP), as defined in the 5th International Planning Competition (IPC5), into a partial satisfaction planning (PSP) (van den Briel *et al.* 2004). The commonality of the semantics between these problem types enable the conversion. In particular, both planning problem definitions include relaxations on goals and both define plan quality metrics. We take advantage of these commonalities and produce a problem solvable by PSP planners from a PDDL3-SP problem definition. A minor restriction is made of resulting PSP plans so the compilation may be simplified to avoid extraneous exponential increases in the number of actions. We chose *Sapa*$^{\mathcal{PS}}$ to solve the new problem.

## PSP Net Benefit and PDDL3-SP

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

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

In PDDL3 "simple preferences" (PDDL-SP), preferences can be defined in goal conditions $g \in G$ and action preconditions $pre(a) \mid a \in A$ (Gerevini & Long 2005). Conditions defined in this way do not need to be achieved for a plan to be valid. This relates well to goals as defined in PSP. However, unlike PSP, cost is acquired by failing to satisfy preferences. There is also no explicit utility defined. Let $\Phi$ be a preference condition, then $Cost(\Phi) = \alpha$, where $\alpha$ is a constant value.[1] Let $pref(G)$ be the set of all preference conditions on goals and $pref(a)$ be all preference preconditions on $a \in A$. For a plan $P$, if a preference precondition, $pref_p \in pref(a)$ where $a \in P$, is applied in state $S$, without satisfying $p$ then cost $Cost(pref_p)$ is incurred. In the case of a preference on a goal, $pref_g \in pref(G)$, cost $Cost(pref_g)$ is applied when the preference goal is not satisfied at the end state of a plan. In PDDL3-SP, we want to find a plan $P$ that incurs the least cost.

[1] In PDDL3, many preferences may have the same name. For PDDL3-SP, this is syntactic sugar and we therefore refer to preferences as if each is uniquely identified to simplify the discussion.

## Compiling PDDL3-SP to PSP

Both PSP and PDDL3-SP use a notion of cost on actions, though their view differs on how to define cost. PSP defines cost directly on each action, while PDDL3-SP uses a less direct approach by defining conditions for when cost is generated. In one sense, PDDL3-SP can be viewed as considering action cost as a conditional effect on an action where cost is increased on the preference condition's negation. We use this observation to inspire our action compilation to PSP. That is, we compile PDDL3 "simple preferences" on actions in a manner that is similar to how (Gazen & Knoblock 1997) compiles conditional effects.

We handle goal preferences differently. In PSP, we gain utility for achieving goals. In PDDL3-SP, we add cost for failing to achieve goals. Taken apart these concepts are complements of one another (i.e. cost for failing and utility for succeeding). The idea is that *not* failing to achieve a goal reduces our cost (i.e. gains utility for us). Therefore, as part of our compilation to PSP we transform a "simple preference" goal to an equivalent goal with utility equal the cost produced for not satisfying it in the PDDL3-SP problem. In this way we can view goal achievement as *canceling out* the cost of obtaining the goal. That is, we can compile a goal preference $pref_p$ to an action that takes $p$ as a condition. The effect of the action would be that we "have the preference" and hence we would place that effect in our goal state with a utility equal to $Cost(pref_p)$.

Figure 1 shows the algorithm for compiling a PDDL3-SP problem into a PSP problem. We begin by first creating a temporary action $a$ for every preference $pref_p$ in the goals. The action $a$ has $p$ as a precondition, and a new effect, $g_p$. $g_p$ takes the name of $pref_p$. We then add $g_p$ to the goal set $G$, and give it utility equal the cost of violating the preference. The process then removes $pref_p$ from the goal set.

After processing the goals into a set of actions and new goals, we proceed by compiling each action in the problem. For each $a \in A$ we take each set $precSet$ of the power set $P(pref(a))$. This allows us to create a version

**forall** $pref(p) \in pref(G)$ **do**
  $pre(a) := p$
  $g_p := name(pref_p)$
  $eff(a) := g_p$
  **forall** $b \in A$ **do**
    $eff(b) := eff(b) \cup \neg\{g_{pref}\}$
  **endfor**;
  $A := A \cup \{a\}$
  $U(g_{pref}) := Cost(pref_p)$
  $G := (G \cup \{g_{pref}\}) \setminus \{p\}$
**endfor**;
$i := 0$
**forall** $a \in A$ **do**
  **for each** $precSet \in P(pref(a))$ **do**
    $pre(a_i) := pre(a) \cup precSet$
    $eff(a_i) := eff(a)$
    $c_{a_i} := Cost(pref(a) \setminus precSet)$
    $A := A \cup \{a_i\}$
    $i := i + 1$
  **endfor**;
  $A := A \setminus \{a\}$
**endfor**;

Figure 1: PDDL3-SP to PSP compilation process.

of $a$ for every combination of its preferences. The cost of the action is the cost of failing to satisfy the preferences in $pref(a) \setminus precSet$. We remove $a$ from the domain after all of its compiled actions are created. Notice that because we use the power set of preferences, this results in an exponential increase in the number of actions.

