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), 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.

For instance, a useful class of preferences than can be expressed in PDDL3 [6] consists of *always preferences*, requiring that a certain condition should hold in *every* state reached by a plan. As discussed in [**?** **?** 6**?**], adding always preferences to the problem model can be very useful to express safety or maintenance conditions, and other desired plan properties. An simple example of such conditions is "whenever a building surveillance robot is outside a room, all the room doors should be closed".

PDDL3 supports other useful types of preferences, and in particular the qualitative preferences of types *at-end*, which are which are equivalent to soft goals, *sometime*, *sometime-before* and *at-most-once*, which are all the types used in the available benchmarks for planning with qualitative PDDL3 preferences [6]. Examples of preferences that can be expressed through these constructs in a logistics domain are: "sometime during the plan the fuel in the tank of every vehicle should be full", "a certain depots should be visited before another once", "every store should be visited at most once" (the reader can find additional examples in [6]).

In this paper, we study propositional planning with these types of preferences through a compilation approach.

Related Work

Our compilative approach is 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 using two actions in mutual exclusion. The first one, which is called *collect(p)*, has cost equal to 0 and requires that p be true when it is applied; the second one, whic is called *forgo(p)*, has cost equal to the utlity of p and requires that p be false when it is applied.

Both of these action can be performed at the end of the plan and for each soft goal p but just one of $\{$*collect(p), forgo(p)*$\}$ can appear in the plan depending on whether the soft goal has been achived or not. This scheme has achieved good perfomance which can be improved with the use of an ad hoc admissible heuristic based on the reachability of soft goals [9].

The most prominent existing planners supporting PDDL3 preferences are HPlan-P [**?** **?**], which won the "qualitative preference" track of IPC-5, MIPS-XXL [**?** **?**] and the more recent LPRPG-P [4] and its extension in [**?**]. These (forward) 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 [**?**]. An implementation of Crosswell and Coddington's approach is unavailable, but Bayer and McIlaraith [**?**] observed that their approach suffers exponential blow up problems and performs less efficiently than the approach of HPlan-P.

IMPORTANTE, CONFRONTO LAVORO NEBEL: An important work about soft-trajectory constraints compilation, which is closely related to ours, has been recently proposed by Wright, Mattüller and Nebel in [12] (cerca riferimento articolo Nebel). This approach is a based on the compilation of the soft trajectory constraints into conditional effects and state dependent action costs using LTL_f and Büchi automata. There are some similarities between between our and their approach but at the same time there are also differences.

FP: differenze approccio Nebel e nostro: 1) numero di fluenti introdotto nella compilazione diverso; 2) il nostro e' un approccio piu' specifico PDDL3-oriented, il loro piu' generale; 3) diverso meccanismo di aggiornamento dei costi, nel loro caso ricompensa e penalita', nel nosto solo penalit 4) il loro schema prevede quindi costi negativi e postivi, il nostro positivi; nel loro caso per avere solo costi positivi e' necessario perdere l'ottimalita' nel nostro caso no ed inoltre volendo potremmo introdurre dei costi negativi per quelle preferenze la cui violazione puo' essere testata solo alla fine del piano mantenendo l'ottimalita'; ho fatto inoltre menzione del fatto che i costi possono essere incrementanti il prima possibile (es. always)

1) In our approach, given a preference P, we introduce at most a pair of boolean fluents, typically one additional fluent to represent if a preference is violated or not (P-*violated*) and in some cases an additional fluent to correctly represent the status of a preference during the planning, while in their approach it is introduced a boolean variable for each state of the corresponding automaton of P. **2)** Their approach is more general while ours is focused on PDDL3 constraints and therefore it is more specific.

