# LPRPG-P: Relaxed Plan Heuristics for Planning with Preferences

**Amanda Coles and Andrew Coles**
Department of Computer and Information Sciences,
University of Strathclyde, Glasgow, G1 1XH, UK
email: firstname.lastname@cis.strath.ac.uk

## Abstract

In this paper we present a planner, LPRPG-P, capable of reasoning with the non-temporal subset of PDDL3 preferences. Our focus is on computation of relaxed plan based heuristics that effectively guide a planner towards good solutions satisfying preferences. We build on the planner LPRPG, a hybrid relaxed planning graph (RPG)–linear programming (LP) approach. We make extensions to the RPG to reason with propositional preferences, and to the LP to reason with numeric preferences. LPRPG-P is the first planner with direct guidance for numeric preference satisfaction, exploiting the strong numeric reasoning of the LP. We introduce an anytime search approach for use with our new heuristic, and present results showing that LPRPG-P extends the state of the art in domain-independent planning with preferences.

## 1 Introduction

Classical AI planning considers the problem of reaching a set of given goals from some specified initial state. It is often the case, however, that we are concerned not only about the final state after plan execution, but also about the structure of the plan itself. For example, we may prefer for a specific truck to pass through a certain location, or that the battery charge of a rover does not drop below a certain safety level throughout the execution of a plan. Further, we may not be certain that a plan exists to satisfy all goals and trajectory constraints we desire; traditional planners report failure in this case. We would prefer the planner to produce a plan that satisfies as many of our constraints/goals as possible weighted according to some utilities we specify. Such soft constraints are referred to as *preferences*.

In this paper we present a new heuristic, based on LPRPG (Coles *et al.* 2008), capable of reasoning with such problems and unique in two regards. First, it provides preference satisfaction guidance through the use of relaxed plans, which have proven highly successful in satisfycing planning; second it provides explicit guidance for satisfaction of preferences involving numeric conditions. We build on existing work in planning with preferences and trajectory constraints.

Work on trajectory constraints began by considering hard constraints (Kabanza & Thiébaux 2005), these can be used to prune the search space and thus find plans with desired properties. Trajectory preferences are an extension of this, where not all constraints can be satisfied, rendering simple pruning ineffective. Researchers began considering these in the Fifth International Planning Competition (IPC5) when the PDDL3 language (Gerevini *et al.* 2009) for preferences was introduced. The most closely related of IPC5 planners, HPLAN-P (Baier *et al.* 2007), performs forward search using heuristics designed for propositional preferences. These are based on the relaxed planning graph (RPG) structure and use techniques such as summing the layers in which goals/preference facts appear (rather than relaxed plans) to estimate goal distance and preference satisfaction potential.

Net-benefit (NB) (or over-subscription) planning is also related. *Yochan*$^{PS}$ (Benton *et al.* 2009) handles a limited subset of PDDL3 preferences by compiling to a cost-based representation, where preferences are implicit; and using an over-subscription planner. The problem of deciding on a set of soft goals to achieve in NB planning has been approached in many ways, one being the use of an LP (Benton *et al.* 2007). Compilation to cost-based satisfycing planning has also been successful (Keyder & Geffner 2009).

NB planning does not generally consider trajectory preferences; compilation in this case is much more complex. One compilation-based approach supporting trajectory preferences, that of MIPS-XXL (Edelkamp *et al.* 2006), is to use compilation with a modified planner (Metric-FF). Metric-FF is repeatedly called, with a goal is added each time that the metric function value improves on the last. There is, however, no direct heuristic guidance about how to satisfy preferences. MIPS-XXL is the only planner that handles the complete PDDL3 language. SGPLAN can reason with preferences in IPC5 domains; however, the mechanism of dealing with these in SGPLAN 6 often relies on recognising textual features of the domain (Hsu & Wah 2008). It is unclear how the implemented techniques generalise to wider application, we return to this in our evaluation.

### 1.1 PDDL 3 Preferences

A PDDL3 planning problem is a tuple $\langle I, G, A, P, M \rangle$ where:

- $I$ is a collection of facts and assignments to numeric variables describing the initial state;
- $G$ is a collection of facts and numeric conditions that must be true in a goal state;
- $A$ is a set of actions, where each $a \in A$ has:

  - a set $pre_a$ of preconditions (numeric or propositional) on its execution;
  - effects, facts that become true, $eff^+_a$, or false, $eff^-_a$; and numeric updates $eff^{num}_a$ that occur upon its execution.
- $P$ is a set of preferences: desired properties of the plan;
- $M$ is a metric function defined over preference violations and numeric variables in the domain.

The preferences in $P$ can be split in to two categories: simple preferences and trajectory preferences. Simple preferences take the form `(at-end f)`, denoting a soft goal; or appear in preconditions `(preference f)`, meaning we prefer `f` to be true upon executing the action. Trajectory preferences specify conditions on the trajectory of the plan: `(always f)`, `(sometime f)` and the restricted nestings, `(at-most-once f)`, `(sometime-after f g)`, `(sometime-before f g)`. In general, `f` and `g` are ADL formulæ. A preference is *numeric* if `f` or `g` refer to numeric variables. The semantics of these preferences are as would be intuitively expected, we refer the reader to (Gerevini *et al.* 2009) for LTL specifications.

$M$ assigns a cost to the violation of each preference specified in $P$; the planner aims to find a solution that gives the best possible value of this function (not necessarily optimal). For precondition preferences, this cost is incurred each time the relevant action is applied but the preference does not hold. For all other preferences, the cost is incurred once, if the plan does not satisfy the preference. In this paper we focus on preference violations, hence we restrict the metric function to these: we do not allow inclusion of domain variables. There are no direct associated costs with applying actions. As we are building upon LPRPG we inherit its restriction to producer-consumer behaviour (Coles *et al.* 2008). That is, all actions increasing (decreasing) numeric variable $v$ must have a precondition $v \leq c$ ($v \geq c$) and effect $v$+=$k$ ($v$-=$k$) where $c$ and $k$ are positive real-valued constants[1].

## 2 LPRPG Heuristic

Here we give an overview of the LPRPG heuristic (Coles *et al.* 2008) upon which we base our work, and discuss the challenges in extending it to reason about preferences. The LPRPG heuristic was designed to reason about problems with strong numeric interaction: it is a hybrid heuristic, using FF's RPG (Hoffmann 2003) for propositional reasoning, tightly coupled with a Linear Program (LP) for numeric reasoning.

