

On Compiling Away PDDL3 Soft Trajectory Constraints without Using Automata

...

Abstract

We address the problem of propositional planning extended with the class of soft temporally extended goals supported in PDDL3, also called qualitative preferences since IPC-5. Such preferences are useful to characterise plan quality by allowing the user to express certain soft constraints on the state trajectory of the desired solution plans. We propose and evaluate a compilation approach that extends previous work on compiling soft reachability goals and always goals to the full set of PDDL3 qualitative preferences. This approach directly compiles qualitative preferences into propositional planning without using automata to represent the trajectory constraints. Moreover, since no numerical fluent is used, it allows many existing STRIPS planners to immediately address planning with preferences. An experimental analysis presented in the paper evaluates the performance of state-of-the-art propositional planners supporting action costs using our compilation of PDDL3 qualitative preferences. The results indicate that our approach is highly competitive with respect to current planners that natively support the considered class of preference, as well as with a recent automata-based compilation approach.

Introduction

Planning with preferences, also called “over-subscription planning” in [2, 5?], concerns the generation of plans for problems involving soft goals or soft state-trajectory constraints (called preferences in PDDL3[6]), that it is desired a plan satisfies, but that do not have to be satisfied. The quality of a solution plan for these problems depends on the soft goals and preferences that are satisfied.

PDDL3 supports a rich set of types of preferences expressed through certain modal operators, and in particular the qualitative preferences of types *at-end* (soft goals), *sometime*, *sometime-before*, *at-most-once*, and *sometime-after*. One of the competition tracks of IPC5 [6] was centered on qualitative preferences, and since then some systems addressing this class of planning problems have been developed. Most of these systems represents state trajectory constraints as automata that are either compiled into the problem operators and problem states, or are handled by the planning algorithm.

In this paper, we study propositional planning with these types of preferences through a compilation approach. In particular, we extend previous work on compiling soft reachability goals [9] and always goals [?] to a compilation that supports the full set of PDDL3 qualitative preferences. Differently from most other systems, our method directly compiles qualitative preferences into propositional planning with action costs without using automata to represent the state-trajectory constraints, and without using numerical fluents. Propositional planning with action costs is supported by many powerful planners, and the proposed compilation method allows them to also immediately support (through the compiled problems) qualitative preferences with no change to their algorithms and code.

An experimental analysis presented in the paper evaluates the performance of state-of-the-art propositional planners supporting action costs using our compilation of PDDL3 qualitative preferences. The results indicate that our approach is highly competitive with respect to current planners that natively support the considered class of preference, as well as with a recent automata-based compilation approach.

An experimental analysis presented in the paper evaluates the performance of state-of-the-art propositional planners supporting action costs using our compilation of PDDL3 qualitative preferences. The results indicate that our approach is highly competitive with respect to current planners that natively support the considered class of preference, as well as with a recent automata-based compilation approach.

Related Work

Our compilative approach has a structure inspired by the work of Keyder and Geffner [?] on compiling soft goals into STRIPS with action costs (here denoted with STRIPS+). In this work the compilation scheme introduces, for each soft goal p of the problem, a dummy goal p' that can be achieved only by two mutually exclusive actions. The first one, called *collect*(p), has cost 0 and requires p true when it is applied; the second one, called *forgo*(p), has cost equal to the utility of p and requires p false when it is applied. These actions can appear only at the end of the plan, and for each soft goal p and one of them has to appear.

The most prominent existing planners supporting PDDL3 preferences are HPlan-P [? ?], which won the “qualitative preference” track of IPC5, MIPS-XXL [? ?] and the more

recent LPRPG-P [4] and its extensions [?] where some different distance-to-go heuristics paired to a cost-to-go heuristic are exploited to promote the expansion of solution of better quality but further away. These planners represent preferences through automata whose states are synchronised with the states generated by the action plans, so that an accepting automaton state corresponds to preference satisfaction. For the synchronisation, HPlan-P and LPRPG-P use planner-specific techniques, while MIPS-XXL compiles the automata by modifying the domain operators and adding new ones modelling the automata transitions of the grounded preferences.

Our computation method is very different from the one of MIPS-XXL since, rather than translating automata into new operators, the problem preferences are compiled by only modifying the domain operators, possibly creating multiple variants of them. Moreover, our compiled files only use STRIPS+, while MIPS-XXL also uses numerical fluents.¹

The works on compiling LTL goal formulas by Cresswell and Coddington [?] and Rintanen [?] are also somewhat related to ours, but with important differences. Their methods handle *hard* temporally extended goals instead of preferences, i.e., every temporally extended goal must be satisfied in a valid plan, and hence there is no notion of plan quality referred to the amount of satisfied preferences. Rintanen's compilation considers only single literals in the always formulae (while we deal with arbitrary CNF formulas), and it appears that extending it to handle more general formulas requires substantial new techniques [?].

Bayer and McIlraith [?] observed that Crosswell and Coddington's approach suffers exponential blow up problems and performs less efficiently than HPlan-P.

The other recent works that are related to ours are the compilation approach in [?], which however supports only soft goals and always preferences, and the recent compilation schema by Wright, Mattmueller and Nebel (here abbreviated WMN) in [12], which supports a class of soft state-trajectory constraints richer than the PDDL3 qualitative preferences. WMN compiles soft trajectory constraints into conditional effects and state dependent action costs using LTL_f [?] and *automata*. Besides the use of automata, other main differences are the following ones. Our compilation has less additional fluents (at most two for each preference against one for each automaton state in WMN); *WMN uses a numerical fluent for the plan cost while we use numerical fluent; in WMN the plan cost can be increased as well as decreased (through the use of negative costs) at any point of the plan, while we only increase it and at the end of the plan (through the *forgo* actions).*²

¹Another compilation scheme using numerical fluents is considered in [6] to study the expressiveness of PDDL3 (without an implementation).