3) In their work the cost of the plan during the planning is updated by using rewards and penalties (negative and positive costs respectively). The cost of the plan is increased whenever a violation of a preference (also reversible) occurs. On the contrary, if an operator causes the satisfaction of a preference then the cost of the plan is decreased. In both cases the negative or positive cost is equal to the utility of the interested preference[2]. **4)** This type of cost update requires the use of negative costs while our compilation scheme produces a problem whose costs are monotonically increasing because, similarly to what was proposed in Keyder and Geffner, costs are realized only at the end of the planning. In our scheme some costs can be anticipated for those preferences whose violation is irreversible (e.g. always, sometime-before) and negative costs could be used for those preferences that may be "temporarily" violated (e.g. sometime, at-end). In both cases our scheme would maintain the optimality but in this work we have provided a compilation based only on positive costs in order to take advantage of a wider spectrum of classical planners (few planners still support negative costs).

we propose a compilation scheme for translating a STRIPS problem with PDDL3 qualitative preferences into an equivalent STRIPS+ problem. Handling action costs is a practically important, basic functionality that is supported by many powerful planners; the proposed compilation method allows them to immediately support (through the compiled problems) the considered class of preferences with no change to their algorithms and code.

Preliminaries

Planning with Qualitative Preferences

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 $\langle \mathit{Pre}(o), \mathit{Eff}(o)\rangle$, where $\mathit{Pre}(o)$ is a sets of atomic formulae over F and $\mathit{Eff}(o)$ is a set of literals over F. $\mathit{Eff}(o)^+$ denotes the set of positive literals in $\mathit{Eff}(o)$, $\mathit{Eff}(o)^-$ the set of negative literals in $\mathit{Eff}(o)$. An action sequence $\pi = \langle a_0, \ldots, a_m\rangle$ is applicable in a planning problem Π if all actions a_i are in O and there exists a sequence of states $\langle s_0, \ldots, s_{m+1}\rangle$ such that $s_0 = I$, $prec(a_i) \subseteq s_i$ and $s_{i+1} = s_i \setminus \{p \mid \neg p \in \mathit{Eff}(a_i)^-\} \cup \mathit{Eff}(a_i)^+$, for $i = 0 \ldots m$. 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 ith action a_i in π and $|\pi|$ is the length of π.

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

[2] Messo in footnote altrimenti era troppo lungo: Note that both the violation and the satisfaction of a preference may be temporary conditions depending on the type of interested preference. For example an always preference can be irreversibly violated and its satisfaction can only be evaluated at the end of the plan considering the whole trajectory of the states; on the contrary a sometime-after preference can be satisfied and violated several times during the execution of the plan.

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

Figure 1: Semantics of the basic modal operators in PDDL3

PDDL3 [] 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 $\langle s_0, s_1, ..., s_n\rangle$ 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., $\phi = \phi_1 \wedge \phi_2 \wedge ... \wedge \phi_n$ 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: $\mathcal{A}$, $\mathcal{SB}$, $\mathcal{ST}$, $\mathcal{AO}$, $\mathcal{G}$ denote the classes of qualitative preferences of type always, sometime-before, sometime, at-most-once and soft goal, respectively, for a given planning problem; A_ϕ, $\mathsf{SB}_{\phi,\psi}$, ST_ϕ, AO_ϕ, G_ϕ denote a particular preference over $\mathcal{A}$, $\mathcal{SB}$, $\mathcal{ST}$, $\mathcal{AO}$, $\mathcal{G}$, respectively, involving formulae ϕ and ψ; moreover, if a plan π satisfies a preference P, we write $\pi \models P$. [CHECK: NOTAZIONE MODIFICATA: $\mathcal{G}$ invece di $\mathcal{SG}$]

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

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

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

Definition 2. *Let* Π *be a* STRIPS+P *problem with preferences* $\mathscr{P}$*. 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:* $u(\pi) = \sum_{P \in \mathscr{P} : \pi \models P} u(P) - c(\pi)$.

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. [AG: CHECK!!!]

Additional Notation and Definitions

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

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

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

Mote 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* $\phi = \phi_1 \wedge ... \wedge \phi_n$*, 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 $\phi_i = l_1 \vee ... \vee l_{m_i}$ 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* $\phi = \phi_1 \wedge ... \wedge \phi_n$*, 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* $C_\phi(o)$, $\overline{L}(\phi_i) \not\subseteq Z(o)$.