RPG heuristic computation begins by building a graph consisting of fact and action layers. The initial fact layer, $f_0$ contains all facts true in $S$, the state to be evaluated, along with upper and lower bounds on the values of each numeric variables $v$, $ub(v)$ and $lb(v)$. The action layer $a_l$, following each fact layer $f_l$, contains all actions whose preconditions are satisfied in $f_l$. Fact layer $f_l$ contains the union of the positive effects ($eff^+$) of all actions in $a_{l-1}$ and all facts in $f_{l-1}$ along with new bounds for each numeric variable $v$ in that layer. Extraction of a relaxed plan is done by regression from the goals starting at the first fact layer in which all goals appear, $f_g$. An action is selected to achieve each goal in turn, and the goal is replaced with the preconditions of that action, until all goals are present in $f_0$.

[1]Increasing actions often have $c = \infty$

The LP is used for two distinct purposes in heuristic computation: in RPG expansion, to compute $ub(v)$ and $lb(v)$ for numeric variables at each layer; and in relaxed plan extraction, to select actions to achieve numeric goals. LPs are built according to the behaviour each numeric variable $v$ in the problem. We define $\delta(v_i, a_i)$ as the (constant) numeric effect of action $a_i$ on variable $v_i$. For each action, $a_i$ we have a variable $a^i$ representing the number of times that action has been applied. The LP contains a constraint, relating the initial value $v$ of each numeric variable to its final value $v'$:

$$v' = v + \sum_{a \in A} a^i . \delta(v, a_i)$$

Further constraints added to the LP record the highest and lowest possible values each $v$ can hold. For lower bounds, we find the action $a_s$ consuming $v$ with the smallest $c - k$, such that $v \geq c \in pre_{a_s}$ and $v$-=$k \in \text{eff}_{a_s}$. The value of $c - k$ is then taken as the lowest value that $v$ can hold ($v_{min}$); similar computation can be performed for upper bounds ($v_{max}$). These become additional constraints in the LP:

$$v' \geq v_{min}, v' \leq v_{max}$$

We can build an LP for fact layer $f_l$, LP$_{f_1}$, by including only actions that appear in $a_{l-1}$. This LP relaxes action ordering, rather than negative effects, and thus better captures numeric interaction. We therefore make use of it to compute $ub(v)$ and $lb(v)$ at each fact layer. For each $v$ we call LP$_{f_l}$ twice, with the objective function maximise/minimise $v'$. The values of $v'$ in LP$_{f_l}$ become bounds on $v$ in $f_l$.

In LPRPG-P solution extraction is performed as in Metric-FF for propositional goals. However, achievers for numeric goals are found using the LP as it has a richer numeric model. Each time the RPG requires a numeric goal $g_n$ to be satisfied an LP is created for the first $f_l$ such that the bounds on variables in $f_l$ allow satisfaction of $g_n$, and the goal formula $g_n$ is added as a constraint. The LP objective function is set to minimise the sum of action variables. The actions whose variables have non-zero values in the resulting solution are added to the relaxed plan, and their preconditions are added to the set of goals to be achieved in the RPG.

### 2.1 Issues with Optimisation

At first glance it may appear relatively straightforward to use the LP for optimisation: change the objective function in solution extraction to include the cost of preference violations.

To explore this, consider a problem with numeric variables $v$, $w$ and proposition $f$. Initially $v$=5, $w$=0 and $f$ is false; the goal is $w$=1. Available actions are $a$: pre$_a$:$\{v \geq 1\}$ eff$_a$:$\{v\text{-=}1 \wedge w\text{+=}1\}$; $b$: pre$_b$:$\{f\}$ eff$_b$:$\{v\text{+=}1\}$; and $c$: pre$_c$:$\emptyset$, eff$_c$:$\{f\}$. We have a preference $p$ `(always (v ≥ 5))`.

If an RPG is built from the initial state, the goal appears in $f_1$, following the application of $a$ in $a_0$. Note $c$ also appears in $a_0$ but $b$ will not (as $f \notin f_0$). As all goals appear in $f_1$, LPRPG will ask LP$_{f_1}$ to achieve the goal $w$=1. As $b$ does not appear in $a_0$ it does not appear in LP$_{f_1}$ so the solution returned is $a$=1, regardless of whether the objective function is modified. The solution $[c, b, a]$ satisfying $p$ simply does not appear in LP$_{f_1}$. Supposing $w$ was replaced with a proposition $g$, and $a$ added $g$, in this case the goals still appear in $f_0$ and the LP would not be called as $g$ is propositional, so $a$ would simply be selected to achieve $g$ in the RPG.

This example demonstrates that it is necessary to do much more than simply modify the LP objective function. We must consider potential preference violations in RPG building and in both the LP and RPG in solution extraction.

## 3 Automata: General Preference Reasoning

We now describe how automata can be used to allow generalised reasoning about all preferences, regardless of which modal-operator they are based on. We extend existing techniques and further develop the ideas for relaxed planning. Hereafter we assume conversion of ADL conditions to NNF, these are handled in an RPG as in (Coles & Smith 2007).

### 3.1 Monitoring Preference Satisfaction

In order to reason with preferences in the heuristic we must be able to determine when a preference is violated by the application of a given action. We also need to detect when preferences have been violated in reaching a state, as we need not penalise violation further. To do this we use automata—this allows the same reasoning to be used for all preference types, we need no longer consider them individually. Automata can be generated using the methods of MIPS-XXL and HPLAN-P. We use their techniques to maintain the position of each preference automaton at every state during search, changing an automaton's position if the condition on a transition out of its current position becomes satisfied.

In our automata we additionally require that each position is labelled either: Sat, the preference is satisfied in the current state, $S$; Unsat, the preference is unsatisfied in $S$; E-Sat, the preference is satisfied in $S$ and can never become unsatisfied; or E-Vio, the preference can never be satisfied. These labels can be generated automatically, Sat positions are accepting positions, all others are Unsat positions; then E-Vio/E-Sat positions are the Unsat/Sat positions from which no Sat/Unsat position is reachable respectively. If no E-Vio position is present in the generated automaton we add one: HPLAN-P and MIPS-XXL consider e-violation of preferences but do not explicitly include them in automata.

In order to determine when preferences can no longer be achieved, we add transitions between Unsat positions and E-Vio by incorporating knowledge from the RPG. For a preference to become satisfied from a given Unsat position, $U$, there must be a path from $U$ to a Sat position upon which all transitions can be activated in order. This problem cannot be solved easily in general (it is plan existence); we can, however, make use of the RPG to identify trivial cases where no plan exists: if no path from $U$ to a Sat position has all its transition conditions appear in the RPG the preference cannot be satisfied in any subsequent state. We create a transition from $U$ to E-Vio with a special label denoting that the disjunction of these conditions does not appear in the RPG.