²The use of negative costs severely restricts the set of planners that can be used, since few planners support them. It appears that if WMN used only positive costs, it would loose optimality (from an optimal compiled plan we could not derive an optimal plan for the original problem). On the contrary, our method could be extended to use negative costs as in WMN without this problem.

$$\begin{aligned}
\langle s_0, s_1, \dots, s_n \rangle &\models (\text{at end } \phi) \text{ iff } S_n \models \phi \\
\langle s_0, s_1, \dots, s_n \rangle &\models (\text{always } \phi) \\
&\text{iff } \forall i : 0 \leq i \leq n \cdot S_i \models \phi \\
\langle s_0, s_1, \dots, s_n \rangle &\models (\text{sometime } \phi) \\
&\text{iff } \exists i : 0 \leq i \leq n \cdot S_i \models \phi \\
\langle s_0, s_1, \dots, s_n \rangle &\models (\text{at-most-once } \phi) \\
&\text{iff } \forall i : 0 \leq i \leq n \cdot \text{if } S_i \models \phi \text{ then} \\
&\quad \exists j : j \geq i \cdot \forall k : i \leq k \leq j \cdot S_k \models \phi \text{ and} \\
&\quad \forall k : k > j \cdot S_k \models \neg \phi \\
\langle s_0, s_1, \dots, s_n \rangle &\models (\text{sometime-after } \phi \psi) \\
&\text{iff } \forall i : 0 \leq i \leq n \cdot \text{if } S_i \models \phi \text{ then} \\
&\quad \exists j : i \leq j \leq n \cdot S_j \models \psi \\
\langle s_0, s_1, \dots, s_n \rangle &\models (\text{sometime-before } \phi \psi) \\
&\text{iff } \forall i : 0 \leq i \leq n \cdot \text{if } S_i \models \phi \text{ then} \\
&\quad \exists j : 0 \leq j < i \cdot S_j \models \psi
\end{aligned}$$

Figure 1: Semantics of the basic modal operators in PDDL3

Preliminaries, Background and Notation

A STRIPS problem is a tuple $\langle F, I, O, G \rangle$ where F is a set of fluents, $I \subseteq F$ and $G \subseteq F$ are the initial state and goal set, respectively, and O is a set of actions or operators defined over F as follows.

A STRIPS operator $o \in O$ is a pair $\langle \text{Pre}(o), \text{Eff}(o) \rangle$, where $\text{Pre}(o)$ is a set of atomic formulae over F and $\text{Eff}(o)$ is a set of literals over F . $\text{Eff}(o)^+$ denotes the set of positive literals in $\text{Eff}(o)$, $\text{Eff}(o)^-$ the set of negative literals in $\text{Eff}(o)$. An action sequence $\pi = \langle a_0, \dots, a_m \rangle$ is applicable in a planning problem Π if all actions a_i are in O and there exists a sequence of states $\langle s_0, \dots, s_{m+1} \rangle$ such that $s_0 = I$, $\text{prec}(a_i) \subseteq s_i$ and $s_{i+1} = s_i \setminus \{p \mid \neg p \in \text{Eff}(a_i)^-\} \cup \text{Eff}(a_i)^+$, for $i = 0 \dots m$. Applicable action sequence π achieves a fluent g if $g \in s_{m+1}$, and is a valid plan for Π if it achieves each goal $g \in G$ (denoted with $\pi \models G$).

A STRIPS+ problem is a tuple $\langle F, I, O, G, c \rangle$, where $\langle F, I, O, G \rangle$ is a STRIPS problem and c is a function mapping each $o \in O$ to a non-negative real number. The cost $c(\pi)$ of a plan π is $\sum_{i=0}^{|\pi|-1} c(a_i)$, where $c(a_i)$ denotes the cost of the i th action a_i in π and $|\pi|$ is the length of π .

PDDL3 [6] introduced state-trajectory constraints, which are modal logic expressions expressible using LTL that ought to be true in the state trajectory produced by the execution of a plan. Let $\langle s_0, s_1, \dots, s_n \rangle$ be the sequence of states in the state trajectory of a plan. Figure 1 defines PDDL3 qualitative state-trajectory constraints, i.e., constraints that do not involve numbers. Here ϕ and ψ are first-order formulae that, without loss of generality, we assume are translated into grounded CNF-formulae; e.g., $\phi = \phi_1 \wedge \phi_2 \wedge \dots \wedge \phi_n$ where ϕ_i ($i = 1 \dots n$) is a clause formed by literals over the problem fluents. These constraints can be either soft or hard. When they are hard soft are called *qualitative preferences*.

We will use the following notation: $\mathcal{A}, \mathcal{SB}, \mathcal{SA}, \mathcal{ST}, \mathcal{AO}, \mathcal{G}$ denote the classes of qualitative preferences of type always, sometime-before, sometime, at-most-once and soft goal, respectively, for a given planning problem; $\mathcal{A}_\phi, \mathcal{SB}_{\phi, \psi}, \mathcal{SA}_{\phi, \psi}, \mathcal{ST}_\phi, \mathcal{AO}_\phi, \mathcal{G}_\phi$ denote a particular preference over $\mathcal{A}, \mathcal{SB}, \mathcal{SA}, \mathcal{ST}, \mathcal{AO}, \mathcal{G}$, respectively, involving formulae ϕ and ψ ; moreover, if a plan π satisfies a preference P , we write $\pi \models P$.

Definition 1. A STRIPS+ problem with preferences is a tuple $\langle F, I, O, G, \mathcal{P}, c, u \rangle$ where:

- $\langle F, I, O, G, c \rangle$ is a STRIPS+ problem;
- $\mathcal{P} = \{\mathcal{P}_A \cup \mathcal{P}_{SB} \cup \mathcal{P}_{SA} \cup \mathcal{P}_{ST} \cup \mathcal{P}_{AO} \cup \mathcal{P}_G\}$ is the set of the preferences of Π where $\mathcal{P}_A \subseteq \mathcal{A}$, $\mathcal{P}_{SB} \subseteq \mathcal{SB}$, $\mathcal{P}_{SA} \subseteq \mathcal{SB}$, $\mathcal{P}_{ST} \subseteq \mathcal{ST}$, $\mathcal{P}_{AO} \subseteq \mathcal{AO}$ and $\mathcal{P}_G \subseteq \mathcal{G}$;
- u is an utility function $u : \mathcal{P} \rightarrow \mathbb{R}_0^+$.

STRIPS+ with preferences will be indicated with STRIPS+P.

Definition 2. Let Π be a STRIPS+P problem with preferences \mathcal{P} . The utility $u(\pi)$ of a plan π solving Π is the difference between the total amount of utility of the preferences by the plan and its cost: $u(\pi) = \sum_{P \in \mathcal{P}, \pi \models P} u(P) - c(\pi)$.

The definition of plan utility for STRIPS+P is similar to the one given for STRIPS+ with soft goals by Keyder and Geffner [?]. A plan π with utility $u(\pi)$ for a STRIPS+P problem is optimal when there is no plan π' such that $u(\pi') > u(\pi)$. The *violation cost* of a preference is the value of its utility.

In order to make the presentation of our compilation approach more compact, we introduce some further notation.

Given a preference clause $\phi_i = l_1 \vee l_2 \vee \dots \vee l_m$, the set $L(\phi_i) = \{l_1, l_2, \dots, l_m\}$ is the equivalent set-based definition of ϕ_i and $\bar{L}(\phi_i) = \{\neg l_1, \neg l_2, \dots, \neg l_m\}$ is the literal complement set of $L(\phi_i)$.