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* $\phi = \phi_1 \wedge ... \wedge \phi_n$*, we say that o* ***can make*** ϕ ***false*** *(dependently from s) if*

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

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 bender 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.

[AG: CHECK. VIOLATION MODIFICATO IN VIOLATOR]

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) $\overline{L}(\phi_i) \subseteq Z(o)$, and (2) $\overline{L}(\phi_i) \not\subseteq Pre(o)$.*

The set of always preferences that are violated by an operator o is denoted with $V_{\mathcal{A}}(o)$.

Operator o is a threat for 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). [CHECK FRASE! RISCRITTO.] 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 always preferences threatened by o is denoted with $T_{\mathcal{A}}(o)$; the set of clauses of a preference A_ϕ threatened by o is denoted with $TC_{\mathcal{A}}(o, \phi)$.

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 $|\overline{L}(\phi_i) \cap Z(o)| = 0$, or*
2. *there exists a clause ϕ_i of ϕ such that $\overline{L}(\phi_i) \subseteq Pre(o)$.*

Example. An operator $o = \langle \neg a, \neg b \rangle$ is a threat for A_{ϕ_1}, a violator for 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 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.*

The set of sometime preferences of Π which are potentially supported by the operator o are denoted with $S_{\mathcal{ST}}(o)$.

Example. An operator $o = \langle \top, \neg b \rangle$ is a potential support for 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 $\mathsf{SB}_{\phi,\psi}$ is violated if ϕ becames 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 $\mathsf{SB}_{\phi,\psi}$. Depending by the state where it is applied such operator could behave as a actual support or as a neutral operators for $\mathsf{SB}_{\phi,\psi}$.

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

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

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

Definition 11. *Given an operator o and a sometime-before preference $\mathsf{SB}_{\phi,\psi}$ 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 $T_{\mathcal{SB}}(o)$.

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

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

Example. An operator $o = \langle \top, b \rangle$ is a potential support for $\mathsf{SB}_{\phi_1,\psi_1}$, a threat for $\mathsf{SB}_{\phi_2,\psi_2}$ and neutral for $\mathsf{SB}_{\phi_3,\psi_3}$ 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 becames 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 AO_ϕ of a* STRIPS+P *problem, o is a **threat** of P if o can make true ϕ, otherwise o is **neutral** for AO_ϕ..*

The set of at-most-once preferences of Π which are threatened by the operator o are denoted with $T_{\mathcal{AO}}(o)$.

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

Operators Affecting Sometime-After Preferences

A sometime-after preference $\mathsf{SA}_{\phi,\psi}$ is violated if ϕ 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 is a soft threat of $\mathsf{SA}_{\phi,\psi}$.

Definition 14. *Given an operator o and a sometime-after preference $\mathsf{SA}_{\phi,\psi}$ of a* STRIPS+P *problem, o is a **threat** on ϕ operator of $\mathsf{SA}_{\phi,\psi}$ if o can make true ϕ.*

There is another way to threat a sometime-after preference. If a state s' generated by o can falsify ψ then $\mathsf{SA}_{\phi,\psi}$ could be violated if ψ remains true in s'.

Definition 15. *Given an operator o and a sometime-after preference $\mathsf{SA}_{\phi,\psi}$ of a* STRIPS+P *problem, o is a **threat** on ψ operator of $\mathsf{SA}_{\phi,\psi}$ if o can make ψ false.*

The set of sometime-after preferences of Π which are threatened by the operator o, indifferently on ϕ or ψ, are denoted with $T_{\mathcal{SA}}(o)$.

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

Definition 16. *Given an operator o and a sometime-after preference $\mathsf{SA}_{\phi,\psi}$ of a* STRIPS+P *problem, o is a* ***potential support*** *of $\mathsf{SA}_{\phi,\psi}$ 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 $S_{\mathcal{SA}}(o)$.