When we output a plan, we must remove all new actions that produce preference goals and our metric value is calculated as follows:

$$\sum_{g \in G} U(g) \; - \sum_{g' \in G'} U(g') \; + \sum_{a \in P} c_a \qquad (2)$$

## Plan Criteria

The reader may notice that the above algorithm will generate a set of actions $A_a$ from an original action $a$ that are all applicable in states where all preferences are met. That is, actions that have cost may be inappropriately included in the plan at such states. This would mean that the PSP compilation could produce incorrect metric values in the final plan. One way to fix this issue would be to explicitly negate the preference conditions that are not included in the new action preconditions. This is similar to the approach taken in (Gazen & Knoblock 1997) for conditional effects. We decided against this for three related reasons. First, all known PSP planners require domains be specified using STRIPS actions and this technique would introduce non-STRIPS actions–specifically, actions with negative preconditions and those with disjunctive preconditions (due to the negation of conjunctive preferences). Second, compiling disjunctive preconditions to STRIPS may require an exponential number of new actions (Gazen & Knoblock 1997; Nebel 2000) and since we are already potentially adding an exponential number of actions in the compilation from preferences, we thought it best to avoid adding more. Lastly, and most importantly, we can use a simple criteria on the plan that removes the need to include the negation of preference conditions: We require 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 state. This criteria is already inherent in some PSP planners such as *Sapa*$^{\mathcal{PS}}$ (Do & Kambhampati 2004) and *OptiPlan* (van den Briel *et al.* 2004).

## Example

As an example, let us see how an action with a preference would be compiled. Consider the following PDDL3 action taken from the IPC5 TPP domain:

```
(: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)))
```

A plan metric assigns a weight to our preferences:

```
(:metric (+ (* 10 (is-violated p-drive) )
            (* 5  (is-violated P0A) )))
```

This action can be compiled into PSP style actions:

```
(:action drive-0
 :parameters
   (?t - truck ?from ?to - place)
 :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 10
 :precondition (and
   (at ?t ?from) (connected ?from ?to))
 :effect (and (not (at ?t ?from))
              (at ?t ?to)))
```

Let us also consider the following goal preference in the same domain:

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

The goal will be compiled into the following PSP action:

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

With the goal:

```
((hasPref-p0a) 5.0)
```

## 5th International Planning Competition

For the planning competition, we used the compilation described in combination with *Sapa*$^{PS}$ (Do & Kambhampati 2004) to create *Yochan*$^{PS}$. *Sapa*$^{PS}$ inherently meets the plan criteria required for our compilation. It performs an A* search, and its cost propagated relaxed planning graph heuristic ensures that, given any set of actions with the same effects, the branch with the least cost action will be taken. As another point, *Sapa*$^{PS}$ is capable of handling "hard" goals, which are prevalent in the competition domains. It has also shown to be successful in solving PSP problems (van den Briel *et al.* 2004).

## Conclusion

We outlined a method of converting domains specified in the "simple preferences" category of the Fifth International Planning Competitions (PDDL3-SP) to partial satisfaction planning (PSP) problems. The technique uses ideas for compiling action conditional effects into STRIPS actions as a basis. Though the process has the potential for adding several actions to the domain, in practice the number of added actions appears manageable.

## References


Do, M., and Kambhampati, S. 2004. Partial satisfaction (over-subscription) planning as heuristic search. In *Knowledge Based Computer Systems*.

Gazen, B., and Knoblock, C. 1997. Combining the expressiveness of ucpop with the efficiency of graphplan. In *Fourth European Conference on Planning*.

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

Nebel, B. 2000. On the compilability and expressive power of propositional planning formalisms. *Journal of Artificial Intelligence Research* (12):271–315.

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

van den Briel, M.; Sanchez, R.; Do, M. B.; and Kambhampati, S. 2004. Effective approaches for partial satisfaction (over-subscription) planning. In *Proc. of AAAI-04*.