Formally, for an Unsat position with a number of paths $p_1...p_n$, each reaching a Sat position, and each labelled with conditions $ci_1...ci_{m^i}$, the preference cannot be satisfied if the following holds at $f_{fix}$, the RPG fix point:

$$\neg(f_{fix} \Rightarrow \bigvee_{i\in[1..n]} \bigwedge_{j\in[1..m^i]} ci_j)$$

The appearance of positive literals in an RPG layer is defined in Section 2. We say negative literals appear in $f_0$ if they are not in $S$. Otherwise $\neg f$ appears in $f_l$, where $a_{l-1}$ is the first layer in which an action deleting $f$ appears. We say $f_l \Rightarrow c$ if $c$ can be made true using only facts (or their negations) that appear in $f_l$ (variable bounds for numeric formulæ). Note: if both $f$ and $\neg f$ appear in $f_l$ then both can be true simultaneously in order to satisfy $c$. This is simply equivalent to saying that if an action had a required a precondition $c$ to be applied in $a_l$ then $c$ could be satisfied in $f_l$, thus the action could be applied in $a_l$.

This general reasoning may appear complicated, but consider the simple automata for PDDL3 preferences in Figure 1. For the `sometime-after` preference we can see that there is only one way to reach a Sat position from Unsat: achieve $b$. We therefore know if $b$ does not appear in the RPG the preference can never be satisfied, so an edge is added from Unsat to E-Vio labelled $b$ not in RPG.

We can now say an action violates a preference when applied in the state $S$ if its application results in a state in which the relevant automaton is in E-Vio (note automaton position may change through a series of positions at once if conditions on edges from subsequent new positions are satisfied). We define the preference violation cost of a state $S$, $PVC(S)$, as the sum of the costs of all preferences whose automata are in E-Vio in $S$, and the cost of precondition preferences violated in reaching $S$. This is a lower bound on the cost of any solution reachable from $S$.

### 3.2 Preference Violations in Relaxed Planning

The position of an automaton in a planning state is well defined; however, in an RPG layer, its position is not so obvious. We have a choice whether each action has been applied or not: the RPG layer represents a *set* of states. In the PDDL3 automata there is a hierarchy of more preferable positions[2]: E-Sat, Sat, Unsat, E-Vio; if there are multiple Sat positions the further a Sat position is from E-Vio the more preferable it is. For PDDL3 preferences we therefore maintain a single optimistic automaton position for each RPG layer. The automaton moves to the most optimistic position reachable from its current position according to the facts true in $f_l$.

We can make use of the optimistic automaton position $p$ for preference $p_i$ in $f_l$ to define whether an action $a_i$ violates $p_i$ in $f_l$. Generally we know $a_i$ will violate $p_i$ in $f_l$, if the effects of $a_i$ would certainly satisfy condition(s) causing the automaton for $p_i$ to move from $p$ to E-Vio. Definition 3.1 specifies a subset of such actions: *certain triggers*.

**Definition 3.1 — Certain Trigger**
Action $A$ is a certain trigger for condition $C = c \wedge c'$ (where $c$ involves no conjunction) at $f_l$ if $\neg(f_l \Rightarrow \neg c')$ — i.e. $c'$ cannot be false given the facts in $f_l$ — and either:

- $A$ adds (deletes) a fact $f$ such that $f \Rightarrow c$ ($\neg f \Rightarrow c$);
- $A$ has effect $v$+=$k$, $v_{min} + k \geq d$ and $(v \geq d) \Rightarrow c$;
- $A$ has effect $v$-=$k$, $v_{max} - k \leq d$ and $(v \leq d) \Rightarrow c$.

In the propositional case we consider whether each single fact added or deleted by $A$ will (when combined with facts

[2]We note that our reasoning can be extended to a more general subset of LTL by maintaining every possible position each automaton could be in, we omit the full details of this due to lack of space.

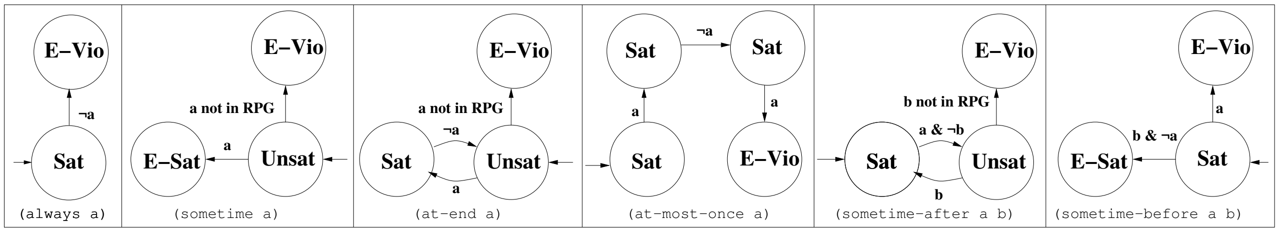


Figure 1: Automata for PDDL 3 Preferences (Self Loops Omitted for Clarity)

in $f_l$) satisfy $C$. Note, a condition $a \vee b$ can be written $(a \vee b) \wedge true$ so in this case an action $A$ adding $a$ will be a certain trigger for $a \vee b$. For $a \wedge b$, $a$ becomes $c$ and $b$ is $c'$ (or vice-versa), so until $\neg b$ is in the RPG, $A$ is a certain trigger. For actions with numeric effects the definition is more complex. Action $A$ decreasing $v$ by 1 may, or may not, cause $v \leq 3$, depending on the current value of $v$. However, we can make use of $v_{max}$ and $v_{min}$ (Section 2) to find when an action will always necessarily satisfy a numeric condition regardless of the value of $v$. If the lowest possible value of $v$, $v_{min}$, plus the effect of $A$, is large enough to satisfy $c$ we know application of $A$ will always cause satisfaction of $c$.

**Definition 3.2 — Preference Violation in RPG layer $a_l$**
The action $A$ violates preference $p_i$ in $a_l$, with optimistic automaton position $p$ for $p_i$ if either:

- $A$ is a certain trigger for the conditions on all edges on the path from $p$ to E-Vio (except the special RPG label, which must be satisfied in $f_l$);
- $p$ is a Sat position; $A$ is a certain trigger for all the conditions on an path from $p$ to an Unsat position $U$; and no transition from $U$ back to a Sat position can be activated using facts in $f_l$.