Definition 17. *Given an operator o and a sometime-after preference $\mathsf{SA}_{\phi,\psi}$ of a* STRIPS+P *problem, o is an* ***updating*** *of $\mathsf{SA}_{\phi,\psi}$ if o can make ψ false.*

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

Example An operator $o = \langle \top, \neg a, b\rangle$ is a potential support of $\mathsf{SA}_{\phi_1,\psi_1}$, a soft threat on ϕ of $\mathsf{SA}_{\phi_2,\psi_2}$, a soft threat on ψ of $\mathsf{SA}_{\phi_2,\psi_2}$, and neutral for $\mathsf{SA}_{\phi_4,\psi_4}$ 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, which are then compiled away obtaining 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.

AG: SPOSTARE NEL RELATED WORK: *The scheme proposed by Keyder and Geffner is considerable simpler than ours because it does not to consider the interaction between actions and preferences such as threats, supports and violators.*

For a STRIPS+P problem $\Pi = \langle F, I, O, G, \mathscr{P}, c, u\rangle$, 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*$\}$;
- $I' = I \cup \overline{C'} \cup V_{\mathcal{ST}} \cup S_{\mathcal{AO}} \cup \{$*normal-mode*$\}$;
- $G' = G \cup C'$;
- $O' = \{$*collect*(P),*forgo*$(P) \mid P \in \mathscr{P}\} \cup \{end\} \cup \{comp(o, \mathscr{P}) \mid o \in O\}$
- *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*$(o, \mathscr{P})$ is the function translating operator o according to Definition 18;
- $c'(o') = \begin{cases} u(P) & \text{if } o' = \textit{forgo}(P) \\ c(o') & \text{if } o' = \textit{comp}(o, \mathscr{P}) \\ 0 & \text{otherwise;} \end{cases}$
- $V = \{P$*-violated* $\mid P \in \mathscr{P}\}$;
- $V_{\mathcal{ST}} = \{P_i$*-violated* $\mid P_i \in \mathscr{P}_{\mathcal{ST}}|\}$;
- $S_{\mathcal{AO}} = \{P_i$-seen $\mid P_i \in \mathscr{P}_{\mathcal{AO}}(I)\}$, with $\mathscr{P}_{\mathcal{AO}}(I) = \{P_i = \mathsf{AO}_\phi \mid P_i \in \mathscr{P}_{\mathcal{AO}}, I \models \phi\}$;
- $C' = \{P' \mid P \in \mathscr{P}\}$ and $\overline{C'} = \{\overline{P'} \mid P \in \mathscr{P}\}$;

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.

[CHECK! NUOVO PARAGRAFO COMPATTO] The P_i*-violated* literals in the compiled initial state I' are used to consider any $\mathcal{ST}$ 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 $\mathcal{AO}$ preference when an operator makes the preference formula true for the second time in the state trajectory.

Function *comp*$(o, \mathscr{P})$ 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*$(o, \mathscr{P})$ 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 18. *Given an operator o the corresponding compiled operator is defined using the following function:*

$$\textit{Pre}(o') = \textit{Pre}(o) \cup \{\textit{normal-mode}\}$$
$$\textit{Eff}(o') = \textit{Eff}(o) \cup \bigcup_{P_i \in P_{\textit{affected}}(o)} \mathcal{W}(o, P_i)$$

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

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

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

where $cond(o,P) = \bigvee_{\phi_i \in TC_{\mathcal{A}}(o,P)}(l_1 \wedge \ldots \wedge l_q)$ *and* $\{l_1,\ldots,l_q\} = \overline{AA}(o)_{\phi_i}$.

For each affected preference $P = \mathsf{A}_\phi$, 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 $cond(o, \mathsf{A}_\phi) = \overline{AA}(o)_{c \vee d \vee e} = \{\neg d, \neg e\}$.