Given an operator o of a STRIPS+P problem, $Z(o)$ denotes the set of literals

$$Z(o) = (Pre(o) \setminus \{p \mid \neg p \in Eff(o)^-\}) \cup Eff(o)^+ \cup Eff(o)^-.$$

Note that the literals in $Z(o)$ hold in any reachable state resulting from the execution of operator o .

The state where an operator o is applied is indicated with s and the state resulting from the application of o with s' .

Definition 3. Given an operator o and a CNF formula $\phi = \phi_1 \wedge \dots \wedge \phi_n$, we define the set $C_\phi(o)$ of clauses of ϕ that o makes **certainly true** (independently from s) in s' as:

$$C_\phi(o) = \{\phi_i : |L(\phi_i) \cap Z(o)| > 0, i \in \{1 \dots n\}\}.$$

Given a clause $\phi_i = l_1 \vee \dots \vee l_{m_i}$ of ϕ , condition $|L(\phi_i) \cap Z(o)| > 0$ in Definition 3 requires that there exists at least a literal l_j , $j \in \{1 \dots m_i\}$ that belongs to $Z(o)$ and thus making clause ϕ_i true in s' .

Definition 4. Given an operator o and a CNF formula $\phi = \phi_1 \wedge \dots \wedge \phi_n$, we say that o **can make ϕ true** (dependently from s) if

1. $|C_\phi(o)| > 0$;
2. for each clause ϕ_i of ϕ not in $C_\phi(o)$, $\bar{L}(\phi_i) \not\subseteq Z(o)$.

Condition 1 in Definition 4 requires that there exists at least a clause of ϕ that is certainly true in s' (independently from s), while Condition 2 requires that the clauses that are not certainly true in s' are not certainly false in s' .

Definition 5. Given an operator o and a CNF formula $\phi = \phi_1 \wedge \dots \wedge \phi_n$, we say that o **can make ϕ false** (dependently from s) if

1. $|\bar{L}(\phi_i) \cap Z(o)| > 0 \wedge \bar{L}(\phi_i) \subseteq Z(o)$
2. $|L(\phi_i) \cap Z(o)| = 0$
3. $\bar{L}(\phi_i) \not\subseteq Pre(o)$.

The conditions of Definition 5 require that there exist at least a clause of ϕ that (1) has some (but not all) literals which are falsified after the execution of o , (2) has not literals which are true in the resulting state from the application of o and (3) this clause is not already false in the state where o is applied.

Operator-Preference Interactions

Operators and preferences may have different kinds of interactions, that we have to deal with in their compilation. We say that an operator o is *neutral* for a preference P if its execution in a plan can affect the satisfaction of P in the state trajectory of the plan. Otherwise, depending on the preferences type of P , o can behave as a *violator*, a *threat* or a *potential support* of P . Informally, a violator falsifies the preference, a threat may falsify it (depending on s), and a potential support may satisfies it over the full state trajectory of the plan. In the following, more formal definitions are given for each type of preference.

Operators Affecting Always Preferences

An always preference A_ϕ is violated if ϕ is false in any state on the plan state trajectory. Hence, if the state s' generated by an operator o makes ϕ false, then o is a violator of A_ϕ .

Definition 6. Given an operator o and an always preference A_ϕ of a STRIPS+P problem, o is a **violator** of A_ϕ if there is a clause ϕ_i of ϕ such that: (1) $\bar{L}(\phi_i) \subseteq Z(o)$, and (2) $\bar{L}(\phi_i) \not\subseteq Pre(o)$.

Operator o is a threat of A_ϕ if it is not a violator, its effects make false at least a literal of a clause ϕ_i of ϕ , and its preconditions don't entail $\neg \phi_i$ (otherwise A_ϕ would be already false in s). Such clause ϕ_i is a *threatened clause* of A_ϕ .

Definition 7. Given an operator o and an always preference A_ϕ of a STRIPS+P problem, o is a **threat** of A_ϕ if it is not a violator and it can make ϕ false.

The set of clauses of a preference A_ϕ threatened by o is denoted with $TC(o, \phi)$.

An operator is neutral for A_ϕ if it makes ϕ true, does not falsify any clause of ϕ , or it can be applied only to states where ϕ is false.

Definition 8. Given an operator o and an always preference A_ϕ of a STRIPS+P problem, o is **neutral** for A_ϕ if:

1. for all clauses ϕ_i of ϕ , $|L(\phi_i) \cap Z(o)| > 0$ or $|\bar{L}(\phi_i) \cap Z(o)| = 0$, or
2. there exists a clause ϕ_i of ϕ such that $\bar{L}(\phi_i) \subseteq Pre(o)$.

Example. An operator $o = \langle \neg a, \neg b \rangle$ is a threat of A_{ϕ_1} , a violator of A_{ϕ_2} and neutral for A_{ϕ_3} where $\phi_1 = c \vee b$, $\phi_2 = a \vee b$ and $\phi_3 = d$.

Operators Affecting Sometime Preferences

A sometime preference ST_ϕ is violated if there is not at least one state in which ϕ is true on the plan state trajectory. Hence, if the state s' generated by an operator o makes ϕ true, then o is a potential support of ST_ϕ .

Definition 9. Given an operator o and a sometime preference ST_ϕ of a STRIPS+P problem, o is a **potential support** for ST_ϕ if o can make true ϕ , otherwise the operator is **neutral** for ST_ϕ .

Example. An operator $o = \langle \top, \neg b \rangle$ is a potential support of ST_{ϕ_1} and neutral for ST_{ϕ_2} where $\phi_1 = c \vee \neg b$ and $\phi_2 = c$.

Operators Affecting Sometime-before Preferences

A sometime-before preference $SB_{\phi,\psi}$ is violated if ϕ becomes true before ψ has been made true on the plan state trajectory. Hence, if the state s' generated by an operator o can make ψ true, then o is a potential support of $SB_{\phi,\psi}$. Depending by the state where it is applied such operator could behave as a actual support or as a neutral operators for $SB_{\phi,\psi}$.

Definition 10. Given an operator o and a sometime-before preference $P = SB_{\phi,\psi}$ of a STRIPS+P problem, o is a **potential support** for P if o can make ψ true.

If the state s' generate by an operator o can make ϕ true, then o is a threat of $SB_{\phi,\psi}$. Depending by the state where it is applied such operator could behave as a violator or as a neutral operators for $SB_{\phi,\psi}$.

Definition 11. Given an operator o and a sometime-before preference $P = SB_{\phi,\psi}$ of a STRIPS+P problem, o is a **threat** for P if o can make true ϕ .

The set of sometime-before preferences of Π which are threatened by the operator o are denoted with $T_{SB}(o)$.