The first part of Definition 3.2 brings together Definition 3.1 and the optimistic automaton position $p$. As an illustrative example, an action $A$ violates the `(sometime-before a b)` preference in layer $a_l$ if b is not in $f_l$ (so the optimistic automaton position is Sat) and $A$ adds a. We extend the violation definition further, giving the second part of Definition 3.2. To understand this consider the (optimistically Sat) preference $p_1$ `(sometime-after a b)` where $b$ is not in $f_l$, and action $A$ in $a_l$ adds the fact $a$ to RPG layer $f_{l+1}$. Here, $A$ is a certain trigger for the condition, $a$, on the transition from Sat to Unsat for $p_1$. The transition from Unsat to Sat requires $b$, which is not in $f_l$. We can see therefore see that $A$ violates $p_1$ in $a_l$. At future layers, once $b$ appears, $A$ will no longer violate $p_1$.

**Generalisation** So far our reasoning has been described for preferences where a single modal operator is applied to a formula that does not contain further modal operators (e.g. `(sometime-after (or a b) (forall x p(x)))`). However, for preferences with formulæ over multiple modal operators (e.g. `(or (always p) (sometime q))`) the reasoning must be slightly extended. Satisfaction of these can be determined by maintaining automata for each sub-clause. Actions' preference violations can be computed: in the `or` case, an action violates the preference if it violates all of the subclauses; whereas for `and` it need violate only one sub-clause.

## 4 Heuristic Computation

We now describe how an RPG can be built using these automata to record sets of preferences violated in achieving each fact, and applying each action, at each layer. We then show how relaxed plans satisfying preferences are extracted.

### 4.1 Planning Graph Construction

We introduce the notion of a preference violation set for facts (and actions) in an RPG. Informally these are preferences that must be violated in order to reach a fact (apply an action) from the state to be evaluated, $S$. At each fact layer, $f_l$, in the planning graph, we record the optimistic position of the automaton for each preference $p$, $f_l[p].P$; and the actions that would violate $p$ if applied in $a_l$, $f_l[p].X$, according to Definition 3.2. From these we define *preference violation sets* for facts and actions. Informally, a fact (or action) preference violation set contains the preferences that are violated in order to reach (apply) it at a given layer. We define the cost of a preference violation set as:

$Cost(\{p_1...p_n\}) = \Sigma_{i=1..n}\, violation\_cost(p_i)$.

...where $violation\_cost(p_i)$ is defined in $M$ (in the planning problem). The violation set for an action contains the preferences that are violated by applying it at layer $l$ and the preference violations needed to meet its preconditions.

**Definition 4.1 — Action Preference Violation Set**
The preference violation set $V_a^l$ for action $a$ in $a_l$, based on the facts in $f_l$, is: $\{p \in P \mid a \in f_l[p].X\} \cup (\cup_{f \in pre_a} V_f^l)$

All facts in $f_0$ have empty violation sets: they have already been achieved, so any costs have already been paid. The preference violation set for a fact $f$ in $f_l, l \geq 1$, is the lowest cost of either its violation set in $f_{l-1}$ or that of any achiever of $f$ in the preceding action layer $a_{l-1}$.

**Definition 4.2 — Fact Preference Violation Set**
The preference violation set $V_f^l$ for a fact $f$ in $f_l$ is $\emptyset$ if $l = 0$. Otherwise, it is the lowest cost set of: $V_f^{l-1}$, if $f$ is in fact layer $l-1$; or $V_a^{l-1}$ for any action $a$ s.t. $f \in \mathit{eff}_a^+$.

Here we commit to choosing the lowest cost preference violation set. The alternative, keeping several and selecting from these, would raise the cost of heuristic calculation.

Finally, we define the preference violation cost at $f_l$ as:
$\text{LVC}(f_l) = Cost(\{p \in P \mid f_l[p].P \in \{\text{E-Vio}, \text{Unsat}\}\})$

We can now consider RPG construction, shown in Algorithm 1. The main loop (line 4 to 34) begins as in FF, determining the next action layer $a_l$ and fact layer $f_{l+1}$. Then, at line 10, it calculates the preference violation set for each fact in $f_{l+1}$ based on the actions in $a_l$. Lines 11 to 30 update the optimistic position of each automaton, following the approach discussed in Section 3.2. If this led to a change in an action's preference violation set (due to its application violating fewer preferences), lines 31 to 34 add additional layers, to propagate the effects of any violation set changes.

As a worked example, consider a preference $p$ = `(sometime-before c d)` that is Sat in $f_0$. In this case, $f_0[p].X$ contains any action adding `c` (it is preferable that `d` is added first). Expanding the planning graph, when (if) a layer $f_l$ is reached containing `d`, four key changes in the behaviour of the algorithm occur, in succession. First as the label `d` on the Sat-to-E-Sat edge is satisfied, $f_l[p].P$=E-Sat (line 16). Second, lines 18 to 26 will assign $f_l[p].X = \emptyset$, as there are no edges out of the new optimistic position, E-Sat. Third, at lines 27 to 30, any actions that would previously violate $p$ (by adding `c`) are given the additional precondition `d`. Fourth, and finally, if the revised $f_l[p].X$ reduced the cost of the violation set of some action $a_i$, then following Definition 4.2 $a_i$ may change the preference violation set of one or more facts. Lines 31 to 34 propagate any consequences of this.

In LPRPG graph building terminates at the first fact layer in which all goals appear. Preferences create two issues with this: first, if all goals appear, it may be beneficial to expand the planning graph further to allow more preferences to be satisfied; second, a layer satisfying all preferences may not exist. We therefore terminate graph expansion in one of two cases: either when all goals have appeared and the lower bound on preference violations $PVC(S)$ has been attained (line 4); or if no new facts are appearing (line 35) (hence no further preferences would become satisfied).

One final group of PDDL3 preferences that does not fit the automaton model is precondition preferences: these are unique because the cost is paid multiple times, once per violation. We handle these by making multiple copies of the action in the RPG: one with the preference as a hard precondition; and one without the additional precondition, but with the preference in its violation set. For actions with multiple precondition preferences we generate a copy of the action with each possible combination of preconditions. Whilst this compilation is potentially exponential, actions with multiple preference preconditions are rare in practice.