Conditional Effects for $\mathcal{ST}$ Preferences

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

Definition 20. *Given a preference $P = \mathsf{ST}_\phi$ and an operator o that potentially supports it, the conditional effect set in the compiled version o' of o (according to Definition 18) is:*

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

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

As described above, for each $P = \mathsf{ST}_\phi$ *P-violated* holds in the compiled initial state, and a potential support o of P makes ϕ true when all clauses $\phi_i \notin C_\phi(o)$ 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 $\mathcal{SB}$ Preferences

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

Definition 21. *Given a preference $P = \mathsf{SB}_{\phi,\psi}$ and an operator o affecting it, the conditional effect set $\mathcal{W}(o,P)$ in the compiled version o' of o (according to Definition 18) is:*

$$\mathcal{W}(o,P) = \begin{cases} \{\textit{when}\ (\textit{cond}_S(o,P))\ (\textit{seen-}\psi)\} \\ \quad \textit{if } o \textit{ is a potential support for } P \\ \{\textit{when}\ (\textit{cond}_T(o,P))\ (P\textit{-violated})\} \\ \quad \textit{if } o \textit{ is a threat for } P \end{cases}$$

where:

- $\textit{cond}_S(o,P) = \{\psi_i \mid\ \psi_i \notin C_\psi(o)\}$
- $\textit{cond}_T(o,P) = \{\neg\textit{seen-}\psi\} \cup \{\phi_i \mid\ \phi_i \notin C_\phi(o)\}$.

An operator o affecting a preference $P = \mathsf{SB}_{\phi,\psi}$ can behave as a (a) potential support of P, (b) a threat for P, or (c) both. These case are captured by the three conditional effects of o' in Definition 21.

In case (a), if all clauses that are not certainly true in s' (i.e., $\textit{cond}_S(o,P)$) 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., $\textit{cond}_T(o,P)$) 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 AO_ϕ preference are defined as follows.

Definition 22. *Given a preference $P = \mathsf{AO}_\phi$ and an operator o that threats P, the conditional effect set $\mathcal{W}(o,P)$ in the compiled version o' of o (according to Definition 18) is:*

$$\begin{aligned}\mathcal{W}(o,P) = \{&\textit{when}\ (\textit{cond}_N(o,P))\ (\textit{seen-}\phi)) \\ &\textit{when}\ (\textit{cond}_T(o,P)))\ (P\textit{-violated})\}\end{aligned}$$

where:

- $\textit{cond}_N(o,P) = \{\neg\textit{seen-}\phi\} \cup \{\phi_i \mid \phi_i \notin C_\phi(o)\}$
- $\textit{cond}_T(o,P) = \{\textit{seen-}\phi\} \cup \{\phi_i \mid \phi_i \notin C_\phi(o)\} \cup \{\bigvee_{\phi_i \in C_\phi(o)}(\neg l_1 \wedge \ldots \wedge \neg l_q) \mid \{l_1,\ldots,l_q\} = L(\phi_i)\}$.

If an operator o affecting $P = \mathsf{AO}_\phi$ makes ϕ true for the first time in the state trajectory (i.e., $\text{cond}_N(o,P)$ 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 $\text{cond}_T(o,P)$), then o violates P and o' has effect *P-violated*.

Conditional effects for $\mathcal{SA}$ Preferences

The conditional effects for a compiled operator threatening preference SA_ϕ preference are defined as follows.

Definition 23. *Given a preference $P = \mathsf{SA}_\phi$ and an operator o that threats P, the conditional effect set $\mathcal{W}(o,P)$ in the compiled version o' of o (according to Definition 18) is:*

$$\mathcal{W}(o,P) = \begin{cases} \{\textit{when}\ (\textit{cond}_{T_\phi}(o,P))\ (P\textit{-violated})\} \\ \quad \textit{if } o \textit{ is a threat on } \phi \textit{ of } P \\ \{\textit{when}\ (\textit{cond}_{T_\psi}(o,P))\ (P\textit{-violated})\} \\ \quad \textit{if } o \textit{ is a threat on } \psi \textit{ of } P \\ \{\textit{when}\ (\textit{cond}_S(o,P))\ (\psi\textit{-true}, \neg P\textit{-violated})\} \\ \quad \textit{if } o \textit{ is a support of } P \\ \{\textit{when}\ (\textit{cond}_U(o,P))\ (\neg\psi\textit{-true})\} \\ \quad \textit{if } o \textit{ is a support of } P \end{cases}$$

where:

- $\textit{cond}_{T_\phi}(o,P) = \{\phi \mid\ \phi_i \notin C_\phi(o), \psi\textit{-is-true}\}$
- $\textit{cond}_{T_\psi}(o,P) = \{\psi \mid\ \psi_i \notin C_\psi(o)\} \cup \{\bigvee_{\psi_i \in TC(o,P)}(l_1 \wedge \ldots \wedge l_q)\}$ *and* $\{l_1,\ldots,l_q\} = \overline{AA}(o)_{\psi_i}$;
- $\textit{cond}_S(o,P) = \{\psi_i \mid\ \psi_i \notin C_\psi(o)\}$
- $\textit{cond}_U(o,P) = \bigvee_{\phi_i \in TC_{\mathcal{A}}(o,P)}(l_1 \wedge \ldots \wedge l_q)$ *and* $\{l_1,\ldots,l_q\} = \overline{AA}(o)_{\phi_i}$.

An operator o affecting a preference $P = \mathsf{SA}_{\phi,\psi}$ can behave as (a) a threat of P on ϕ, (b) a threat of P on ψ, (c) a potential support or an (d) updating operator of P.

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 [**?** , see proof of Theorem 20], 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, DARE SPIEGAZIONE DI QUALI SONO LE PROPRIETA' STRUTTURALI DEI COND EFFECTS CHE SFRUTTIAMO. 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.

Compilation Properties

Proposition 1 (Correspondence between plans)**.** *For an applicable action sequence π for a Π problem, let π' a compiled plan such that each action o' in π it is built by adding a set of conditional effects to the original operator effects according to Definition 18, then:*

$$\pi \textit{ is a plan for } P \iff \pi' \textit{ is a plan for } P'.$$

Proof. ($\Rightarrow$)
(a) The original initial state I is extendend with some additional predicates such that $I \subseteq I'$. Excluding from the demonstration the *forgo* and *collect* operators for soft goals and sometime preferences, whose correctness has already been demonstrated in [**?**], we can observe that (b) the compiled goal G' is equal to the original goal G. Using Definition 18, if $o \in O_{\text{neutral}}$ then no conditional effects are added by the compilation to the operator since it does not affect any preferences of Π', while if $o \notin O_{\text{neutral}}$ then its effects are extended with a set of conditional effects which only affect the compiled additional fluents, e.g. P-violated (see Definitions xxx-yyy). SoB (c) the compiled operators have the property of not deleting any predicate belonging to F.

The first action of π' is applicable in I' cause (a), the following ones are applicable cause (b) and so π' is valid and $\pi' \models G$ cause (b) and (c). □

Proof. ($\Leftarrow$)

□

Experimental Evaluation

Experiment Settings

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]. 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**?**]: 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 $\text{LAMA}_{\text{P}}(h_{\text{R}})$, exploiting admissibile heuristic h_{R} for testing reachability of soft goals during search [9].

As benchmarks we used all (100) [CHECK!!! SONO 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.

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 (compilation time included [CHECK!!! VERO?]) and 8GB, respectively.

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

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

$$\alpha_{\textit{cost}}(p,i) = \frac{\textit{cost}(p,i)}{\textit{cost}_{\textit{total}}(i)} = \frac{\sum_{P \in \mathscr{P}(i)\,:\,\pi \not\models P} c(P)}{\sum_{P \in \mathscr{P}(i)} c(P)}$$

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

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 preferences, while those in Table 2 concern plan quality when only the specified preferences of a particular class are used for the plan evaluation. 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 $\text{LAMA}_\text{P}(h_\text{R})$ 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 $\text{LAMA}_\text{P}(h_\text{R})$ 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 $\text{LAMA}_\text{P}(h_\text{R})$ 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. Figures **??** and **??** show, 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} restricted 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 to Figure **??** we note that the crucial preferences in this domain are mainly soft goals, which are more satisfied by LPRPG-P. The search pruning technique in $\text{LAMA}_\text{P}(h_\text{R}$ slightly helps LAMA to achieves more soft goals, but not enough to reach the performance of LPRPG-P.