If the state s' generated by an operator o can not make neither ϕ nor ψ true, then o is a neutral operator for $SB_{\phi,\psi}$ regardless of the state in which it is applied.

Definition 12. Given an operator o and a sometime-before preference $P = SB_{\phi,\psi}$ of a STRIPS+P problem, o is a **neutral** for P if o is not a potential support or a threat.

Example. An operator $o = \langle \top, b \rangle$ is a potential support of SB_{ϕ_1,ψ_1} , a threat of SB_{ϕ_2,ψ_2} and neutral for SB_{ϕ_3,ψ_3} where $\phi_1 = c$ and $\psi_1 = a \vee b$, $\phi_2 = c \vee b$ and $\psi_2 = d$ and $\phi_3 = d$ and $\psi_3 = e$.

Operators Affecting At-most-once Preferences

An at-most-once preference AO_ϕ is violated if ϕ is false if ϕ becomes true more than once on the plan state trajectory. Hence, if the state s' generated by an operator o can make ϕ true, then o is a threat of AO_ϕ .

Definition 13. Given an operator o and an at-most-once preference $P = AO_\phi$ of a STRIPS+P problem, o is a **threat** of P if o can make true ϕ , otherwise o is **neutral** for AO_ϕ .

Example. An operator $o = \langle \top, \neg b \rangle$ is a potential support of AO_{ϕ_1} where $\phi_1 = c \vee \neg b$.

Operators Affecting Sometime-After Preferences

A sometime-after preference $SA_{\phi,\psi}$ is violated if ϕ becomes true in a state without ψ becoming true in a succeeding state on the plan state trajectory. Hence, if the state s' generated by an operator o can make ϕ true, then o threatens $SA_{\phi,\psi}$ because ψ could be false in s' ; o threatens the preference also if it can make ψ false, because ϕ could be true in s' .

Definition 14. Given an operator o and a sometime-after preference $P = SA_{\phi,\psi}$ of a STRIPS+P problem, o is a **threat** for P if o can make ϕ true or ψ false.

If the state s' generated by an operator o can make ψ true but certainly can not make ϕ true then o is a potential support of $SA_{\phi,\psi}$ because if the preference is temporarily violated in s' , o could make it satisfied again in the following state.

Definition 15. Given an operator o and a sometime-after preference $SA_{\phi,\psi}$ of a STRIPS+P problem, o is a **potential support** of $SA_{\phi,\psi}$ if o is not a threat of the preference and o can make true ψ .

The set of sometime-after preferences of Π which are potentially supported by the operator o are denoted with $S_{SA}(o)$.

Definition 16. Given an operator o and a sometime-after preference $SA_{\phi,\psi}$ of a STRIPS+P problem, o is a **neutral** for $SA_{\phi,\psi}$ if o is neither a threat nor a potential support of preference $SA_{\phi,\psi}$.

Example. An operator $o = \langle \top, \neg a, b \rangle$ is a potential support of SA_{ϕ_1,ψ_1} , a threat of SA_{ϕ_2,ψ_2} and SA_{ϕ_3,ψ_3} and neutral for SA_{ϕ_4,ψ_4} where $\phi_1 = c$ and $\psi_1 = \neg a$, $\phi_2 = \neg a$ and $\psi_2 = b \vee d$, $\phi_3 = \neg d$ and $\psi_3 = b$, and $\phi_4 = \neg c$ and $\psi_4 = d$.

Compilation of Qualitative Preferences

In this section we describing the general compilation scheme of a STRIPS+P Π problem. First we compile Π into a problem with conditional effects (and possibly also disjunctive preconditions), which can then be compiled away obtaining a STRIPS+ problem equivalent to Π . We will use $O_{neutral}$ to denote the set of the problem operators that are neutral for all preferences, and $P_{affected(o)}$ to denote the set of all problem preferences that are affected by an operator o .

Since the compilation of soft goals is the same as in [?], we omit its description and focus on the other types of preferences. Moreover, we use a preprocessing step to filter out all preferences of type \mathcal{A} and \mathcal{SB} that are falsified in the initial state and all preferences of type \mathcal{ST} that are satisfied in it.

For a STRIPS+P problem $\Pi = \langle F, I, O, G, \mathcal{P}, c, u \rangle$, the compiled problem of Π is $\Pi' = \langle F', I', O', G', c' \rangle$ where:

- $F' = F \cup V \cup D \cup C \cup \overline{C'} \cup \{normal-mode, end-mode\}$;
- $I' = I \cup \overline{C'} \cup V_{ST} \cup S_{AO} \cup \{normal-mode\}$;
- $G' = G \cup C'$;
- $O' = \{collect(P), forgo(P) \mid P \in \mathcal{P}\} \cup \{end\} \cup \{comp(o, \mathcal{P}) \mid o \in O\}$
- $forgo(P) = \langle \{end-mode, P-violated, \overline{P'}\}, \{P', \neg \overline{P'}\} \rangle$;
- $collect(P) = \langle \{end-mode, \neg P-violated, \overline{P'}\}, \{P', \neg \overline{P'}\} \rangle$;
- $end = \langle \{normal-mode\}, \{end-mode, normal-mode\} \rangle$;
- $comp(o, \mathcal{P})$ is the function translating operator o according to Definition 17;
- $c'(o') = \begin{cases} u(P) & \text{if } o' = forgo(P) \\ c(o') & \text{if } o' = comp(o, \mathcal{P}) \\ 0 & \text{otherwise;} \end{cases}$

- $V = \{P\text{-violated} \mid P \in \mathcal{P}\};$
- $V_{ST} = \{P_i\text{-violated} \mid P_i \in \mathcal{P}_{ST}\};$
- $S_{AO} = \{P_i\text{-seen} \mid P_i \in \mathcal{P}_{AO}(I)\},$ with $\mathcal{P}_{AO}(I) = \{P_i = AO_\phi \mid P_i \in \mathcal{P}_{AO}, I \models \phi\};$
- $C' = \{P' \mid P \in \mathcal{P}\}$ and $\overline{C'} = \{\overline{P'} \mid P \in \mathcal{P}\};$

The *collect* and *forgo* actions can only appear at the end of the plan. For each preference P the compilation of Π into Π' adds a dummy hard goal P' that is false in the initial state I' ; P' be achieved either by action *collect*(P), that has cost 0 but requires P to be satisfied, or by action *forgo*(P), that has cost equal to the utility of P and can be performed only if P is false (P -violated is true in the goal state). For each P , exactly one of *collect*(P) and *forgo*(P) appears in the plan.

The P_i -violated literals in the compiled initial state I' are used to consider any ST preference violated until a operator supporting it is inserted into the plan; the P_i -seen literals in I' are necessary to capture the violation of any AO preference when an operator makes the preference formula true for the second time in the state trajectory.