**Algorithm 1**: RPG Expansion with Preferences

```
Data: $S$, a state to evaluate
1  $l \leftarrow 0$; $f_l \leftarrow$ facts and automaton positions from $S$;
2  for each $\{p \in P \mid f_l[p].P \notin \{$E-Vio, E-Sat$\}\}$ do
3      define $f_0[p].X$ (see lines 18 to 26);
4  while $f_l$ does not meet goals $\vee LVC(f_l) > PVC(S)$ do
5      $k \leftarrow l$;
6      $A' \leftarrow \{a \in A \mid f_l$ satisfies $pre_a\}$; $a_l \leftarrow A'$;
7      $f_{l+1} \leftarrow f_l \cup (\cup_{a \in A'} \mathit{eff}^+_a)$;
8      call LP to update numeric bounds in $f_{l+1}$;
9      $l \leftarrow l + 1$;
10     for $f \in f_l$ do determine $V^l_f$ (Definition 4.2);
11     for each $\{p \in P \mid f_{l-1}[p].P \notin \{$E-Vio, E-Sat$\}\}$ do
12         $AV, AE \leftarrow$ vertices, edges from $p$'s automaton;
13         $E \leftarrow \{e \in AE \mid f_l$ satisfies label on $e\}$;
14         $R \leftarrow$ vertices in the digraph $\langle AV, E\rangle$ reachable
           from $f_{l-1}[p].P$;
15         if ($\exists r \in R$ more optimistic than $f_{l-1}[p].P$) then
16             $f_l[p].P \leftarrow$ (most optimistic) $r$;
17         else $f_l[p].P \leftarrow f_{l-1}[p].P$;
18         $f_l[p].X \leftarrow \emptyset$;
19         for each edge $(f_l[p].P, v) \in AE$ do
20             $R' \leftarrow$ vertices in $\langle AV, E\rangle$ reachable from $v$;
21             if ($R' \cap \{$E-Sat, Sat$\}) = \emptyset$ then
22                 $c \leftarrow$ label on $(f_l[p].P, v)$ in $AE$;
23                 $trig \leftarrow$ actions triggering $c$;
24                 $c' \leftarrow$ label on $(v, f_l[p].P)$ in $AE$;
25                 if $f_l$ satisfies $c'$ then each of $trig$ has
                   additional precondition(s) $c'$ from layer
                   $a_l$ onwards;
26                 else add $trig$ to $f_l[p].X$;
27         if $f_l[p].P \neq f_{l-1}[p].P$ then
28             $c \leftarrow$ label on $(f_{l-1}[p].P, f_l[p].P)$ in $AE$;
29             for $a \in (f_{l-1}[p].X \setminus f_l[p].X)$ do
30                 $a$ has additional precondition(s) $c$ from
                   layer $a_l$ onwards;
31     while $\exists a \in A' \mid Cost(V^l_a) < Cost(V^{l-1}_a)$ do
32         $a_l \leftarrow A', l \leftarrow l + 1$;
33         $f_l \leftarrow f_{l-1}$ (including preference data);
34         for $f \in f_l$ do determine $V^l_f$ (Definition 4.1);
35     if $f_k = f_l$ then break out of the while loop;
36 if $f_l$ satisfies the problem goals $G$ then return An
   RPG, layers $f_0..f_l$ else return Failure
```

## 4.2 Relaxed Plan Extraction

Relaxed plan extraction must be modified to encourage the generation of plans that satisfy preferences. First the goal set must be augmented to include soft goals: these might be explicit `(at-end)` preferences, or arise from other preferences that need additional facts for satisfaction (e.g. as-of-yet unseen `(sometime)` preference facts, or `(sometime-after a b)`, where `a` has been seen but `b` has not).

Generally, if a preference automaton is in an Unsat position in the state to be evaluated, $S$, the condition on the transition from Unsat to Sat in that automaton is added as a goal. Such goals necessarily appear in the RPG: if they did not the automaton position would be E-Vio. This generates a new goal set containing all facts needed to satisfy the preferences that can be (relaxedly) satisfied from $S$, plus all hard goals; we extract a relaxed plan to achieve these.

Extraction is performed as in the LPRPG heuristic with a few minor modifications. First, goals (and preconditions of selected achievers) are added to be achieved at the earliest layer they appear with their lowest cost violation set (rather than the earliest layer they appear). Then, when choosing actions to achieve fact $f$ in layer $f_l$, we select the action with the lowest cost violation set in a layer earlier than $f_l$. As in LPRPG actions achieving numeric conditions are chosen by the LP; modifications to this are considered in Section 4.3.

The inclusion of extra preconditions in RPG building has an important effect during solution extraction: recall that extra preconditions are added when an action appears with a lower cost violation set. Therefore, when achievers with the lowest cost violation set are selected, these will be the ones with extra preconditions (if such actions exist) and the extra preconditions are precisely the extra facts needed to satisfy the preference. Note that the consequences of using this mechanism for `(sometime-after c d)` is that `d` is actually achieved *before* `c` in the relaxed plan, not after; however, the important thing is that achievers for `d` are inserted in to the relaxed plan, not their relative order.

## 4.3 Using the LP to Satisfy Numeric Preferences

The LP is used in solution extraction to determine the achievers for numeric goals, we therefore modify it to select achievers that satisfy preferences. Key to this is the inclusion of a binary variable $p^i$ for each preference $p_i$ (forming a mixed integer-linear program), with constraints such that $p^i$=0 *iff* the values assigned to the action variables could conceivably allow the preference to be (or remain) satisfied. The LP is solved with objective function: minimise $\Sigma p^i * violation\_cost(p_i)$. We call the value of the objective function upon solving LP$_{f_l}$ VLP$(f_l)$. We then add a constraint $\Sigma p^i * violation\_cost(p_i)$ = VLP$(f_l)$ and solve again minimising $\Sigma a^i$ (LPRPG's objective). As in LPRPG if $a^i$ is non-zero in the solution to this LP $a_i$ is added to the relaxed plan.

The LP has no internal notion of action ordering or layers, so we must use the preference violation information from the RPG. Suppose we are building LP$_{fl}$ to find achievers for a numeric goal at RPG layer $f_l$. We know the set of actions that will definitely violate each preference $p_i$ if applied before $a_l$ is $f_l[p_i].X$ (Section 4.1). For each action $a_i$ in this set we add LP constraints to enforce that $(a^i > 0) \Rightarrow (p^i = 1)$.

Of the remaining actions not in $f_l[p_i].X$ some will simply never violate any preferences. Others, however, will have extra preconditions in the RPG that are required in order for their application not to violate preferences. The mechanisms in the RPG will, of course, deal with satisfaction of these preconditions: when the LP selects an achiever (returns a solution in which $a^i > 0$) the RPG puts this achiever at the layer in which it has the lowest cost violation set, thus requiring any extra preconditions. Since the LP does not deal with propositional preconditions this mechanism is invaluable. However, if the extra precondition that is going to be added is numeric, the LP is inevitably later going to be asked to achieve this. It is useful, therefore, to allow the LP to see the *numeric* consequences (i.e. any added numeric preconditions) of actions it selects for application.