Planner	Rovers	TPP	Trucks	Openstacks	Storage	TOTAL
$\text{LAMA}_\text{B}(h_\text{R})$	16.98	8.34	15.32	19.28	**18.47**	**78.39**
FDRemix	17.89	7.1	**17.67**	18.99	16.2	77.86
FDSS 2018	17.6	7.03	17.08	18.7	17.11	77.52
LAMA(2011)	17.01	7.53	13.04	18.42	17.81	73.82
IBaCoP2	**19.62**	9.68	10.0	17.85	15.72	72.87
LAMA(2018)	16.44	7.63	13.34	16.03	17.78	71.22
LPRPG-P	11.36	**18.74**	6.99	**19.71**	12.87	69.66
MIPlan	17.65	8.8	9.23	17.35	14.42	67.45
Mercury	16.07	6.57	7.78	18.06	14.5	62.97

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.

Optimal Planning Results

Conclusions

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. Núnez, 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

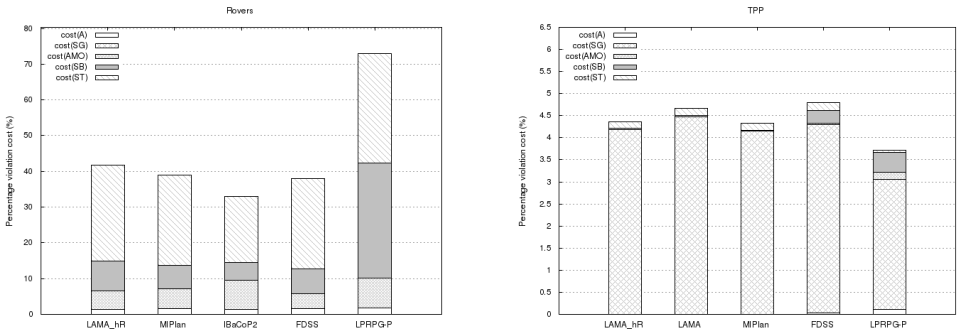

Figure 2: α_{cost} comparison for TPP domain.

$\mathscr{P}_{\mathcal{A}}$						
Planner	Rovers	TPP	Trucks	Openstacks	Storage	TOTAL
$\text{LAMA}_{\text{P}}(h_{\text{R}})$	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

$\mathscr{P}_{\mathcal{SG}}$						
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
$\text{LAMA}_{\text{P}}(h_{\text{R}})$	—	14.82	14.36	18.85	18.92	66.94
MIPlan	—	15.08	9.85	16.84	19.32	61.09

$\mathscr{P}_{\mathcal{AO}}$						
Planner	Rovers	TPP	Trucks	Openstacks	Storage	TOTAL
FDSS 2018	17.22	18.0	20.0	—	19.0	74.22
$\text{LAMA}_{\text{P}}(h_{\text{R}})$	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

$\mathscr{P}_{\mathcal{SB}}$						
Planner	Rovers	TPP	Trucks	Openstacks	Storage	TOTAL
$\text{LAMA}_{\text{P}}(h_{\text{R}})$	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

$\mathscr{P}_{\mathcal{ST}}$						
Planner	Rovers	TPP	Trucks	Openstacks	Storage	TOTAL
$\text{LAMA}_{\text{P}}(h_{\text{R}})$	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.

Domain	h^{blind}		$h^{\max}$		h^{cpdb}	
	WRB	Our	WRB	Our	WRB	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 Nebel compilation scheme (WRB) on the IPC5 benchmarks set with additional instances with random sampled soft-trajectory constraints, A* search for optimal solution. Our results concerning the sampled instances are averaged for each generated instance. The best performance are indicated in bold.

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. Chrpa, 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] Mattmller 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.