Function *comp*(o, \mathcal{P}) transforms an original operator o into the equivalent compiled operator o' with an additional preconditions forcing it to appear before the *forgo* and *collect* operators. Regarding the effects of o' , if $o \in O_{neutral}$, they are the same of o ; otherwise, *comp*(o, \mathcal{P}) extends the effects of o in o' with a set of conditional effects for each preference affected by o . The definition of such additional effects depend on the type of the affected preference and on how o interacts with it; this is detailed below.

Definition 17. Given an operator o the corresponding compiled operator is defined using the following function:

$$\begin{aligned} Pre(o') &= Pre(o) \cup \{normal\text{-}mode\} \\ Eff(o') &= Eff(o) \cup \bigcup_{P_i \in P_{affected}(o)} \mathcal{W}(o, P_i) \end{aligned}$$

where $\mathcal{W}(o, P_i)$ is the set of conditional effects concerning the affected preference P_i (if any).

In the following ϕ_i denotes a clause of ϕ , ψ_i denotes a clause of ψ , and a set of formulas is interpreted as their conjunction.

Conditional Effects for \mathcal{A} Preferences

The conditional effects for a compiled operator affecting a preference A_ϕ are defined as follows, where $\overline{AA}(o)_{\phi_i}$ is the literal-complement set of the subset of literals in $L(\phi_i)$ that are not falsified by the effects of o .

Definition 18. Given a preference $P = A_\phi$ and an operator o affecting it, the conditional effect set $\mathcal{W}(o, P)$ in the compiled version o' of o (according to Definition 17) is:

$$\mathcal{W}(o, P) = \begin{cases} \{\text{when } (cond(o, P)) \text{ } (P\text{-violated})\} \\ \quad \text{if } o \text{ is a threat of } P \\ \{\text{when } (\top) \text{ } (P\text{-violated})\} \\ \quad \text{if } o \text{ is a violator for } P \end{cases}$$

where $cond(o, P) = \bigvee_{\phi_i \in TC(o, P)} (l_1 \wedge \dots \wedge l_q)$
and $\{l_1, \dots, l_q\} = \overline{AA}(o)_{\phi_i}$.

For each affected preference $P = A_\phi$, o' contains a conditional effect P -violated with a condition depending on how A_ϕ is affected: if o is a violator, then the condition is always true; if o is a threat, the condition checks that there exists at least a clause of ϕ of that is certainly false in s' – this is the case if there is at least a threatened clause whose literals that are not falsified in s' are false in s .

Example. Consider $o = \langle \top, \{a, \neg c\} \rangle$ and preference A_ϕ with $\phi = (a \vee b) \wedge (c \vee d \vee e)$. The second clause of ϕ is threatened by o , and $cond(o, A_\phi) = \overline{AA}(o)_{c \vee d \vee e} = \{\neg d, \neg e\}$.

Conditional Effects for ST Preferences

The conditional effects for a compiled operator affecting a preference ST_ϕ are defined as follows.

Definition 19. Given a preference $P = ST_\phi$ and an operator o that potentially supports it, the conditional effect set in the compiled version o' of o (according to Definition 17) is:

$$\mathcal{W}(o, P) = \{\text{when}(cond(o, P)) \text{ } (\neg P\text{-violated})\}$$

where $cond(o, P) = \{\phi_i \mid \phi_i \notin C_\phi(o)\}$.

As described above, for each $P = ST_\phi$ P -violated holds in the compiled initial state, and a potential support o of P makes ϕ true when all clauses ϕ_i of ϕ not in $C_\phi(o)$ hold in s , where $C_\phi(o)$ is the set of clauses of ϕ that will be certainly true in s' . If this condition holds in s , then o' falsifies it in s' .

Conditional Effects for SB Preferences

The conditional effects for a compiled operator affecting a preference $SB_{\phi, \psi}$ are defined as follows.

Definition 20. Given a preference $P = SB_{\phi, \psi}$ and an operator o affecting it, the conditional effect set $\mathcal{W}(o, P)$ in the compiled version o' of o (according to Definition 17) is:

$$\mathcal{W}(o, P) = \begin{cases} \{\text{when } (cond_S(o, P)) \text{ } (seen\text{-}\psi)\} \\ \quad \text{if } o \text{ is a potential support of } P \\ \{\text{when } (cond_T(o, P)) \text{ } (P\text{-violated})\} \\ \quad \text{if } o \text{ is a threat of } P \end{cases}$$

where:

- $cond_S(o, P) = \{\psi_i \mid \psi_i \notin C_\psi(o)\}$
- $cond_T(o, P) = \{\neg seen\text{-}\psi\} \cup \{\phi_i \mid \phi_i \notin C_\phi(o)\}.$

An operator o affecting a preference $P = SB_{\phi, \psi}$ can behave as a (a) potential support of P , (b) a threat of P , or (c) both. These case are captured by the three conditional effects of o' in Definition 20.

In case (a), if all clauses that are not certainly true in s' (i.e., $cond_S(o, P)$) hold in s , then ψ is true in s' , and o' keeps track of this by making *seen- ψ* true. In case (b), if ψ has never been true in the state-trajectory up to s and all clauses of ϕ that are not certainly true (i.e., $cond_T(o, P)$) hold in s , then P is violated by o and o' has effect P -violated. In case (c), if the conditions of both conditional effects hold, P is violated because ψ is made true simultaneously with ϕ .

Conditional effects for \mathcal{AO} Preferences

The conditional effects for a compiled operator threatening preference \mathcal{AO}_ϕ preference are defined as follows.

Definition 21. Given a preference $P = \mathcal{AO}_\phi$ and an operator o that threatens P , the conditional effect set $\mathcal{W}(o, P)$ in the compiled version o' of o (according to Definition 17) is:

$$\mathcal{W}(o, P) = \{ \text{when } (cond_N(o, P)) \text{ (seen-}\phi) \\ \text{when } (cond_T(o, P)) \text{ (P-violated)} \}$$

where:

- $cond_N(o, P) = \{\neg \text{seen-}\phi\} \cup \{\phi_i \mid \phi_i \notin C_\phi(o)\}$
- $cond_T(o, P) = \{\text{seen-}\phi\} \cup \{\phi_i \mid \phi_i \notin C_\phi(o)\} \cup \{\bigvee_{\phi_i \in C_\phi(o)} (\neg l_1 \wedge \dots \wedge \neg l_q) \mid \{l_1, \dots, l_q\} = L(\phi_i)\}.$

If an operator o affecting $P = \mathcal{AO}_\phi$ makes ϕ true for the first time in the state trajectory (i.e., $cond_N(o, P)$ holds in s), then the first conditional effect of o' keeps track that ϕ has become true. Otherwise, if (1) ϕ was true in any state before s' , (2) the execution of o in s makes ϕ true in s' , and (3) ϕ was false before (i.e., the three condition sets in $cond_T(o, P)$), then o violates P and o' has effect P -violated.