The numeric preconditions of actions are captured in the LP constraints (Section 2). Note that these are the original preconditions of the actions in the planning problem and do not include those added in the RPG. The additional preconditions are therefore encoded by separate constraints: for each action $a_i$ that has had a wholly numeric precondition $c$ added at layer $l$ (removing $p_i$ from its preference violation set) we add to the LP a constraint enforcing that if $a^i > 0$, $c$ *could be satisfied* by assignments to action variables, or $p^i = 1$.

To define precisely what we mean by *could be satisfied* we make use of optimistic upper (OUB$(v)$) and lower (OLB$(v)$) bounds on LP variable $v$. OUB$(v)$, is the maximum value $v$ could hold given any possible ordering of the actions (recall the LP relaxes action ordering). Formally, OUB$(v) = v + \sum a^i.\delta(v, \mathrm{a}_i)$, $\forall \mathrm{a}_i \in A$ such that $\delta(v, \mathrm{a}_i) > 0$ (OLB$(v)$ is computed by considering $\mathrm{a}_i \in A$ such that $\delta(v, \mathrm{a}_i) < 0$). Now we say a condition $v \geq c$ *could be satisfied* if OUB$(v) \geq c$. Similarly $v \leq c$ requires OLB$(v) \leq c$. This reasoning extends to arbitrary linear numeric formulæ across several variables by taking the upper or lower bound for each as appropriate to minimise (or maximise) the value of the formula.

To illustrate this consider `(sometime-before (`$v_1 \geq 1$`) (`$v_2 \geq 1$`))`, $p_1$. Suppose that the RPG bounds on $v_1$ and $v_2$ in $f_l$ are [0,2] and that action $a_i$ always causes $v_1 \geq 1$ (and has no conditions of $v_2$). In this situation $a_i$ is not in $f_l[p_1].X$ but has the additional precondition $(v_2 \geq 1)$ in $f_l$. LP$_{f_l}$ is therefore constrained such that if $a_i$ is applied $(a^i > 0)$, OUB$(v_2) \geq 1$ or $p^1$ = 1. This forces the LP to trade off the cost of the obligation to achieve $(v_2 \geq 1)$ in order to apply $a_i$ against that of violating $p_i$. If $v_1 \geq 1$ was instead a proposition `f` this constraint would still be enforced (because the additional precondition, $v_2 \geq 1$, is numeric). If, however, we were considering `(sometime-before (`$v_1 \geq 1$`) (or (`$v_2 \geq 1$`) f))` the constraint would not be added to the LP as the condition is not wholly numeric (the LP does not consider achieving propositions): the RPG must decide whether to satisfy `f` or to later ask the LP for $(v_2 \geq 1)$.

As well as single-action triggers, we consider the consequences of the final value $v'$ of each variable on preference satisfaction: we could apply multiple actions which individually do not violate a preference, but in combination do. For example, an action with effect $v$+=1 does not necessarily violate `(always (`$v \leq 5$`))` (or `(at-most-once (`$v \leq 5$`))` where we have already seen $(v \leq 5)$), but could if applied many times. In Definition 3.1 we showed what it means for an action to be a certain trigger for a condition. Here in the LP we want to say what it means for the collection of actions selected to be a certain trigger. It makes more sense here to speak of their results, i.e. the values of $v'_i$ in the solution to the LP, being a certain trigger for conditions.

The values of $v'_i$ in LP$_{f_l}$ are a certain trigger for condition $C = c \wedge c'$ if $f(v'_0..v'_n) \geq (\leq)k$ and $f(v'_0..v'_n) \geq (\leq)k \Rightarrow c$; and, as before, $\neg(f_l \Rightarrow \neg c')$. Now we can add constraints to LP$_{f_l}$: if the values of $v'$ are a certain trigger for edge conditions on a path to E-Vio from the optimistic automaton position $p$ (of the automaton for preference $p_i$ in $f_l$), then $p_i$ is violated (again excepting any special RPG condition which must be true in $f_l$). That is, $(f(v'_0..v'_n) < (>)k) \vee (p^i = 1)$.

We can now use the LP to satisfy the hard (numeric) goals requested and attempt to satisfy the preferences. Each hard goal is added to the LP as a constraint over the variables $v'$. Then, we add constraints for preferences that, according to Section 4.2, require additional goals. For the special case of `at-end` preference $p_i$, we constrain the LP such that either the $v'$ values satisfy $p_i$, or $p^i$=1. All other goals (which are required only to hold at some point) become conditions on optimistic bounds: for instance, if $p_1$, `(sometime (`$v \geq 5$`)`, has not yet been satisfied, either OUB$(v) \geq 5$ or $p^1$=1.

We can make additional use of the modified LP during graph building. Its use to compute bounds is exactly the same as that in LPRPG but VLP$(f_l)$ is used in the termination condition. VLP$(f_l)$ (constrained to achieve all hard numeric goals) is an admissible estimate of the minimum preference violation cost attainable given the actions in RPG layer $a_{l-1}$. Thus we modify the definition of LVC$(f_l)$ to be the maximum of its previously defined value and VLP$(f_l)$. This ensures that graph expansion continues if there is still potential for LP violation cost to be reduced by adding further layers.

## 5 Search

Search to find low cost plans involves trade-off: exploring not only states with small relaxed plan length, $h(S)$, to find *feasible solutions* (satisfying hard goals); but also those with low violation cost. Anytime search approaches can produce solutions of increasing quality, reporting solutions when found and continuing search.

The LPRPG-P heuristic is non-zero for most feasible solutions as the relaxed plan will still contain actions to achieve unsatisfied preferences. Indeed $h(S) = 0$ only when $S$ contains the hard goals and all preferences are either satisfied or E-Vio in $S$. We can therefore use the LPRPG-P heuristic to continue search when feasible solutions are found, creating anytime variants of FF's (Hoffmann 2003) search strategies enforced hill-climbing (EHC) and best-first search (BFS).

Pruning can be performed once the first feasible solution $P$, with metric cost $c$, has been found. We need no longer need to consider states, $S_i$ with $PVC(S_i) \geq c$. Further, pruning can be performed using LP$f_g(S)$ where $f_g(S)$ was the latest planning graph layer built when evaluating state $S$. When constraining LP$f_g(S)$ to achieve all numeric hard goals (as discussed in Section 4.3), we can prune any state for which VLP$(f_g(S))$ + $PVC(S) \geq c$.