Conditional effects for \mathcal{SA} Preferences

The conditional effects for a compiled operator affecting preference \mathcal{SA}_ϕ preference are defined as follows.

Definition 22. Given a preference $P = \mathcal{SA}_\phi$ and an operator o that affects P , the conditional effect set $\mathcal{W}(o, P)$ in the compiled version o' of o (according to Definition 17) is:

$$\mathcal{W}(o, P) = \begin{cases} \{ \text{when } (cond_T(o, P)) \text{ (P-violated)} \} \\ \quad \text{if } o \text{ is a threat on } \phi \text{ of } P \\ \{ \text{when } (cond_S(o, P)) \text{ (}\neg P\text{-violated)} \} \\ \quad \text{if } o \text{ is a support of } P \end{cases}$$

where:

- $cond_T(o, P) = \{R(o, \phi), R(o, \neg\psi)\}$
- $cond_S(o, P) = \{R(o, \psi)\}$ **CHECK: versione corretta di $R(o, \phi?)$**
- $R(o, \phi) = \begin{cases} \top \text{ if } \forall \text{ clause } \phi_i \text{ of } \phi, |Z(o) \cap L(\phi_i)| > 0 \\ \bigwedge_{\phi_i \notin C_\phi(o)} (l_1 \vee \dots \vee l_q) \text{ where } \{l_1, \dots, l_q\} \\ \text{is the set of literals of } \phi_i \text{ not falsified by } o. \end{cases}$

An operator o affecting a preference $P = \mathcal{SA}_{\phi, \psi}$ can behave as (i) a threat of P or (ii) a potential support of P . In case (i) the effect condition captures the fact that o generates a state s' where ϕ is true and ψ is false, (temporarily) violating P . In case (b) the condition captures the fact that o generates s' in which ψ is true, and so s' cannot violate P .

Compilation equivalence

It can be proved that the original planning problem with preferences has a solution plan with a certain cost (sum of its action costs and of the violated preference costs) if only if the compiled plan has a solution with the same cost.

Proposition 1. Let Π' be the compiled problem with conditional effects of a STRIPS+ problem Π . From any plan π solving Π we can derive a plan π' solving Π' , and viceversa, such that the costs of π and π' are the same.

Proof. (Sketch). The proof has the same structure of the plan-correspondence proof for Keyder and Geffner's compilation of soft goals [?], with $\pi' = \langle \pi'', end, \pi''' \rangle$ in which π'' is obtained from π by replacing the original operators with the compiled ones involving conditional effects, and the rest of π' defined as in Keyder and Geffner's proof (π''' involves only *collect* and *forgo* actions). Since the conditional effects in π'' affect only the additional fluents of the compiled problem, all original operator preconditions remain satisfied in the state trajectory of π'' . Moreover, by construction of the conditional effects for the compiled operators, it can be proved that in the state where *end* is applied, for each preference P , P -violated holds if and only if P is violated in π . Viceversa, from a valid plan π' we can obtain a plan for π by replacing the compiled operators with their original version, and removing *end* and all *collect/forgo* actions. \square

Compilation of Conditional Effects

According to the described compilation schema, each compiled operator o' of an original operator o affecting n preferences has $m \leq 2n$ conditional effects.

In the literature, there are two main general methods for compiling away conditional effects. In the first method by Gazen and Knoblock's [?], each plan of the compiled problem preserves the length of the corresponding plan for the original problem, but an exponential number of compiled operators are generated (in our context $O(2^m)$ operators for each o'). In the second method by Nebel [?], a polynomial number of new operators are generated, but each plans for the compiled problem increases polynomially the length.

In our context, we use Nebel's method because, depending on the operators' structure and the input preferences, many conditional effects can be generated, making the other approach impractical. Moreover, Nebel's method can be optimised for our conditional effects because of their particular structure. In particular **conflicts can not arise between the effects of conditional effects referring to different preferences, while those referring to the same preference can be resolved by imposing an order of evaluation.**

This allows us to simplify the compilation by omitting the so called "copy-operators" of Nebel's method. Another optimization concerns the ordering of the set of operator pairs "activating" the conditional effects, which in the original version of unordered, while in our context they can be ordered as a sort of macro operators. For lack of space, in this paper we don't give a detailed description of these optimisations, which leads to a revised method similar to the technique described in [?] for always constraints, that we extended to deal with every class of PDDL3 qualitative preferences.

Planner	Rovers	TPP	Trucks	Openstacks	Storage	TOTAL
LAMAp(h_R)	16.98	8.34	15.43	19.28	18.48	78.51
FDRemix	17.89	7.1	17.8	18.99	16.21	78.0
FDSS 2018	17.6	7.03	17.21	18.7	17.12	77.66
LAMA(2011)	17.01	7.53	13.16	18.42	17.83	73.94
IBaCoP2	19.62	9.68	10.0	17.85	15.73	72.88
LPRPG-P	11.36	18.74	7.1	19.71	12.88	69.78
MIPlan	17.65	8.8	9.23	17.35	14.42	67.46
Mercury	16.07	6.57	7.84	18.06	14.51	63.04

Table 1: IPC comparison calculated using all kinds of preferences together. LPRPG-P is the planning system which natively support preferences, while the others are all classical planners. The considered planning system are sorted by the total IPC score. The best performance are indicated in bold.

Experimental Evaluation

We implemented the proposed compilation scheme, and we have compared the performance (plan quality) of propositional planners using it and of LPRPG-P, a the state-state-of-the art system for satisficing planning with PDDL3 preferences [4]. We chose LPRPG-P because we observed it has the best overall performance for the considered benchmarks, over all available systems supporting PDDL3 preferences.

Moreover, we have evaluated the effectiveness of using our compilation for optimal planning with preferences against an alternative recent compilation based on automata.

We used a selection of the best performing planners at IPC-8 and IPC-9 [11, 1]: LAMA [10], Mercury [7], MIPlan [8], IBaCoP2 [3], Fast Downward Stone Soup 2018[?] (abbreviated FDSS), Fast Downward Remix[?] (abbreviated FDRemix). In addition, we used a recent version of LAMA, called LAMAp(h_R), exploiting admissible heuristic h_R for testing reachability of soft goals during search [9].

As benchmarks we used all (100) original problems of the qualitative preference track of IPC5 [6], which has five domains, i.e. Rovers, TPP, Trucks, Openstacks and Storage, of which the problems in Storage and TPP have no hard goal. In these benchmarks there is no preference of type sometime-after.

The propositional planners were run on the compiled problems, while LPRPG-P was run on the original problems. All the experiments were conducted on a 2.00GHz Core Intel(R) Xeon(R) CPU E5-2620 machine with CPU-time and memory limits of 30 minutes and 8GB, respectively.

The compared planners are evaluated using the IPC quality score (introduced in IPC-6). Given a planner p and a planning instance i , if p solves i , the following score is assigned to p : $score(p, i) = \frac{cost_{best}(i)}{cost(p, i)}$, where $cost_{best}(i)$ is the cost of the best known solution for i found by any planner, and $cost(p, i)$ is the cost of the best solution found by p , using at most 30 CPU minutes. If p does not find a solution, then $score(p, i) = 0$.

We also consider another metric of plan quality evaluation, that we denote α_{cost} : if planner p solves instance i , we assign the following score to p

$$\alpha_{cost}(p, i) = \frac{cost(p, i)}{cost_{total}(i)} = \frac{\sum_{P \in \mathcal{P}(i) : \pi \notin P} c(P)}{\sum_{P \in \mathcal{P}(i)} c(P)}$$

where $\mathcal{P}(i)$ is the set of the preferences in i . Note that

\mathcal{P}_A						
Planner	Rovers	TPP	Trucks	Openstacks	Storage	TOTAL
LAMAp(h_R)	14.91	20.0	15.0	20.0	19.0	88.91
FDSS 2018	14.75	17.0	18.0	17.83	20.0	87.59
MIPlan	15.27	20.0	12.0	19.0	20.0	86.27
LPRPG-P	15.02	7.0	0.0	19.5	11.0	52.52
\mathcal{P}_G						
Planner	Rovers	TPP	Trucks	Openstacks	Storage	TOTAL
LPRPG-P	—	19.45	16.48	19.57	14.96	70.47
FDSS 2018	—	14.66	16.49	18.55	18.46	68.16
LAMAp(h_R)	—	14.82	14.36	18.85	18.92	66.94
MIPlan	—	15.08	9.85	16.84	19.32	61.09
\mathcal{P}_{AO}						
Planner	Rovers	TPP	Trucks	Openstacks	Storage	TOTAL
FDSS 2018	17.22	18.0	20.0	—	19.0	74.22
LAMAp(h_R)	15.33	19.0	20.0	—	19.0	73.33
MIPlan	14.76	17.0	15.0	—	20.0	66.76
LPRPG-P	14.11	2.0	19.0	—	12.0	47.11
\mathcal{P}_{SB}						
Planner	Rovers	TPP	Trucks	Openstacks	Storage	TOTAL
LAMAp(h_R)	18.6	20.0	18.0	—	19.0	75.6
MIPlan	18.66	20.0	12.0	—	20.0	70.66
FDSS 2018	17.92	17.0	16.5	—	19.0	70.42
LPRPG-P	8.63	14.0	15.5	—	7.0	45.13
\mathcal{P}_{ST}						
Planner	Rovers	TPP	Trucks	Openstacks	Storage	TOTAL
LAMAp(h_R)	15.3	10.0	—	—	19.0	44.3
FDSS 2018	17.2	8.0	—	—	19.0	44.2
LPRPG-P	10.42	17.0	—	—	14.0	41.42
MIPlan	15.86	9.0	—	—	14.0	38.86

Table 2: IPC comparison calculated considering all kinds of preferences separately. Each subtable concerne a single class of preferences which is indicated in the first row. LPRPG-P is the planning system which natively support preferences, while the others are all classical planners. The considered planning system are sorted in each subtable by the total IPC score. The best performance are indicated in bold.

$\alpha_{cost}(p, i)$ can vary between 0 and 1; if $\alpha_{cost}(p, i) = 0$, then $cost(p, i) = 0$ and p has found an optimal plan for i preference-wise. On opposite side, if $\alpha_{cost}(p, i) = 1$, p has found the worst plan for i (all preferences are violated). If for two planners p and p' we have $\alpha_{cost}(p, i) < \alpha_{cost}(p', i)$, then p performs better than p' for i , and the difference between these metric values quantifies the performance discrepancy.

Concerning the optimal planning, similarly to what done in [12], we have tested our scheme using LAMA with three admissible heuristics: h^{blind} , assigning zero to goal states and 1 to every other state, h^{max} [?], h^{cpdb} [?].

For this analysis we generated a set of benchmarks using the same methodology as in [12]. Starting from the IPC5 problems, we created simpler instances by randomly sampling subsets of the soft trajectory constraints in original instances: from each original instance, five new instances are generate, each of which has with 1%, 5%, 10%, 20% and 40% of the original (grounded) soft trajectory constraints, while the hard goals are all unchanged, if they exist. Since the authors of the aforementioned paper did not make their sampled instances available, we generated, for each sampling percentage (except for 100 %), 3 sampled instances and considered the average performace on them.

Experimental Results

Tables 1 and 2 give the performances of the compared planners in term of IPC quality score aggregated by domain. The results in Table 1 concern plan quality considering all pref-

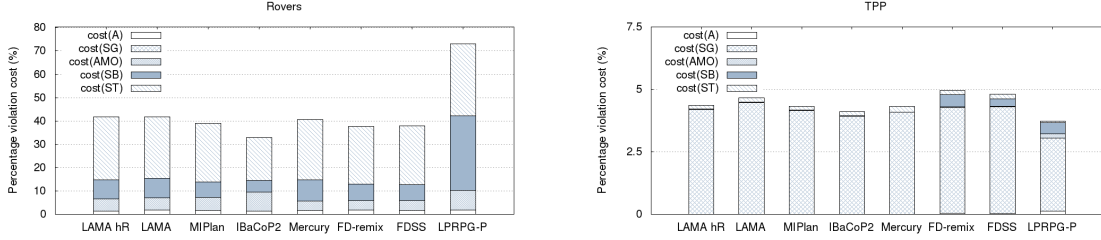


Figure 2: α_{cost} analysis for Rovers and TPP domain.

erences, while those in Table 2 concern plan quality when only the specified preferences of a particular class are used for the plan evaluation.

The analysis in Table 2 is restricted to a subset of those considered in Table 1. Overall the compilation approach performs better than LPRPG-P, with six planners obtaining better total IPC scores. The comparison considering each preference class separately shows good performance for every preference class except for soft goals.