Note that BFS with this pruning is complete: given sufficient resources it finds provably optimal solutions. In practice this happens rarely: it requires search space exhaustion.

## 6 Results

We use all non-temporal IPC 5 benchmark domains handled by LPRPG-P. Only 2 IPC benchmark domains have numeric preferences, but both use temporal PDDL; we created non-temporal variants by compressing the durative actions into instantaneous actions and removing temporal preferences (we also fixed some bugs). We also introduce a new domain with numeric preferences based on a problem that requires that the voltage in a substation be controlled under changing customer demand (Bell *et al.* 2009). Our new domain has hard goals requiring the voltage to remain within the 1.05, 0.95 range; and preferences that it remain within [0.97,1.03] and to minimise transformer/MSC switching actions. Our modified problems are available at `http://personal.cis.strath.ac.uk/~amanda/preferences`.

We compare to the most successful planners in IPC 5 as well as the standard RPG heuristic with the anytime search strategy (HFF-A) and LPRPG-P stopping at the first solution found (i.e. standard EHC using LPRPG-P). Further we include a baseline control: producing the plan computed by FF to satisfy the hard goals (ignoring preferences) where these are present (marked * in Table 1); and the empty plan otherwise. SGPLAN 6 makes use of textual domain features

| Domain | Control | SG PLAN 5.1 | c | LPRPG -P-A (NO-LP) | c | HFF -A | c | LPRPG -P-1st -Sol. | c | SG PLAN -W | c | MIPS -XXL | | H PLAN -P | c |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Pathways SP | 20.46 | ***29.79*** | ***30*** | **25.85** | **24** | 22.73 | 4 | 20.46 | 0 | 0 | 0 | 22.53 | 6 | 22.38 | 5 |
| Trucks SP | 1.01* | ***17.02*** | 18 | **6.92** | **19** | 3.05 | 11 | 1.18 | 16 | 3 | 3 | 4 | 4 | 3 | 4 |
| Storage SP | 5.36 | ***13.37*** | ***15*** | **10.81** | **12** | 8.67 | **12** | 5.36 | 0 | 0 | 0 | 7.51 | 5 | 9.19 | 10 |
| TPP SP | 15* | ***20*** | ***20*** | 16.87 | 11 | 15.78 | **20** | 15 | 0 | **18.66** | 16 | 15.79 | 14 | 15.82 | 6 |
| SP Total | 41.83 | ***80.18*** | ***83*** | **60.45** | **66** | 50.23 | 47 | 42 | 16 | 21.66 | 19 | 49.83 | 29 | 50.39 | 25 |
| Rovers QP | 8.03* | 14.88 | ***20*** | **19.11** | **20** | 11.51 | 16 | 16.75 | **20** | 1.38 | 0 | 0 | 0 | 3.17 | 5 |
| Storage QP | 4.33 | ***14.57*** | ***20*** | **11.5** | **16** | 9.3 | 16 | 4.33 | 0 | 4.33 | 1 | 7.3 | 5 | 7.98 | 11 |
| TPP QP | 3.84* | ***18.7*** | ***20*** | **15.91** | **20** | 6.62 | 20 | 3.84 | 0 | 0 | 0 | 4.54 | 4 | 6.27 | 4 |
| Trucks QP | 1.16 | ***16.89*** | ***20*** | **9.68** | **20** | 2.2 | 13 | 3.5 | 19 | 0 | 0 | 2.12 | 3 | 2 | 5 |
| Prop. Total | 59.19 | ***145.22*** | ***163*** | **116.65** | **142** | 79.86 | 112 | 70.42 | 55 | 27.37 | 20 | 63.79 | 41 | 69.81 | 50 |
| TPP CNP | 0.61 | 1.62 | 9 | **15.51** (1.26) | **19** (15) | 14.06 | **19** | 5.37 | **19** | 0 | 0 | 0 | 1 | - | - |
| Pathways CNP | 3.09 | 0 | 0 | 27.75 **(28.38)** | **30** **(30)** | 3.09 | 1 | 3.09 | 1 | 0 | 0 | 4.11 | 4 | - | - |
| Voltage CNP | 4.13* | 0.46 | 0 | **16.56** (8.04) | **18** (10) | 1.9 | 2 | 15.21 | 18 | 0 | 0 | 3.91 | 6 | - | - |
| Total | 67.02 | 147.3 | 172 | **176.47** (154.33) | **209** (197) | 98.91 | 134 | 94.09 | 93 | 27.37 | 20 | 71.81 | 52 | - | - |

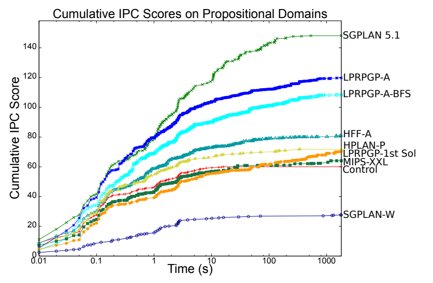
Table 1: IPC quality score, and coverage (c): Number of Problems the Planner Found a Solution Better than the Control.

to recognise some planning problems. We want to explore how planners perform on general unknown problems, so we include a second SGPLAN configuration SGPLAN-W: this consists of a script that changes the domain name and adds the never-applicable action $z$ (below), then runs SGPLAN. This prevents the text matching code in SGPLAN *6* from identifying any domains. We experimented three versions of SGPLAN (5.1, 5.2.1 and 6) and selected the best, 5.1; this is also used inside the SGPLAN-W wrapper.

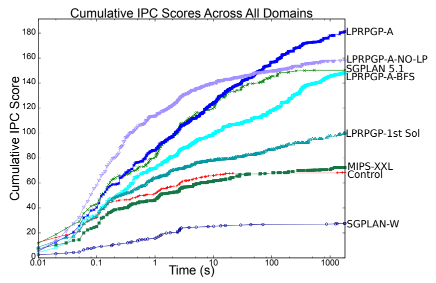
```
(:action z   :parameters (?l - dummytype)
             :precondition (and (neveradded))
             :effect (neveradded))
```

Table 1 shows the IPC scores (computed as in IPC6) for each planner after 30 minutes (1.5GB memory limit, 3.4 GHz Pentium D). We observed that other planners do not check whether the empty plan is a solution. We therefore awarded each the value of the empty plan (in the time taken by LPRPG-P to compute it) in domains where the planner solved at least one problem (to avoid masking failures due to crashing/grounding). Comparison of LPRPG-P-A to HFF-A shows that our new heuristic finds much lower cost solutions than the standard RPG heuristic when used with anytime search. It is guided to find solutions satisfying preferences; whereas the standard RPG effectively performs blind search once the first feasible solution is found. In the absence of anytime search (LPRPG-P-1st Sol.) LPRPG-P still finds lower cost solutions than HFF (control); but, because it attempts to return a feasible solution quickly, rather than satisfying all preferences first, its solution quality is not as good as in the anytime setting. LPRPG-P shows good scalability in comparison to HFF in both settings.

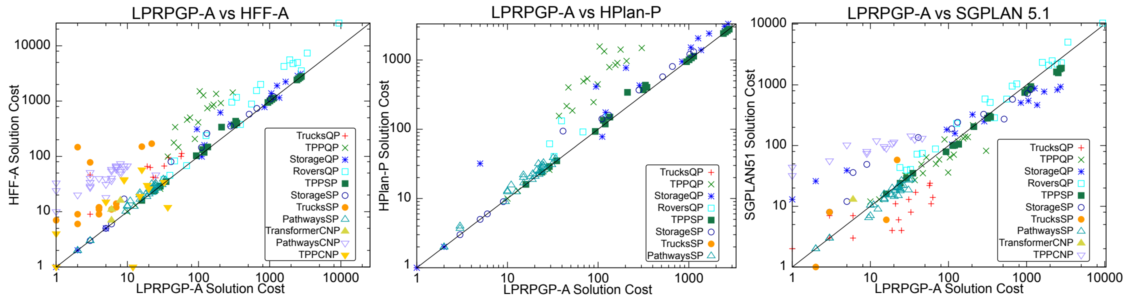
Comparing to existing planners, SGPLAN 5.1 dominates on most propositional domains; but its performance is significantly affected by the changes that disable SGPLAN 6 domain recognition. Indeed, overall, SGPLAN-W is outperformed by all other planners. Of the remaining planners LPRPG-P-A scales to solve many more problems and dominates in terms of solution quality in all but one domain. In this domain SGPLAN-W performs best. We do not understand precisely how it works; however, its behaviour appears consistent with the statement that it selects a set of prefer-



(a) IPC Score vs time: Propositional Domains



(b) IPC Score vs time: All Domains



(c) Comparing the Cost of the Best Solution Found After 30 Minutes on Mutually Solved Problems

Figure 2: IPC scores over time and Comparison on Mutually Solved Problems

ences to achieve up-front and plans to achieve these (Hsu *et al.* 2006). The current implementation is not, however, robust across this set of domains.

On the numeric planning domains LPRPG-P demonstrates exceptional performance compared to existing planners. The (-LP) figures, in which the LP is disabled, demonstrate that the LP (used only in numeric problems) improves performance; except in Pathways where increased per-node evaluation time outweighs benefits. The loss in Pathways is small compared to the much larger gains in the other domains. Despite being able to solve the equivalent competition problems, SGPLAN 5.1 fails on our debugged non-temporal problems. MIPS-XXL handles all of the domains; but, limited heuristic guidance with respect to preferences affects its scalability, and ability to find low cost solutions.

Next, we investigate solution improvement over time, Figures 2a,b, to see how we might expect planners to continue to scale. The greatest period of solution improvement is up to 10 seconds for most planners; however, the anytime planners continue to produce better solutions at 30 mins. Further, LPRPG-P-A achieves a greater score after 1 second than MIPS-XXL, HPLAN-P and SGPLAN-W do in 30 minutes. Performing EHC then BFS outperforms BFS alone.

Whilst a good measure of both coverage and solution quality, IPC scores can reward highly a planner that has strong coverage over one that produces lower cost solutions but does not scale. To address this we present results, Figure 2c, comparing LPRPG-P-A to the 2 best other planners above, and the standard RPG (-A) on mutually solved problems; points above the line indicate LPRPG-P generated a lower cost plan. Here we do not award the score of the empty plan, we include only solutions generated by the planners. Our solution cost is lower than that of the standard RPG and HPLAN-P; further it is comparable to that of SGPLAN 5.1.

## Acknowledgements

The authors would like to thank Giuseppe Turelli for early investigations in to this work and Sheila McIlraith for helpful discussions. We also thank J Benton, Emil Keyder, Jorge Baier and Shahid Jabbar for assistance with their planners.

## References

Baier, J.; Bacchus, F.; and McIlraith, S. 2007. A heuristic search approach to planning with temporally extended preferences. In *IJCAI 2007*.

Bell, K. R. W.; Coles, A. J.; Coles, A. I.; Fox, M.; and Long, D. 2009. The role of AI planning as a decision support tool in power substation management. *AI Communications* 22(1):37–57.

Benton, J.; Do, M. B.; and Kambhampati, S. 2009. Anytime heuristic search for partial satisfaction planning. *AIJ* 173.

Benton, J.; van den Briel, M.; and Kambhampati, S. 2007. A Hybrid Linear Programming and Relaxed Plan Heuristic for Partial Satisfaction Planning Problems. In *ICAPS 2007*.

Coles, A. I., and Smith, A. J. 2007. Marvin: A heuristic search planner with online macro-action learning. *JAIR* 28.

Coles, A. I.; Fox, M.; Long, D.; and Smith, A. J. 2008. A Hybrid Relaxed Planning Graph-LP Heuristic for Numeric Planning Domains. In *ICAPS 2008*.

Edelkamp, S.; Jabbar, S.; and Nazih, M. 2006. Large-Scale Optimal PDDL3 Planning with MIPS-XXL. In *IPC5 booklet, ICAPS*.

Gerevini, A. E.; Long, D.; Haslum, P.; Saetti, A.; and Dimopoulos, Y. 2009. Deterministic Planning in the Fifth International Planning Competition: PDDL3 and Experimental Evaluation of the Planners. *AIJ* 173.

Hoffmann, J. 2003. The Metric-FF Planning System: Translating Ignoring Delete Lists to Numeric State Variables. *JAIR* 20.

Hsu, C., and Wah, B. 2008. SGPlan 6 source code, http://ipc.informatik.uni-freiburg.de/planners, accessed 14/09/10.

Hsu, C.-W.; Wah, B.; Huang, R.; and Chen, Y. 2006. New Features in SGPlan for Handling Preferences and Constraints in PDDL3.0. In *IPC5 booklet, ICAPS*.

Kabanza, F., and Thiébaux, S. 2005. Search control in planning for temporally extended goals. In *ICAPS 2005*.

Keyder, E., and Geffner, H. 2009. Soft goals can be compiled away. *JAIR* 36.