In Rovers, Trucks and Storage each considered planner performs better than, or at least similarly to, LPRPG-P (except for Mercury in Trucks); IBaCoP2 performs particularly well in Rovers, FDRemix in Trucks and LAMAP(h_R) in Storage. Also MIPlan works well in Trucks but it is penalized due to coverage (it solves only 15 instances out of 20). The tested planners from IPC9, FDRemix and Fast Downward Stone Soup 2018, perform overall better than those from IPC8, but LAMAP(h_R) is better than everyone else (it improves the performance of LAMA in all the considered domains except Rovers, where there is no soft goal and h_r is not exploited for pruning).

On the other hand, LPRPG-P performs comparably with LAMAP(h_R) in Openstacks and much better than the other planners in TPP. The bad performances of the compilation approach in TPP is mainly due to presence of many soft goals and, as shown in [9], compiling soft goals through Keyder and Geffner’s method can sometimes be problematic. Indeed Table 2 shows that for soft goals LPRPG-P has higher IPC score than all others planners. We can also observe that, compared to LPRPG-P, the classical planners achieve better results for preferences of classes always, sometime-before and at-most-once, but this has not a crucial impact of the overall plan quality, because in these problems violating the soft-goals is more expensive than the other preferences (or equivalently they are more useful to satisfy than the other preferences).

The comparison of the planners’ performance using the α_{cost} helps to further understand the behaviour of the planners. For lack of space, we will focus this analysis on two selected domains: Rovers, one of the domains where the compilation approach works better, and TPP, the only domain where we observed poor performance compare to LPRPG-P. Figure 2 shows, for each preferences class, the planners’ α_{cost} values obtained by adding the relative α_{cost} for every instance in the of these two benchmark domains. Each level of the stacked histogram represents the aggregated α_{cost} re-

Domain	h^{blind}		h^{max}		h^{cpdb}	
	WMB	Our	WMB	Our	WMB	Our
Storage	24.78	57.0	29.2	45.0	23.10	57.0
Rovers	17.4	24.0	21.43	25.0	15.17	23.0
Trucks	18.84	24.0	23.19	25.0	n/a	25
TPP	—	47.0	—	45.0	—	40.0

Table 3: Coverage of our and Wright, Matmueller and Nebel (WMB)’s compilation schemes on the IPC5 benchmarks set with additional sampled soft-trajectory constraints. “n/a” means that the considered heuristic was not applicable; “—” means that no data are reported in [12].

stricted to a specific class of preferences, which indicates how much each class of the violated preferences contributes to the total cost of the plans.

For Rovers, the IPC-score gap between the classical planners and LPRPG-P is due to LPRPG-P’s violation of the sometime-before preferences. Regarding the other classes preferences, the violation costs in the generated plans are similar except for IBaCoP2, that satisfies more sometime-before and sometime preferences than the others planning, and violates more at-most-once preferences, generally obtaining better quality plans.

For TPP, indeed looking at Figure 2 we note that the most important preferences in this domain are the soft goals, which are better satisfied by LPRPG-P. The search pruning technique in LAMAP(h_R) slightly helps LAMA to achieve more soft goals, but not enough to reach the performance of LPRPG-P.

Table 3 gives results about using our compilation scheme for optimal planning. For each of the three considered versions of LAMA using admissible heuristics, the table indicates the percentage of solved problems. The results are compared with those of reported in [12] for WMN’s compilation.³ Domain Openstacks here is not considered because no one of the considered planner solved any instance. Also note that results for TPP are missing because they were not reported in [12] they were not reported in their work.

According to these results, the compilation approach without automata seems quite preferable, because a higher coverage is obtained in all three considered domains. This is the case even if the machine that we used for our experiments is less powerful (CPU and memory wise) than the one

³A more detailed comparison with WMN’s approach is very difficult because, at the time of writing, WMN’s compiler, compiled files and solution plans are unavailable.

used in [12], and moreover our CPU-time limit was half of that used in [12].

Conclusions

We have proposed a new compilation schema for solving propositional planning with the full class of PDDL3 soft state-trajectory constraints called qualitative preferences. Our work significantly extends the approach of Keyder and Geffner, that is restricted to only softgoals. Experimental results show that, despite the compilation of only soft goals may be less effective than other approaches, such LPRPG-P, for soft state-trajectory constraints our compilation is quite competitive with state-of-the-art systems supporting them. Since the planning language of the compiled problem is very simple, many available planners can use it.

References

- [1] The 2018 international planning competition (IPC) classical tracks. <https://ipc2018-classical.bitbucket.io>. Accessed: 2018-10-12.
- [2] M. Briel, R. Sanchez, M. Do, and S. Kambhampati. Effective approaches for partial satisfaction (over-subscription) planning. In *Proceedings of Nineteenth National Conference on Artificial Intelligence (AAAI04)*, pages 562–569, 2004.
- [3] I. Cenamor, T. De La Rosa, and F. Fernández. IBaCoP and IBaCoP planner. In *In Eighth International Planning Competition Booklet (ICAPS-14)*, pages 35–38, 2014.
- [4] Amanda Jane Coles and Andrew Coles. Lprpg-p: Relaxed plan heuristics for planning with preferences. In *Proceedings of the Twenty-First International Conference on Automated Planning and Scheduling (ICAPS 2011)*, pages 26–33, 2011.
- [5] M.B. Do and S. Kambhampati. Partial satisfaction (over-subscription) planning as heuristic search. In *Proceedings of Fifth International Conference on Knowledge Based Computer Systems (KBCS04)*, page mancanti, 2004.
- [6] A. Gerevini, P. Haslum, D. Long, A. Saetti, and Y. Dimopoulos. Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners. *Artificial Intelligence*, 173(5-6):619–668, 2009.
- [7] M. Katz and J. Hoffmann. Mercury planner: Pushing the limits of partial delete relaxation. In *In Eighth International Planning Competition Booklet (ICAPS-14)*, pages 43–47, 2014.
- [8] S. Núñez, D. Borrajo, and C. Linares López. MIPlan and DPMPlan. In *In Eighth International Planning Competition Booklet (ICAPS-14)*, pages 13–16, 2014.
- [9] F. Percassi, A. Gerevini, and H. Geffner. Improving plan quality through heuristics for guiding and pruning the search: A study using LAMA. In *Proceedings of the Tenth International Symposium on Combinatorial Search (SoCS 2017)*, pages 144–148, 2017.
- [10] S. Richter and M. Westphal. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research*, 39(1), 2010.
- [11] M. Vallati, L. Chrapa, M. Grześ, T. L. McCluskey, M. Roberts, and S. Sanner. The 2014 international planning competition: Progress and trends. *AI Magazine*, 36(3):90–98, 2015.
- [12] Mattmiller R. Wright B. and Nebel B.. Compiling away soft trajectory constraints in planning. In *In Proceedings of the Sixteenth Conference on Principles of Knowledge Representation and Reasoning (KR18)*, 2018.