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 $\mathsf{LTL_f}$ 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 $\pi = \langle a_0, \dots, a_m\rangle$ 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 $s_0 = I$, $prec(a_i) \subseteq s_i$ and $s_{i+1} = s_i \setminus \{p \mid \neg p \in \mathit{Eff}(a_i)^-\} \cup \mathit{Eff}(a_i)^+$, for $i = 0 \dots m$. 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 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* ϕ ψ)
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* ϕ ψ)
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 NB: FIGURA DA SEMPLIFICARE: TOGLIERE GLI HAPPENINGS .

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 ($i = 1 \ldots n$) 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 $\mathcal{G}$]

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* $u : \mathscr{P} \rightarrow \mathbb{R}_0^+$.

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:*

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

[AG: CHECK!!! CAMBIARE NOTAZIONE PERCHE' ABBIAMO TOLTO $\overline{C}_\phi(o)$ e DICIAMO o MAKES CERTAINLY TRUE] —— FATTO

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, $j \in \{1 \ldots m_i\}$ 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'.

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 *violatior*, 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 5. *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 6. *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 violatior and there exists a clause* ϕ_i *of* ϕ *such that:*

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

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 7. *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 8. *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 9. *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 10. *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 11. *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 12. *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 $\phi_1 = c \vee \neg b$.

Compilation of Qualitative Preferences

Before describing the general compilation scheme, some further definitions should be provided.

Definition 13. *Given a* STRIPS+P *problem* Π *if an operator o of* Π *is neutral for every given preference of* Π *over* $\mathcal{A}$*,* $\mathcal{SB}$*,* $\mathcal{ST}$*,* $\mathcal{AO}$ *and* $\mathcal{G}$ *then we say that o is* ***neutral*** *for* Π*. The set of all the neutral operator for* Π *is denoted by* $O_{neutral}$*.*

Definition 14. *Given an operator o of a* STRIPS+P Π *problem the set* $P_{affected(o)}$ *of prefereces affected by o is defined as:*

$$P_{affected(o)} = T_{\mathcal{A}}(o) \cup T_{\mathcal{SB}}(o) \cup S_{\mathcal{SB}}(o) \cup T_{\mathcal{AO}}(o) \cup S_{\mathcal{ST}}(o).$$

Given a STRIPS+P problem, an equivalent STRIPS+ problem can be derived by translation which has some similarities to what proposed by Keyder and Geffner for soft goals but also significant difference. 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. In order to simplify the compilation scheme we don't consider the compilation of soft goals because it can be easily added using the same method of Keyder and Geffner.

Moreover we assume that every always and sometime-before preference are not violated and every sometime preference are not satisfied in the problem initial state I. Before starting the compilation, we carry out the following checks in order to exclude some preferences from the process:

- for each $P = \mathsf{A}_\phi \in \mathscr{P}_\mathcal{A}$ we check that $I \models \phi$, if the condition does not hold we exclude P from the compilation increasing the cost of the plan by $u(P)$ because it is already violated in I;
- for each $P = \mathsf{SB}_{\phi,\psi} \in \mathscr{P}_\mathcal{SB}$ we check that $I \not\models \phi$, if the condition does not hold we exclude P from the compilation increasing the cost of the plan by $u(P)$ because it is already violated in I; after that we check that $I \not\models \phi \wedge I \models \psi$, if the condition hold we exclude P because it is already satisfied in I;
- for each $P = \mathsf{ST}_\phi \in \mathscr{P}_\mathcal{ST}$ we check that $I \models \phi$, if the conditiom holds we exclude P because it is already satisfied in I;

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

- $F' = F \cup V \cup D \cup C \cup \overline{C'} \cup \{\textit{normal-mode}, \textit{end-mode}\}$;
- $I' = I \cup \overline{C'} \cup V_\mathcal{ST} \cup S_\mathcal{AO} \cup \{\textit{normal-mode}\}$;
- $G' = G \cup C'$;
- $O' = \{\textit{collect}(P_i), \textit{forgo}(P_i) \mid P_i \in \mathscr{P}\} \cup \{\textit{end}\} \cup \{\textit{comp}(o, \mathscr{P}) \mid o \in O\}$
- $c'(o') = \begin{cases} u(P) & \text{if } o' = \textit{forgo}(P) \\ c(o') & \text{if } o' = \textit{comp}(o, \mathscr{P}) \\ 0 & \text{otherwise} \end{cases}$

where:

- $V = \cup_{i=1}^{|\mathscr{P}|}\{P_i\text{-}\textit{violated}\}$;
- $V_\mathcal{ST} = \cup_{i=1}^{|\mathscr{P}_\mathcal{ST}|}\{P_i\text{-}\textit{violated}\}$, $\mathscr{P}_\mathcal{ST} \subseteq \mathscr{P}$; this is the set of the violator predicates restricted to the subset of the sometime preferences of all preferences $\mathscr{P}$;
- $S_\mathcal{AO} = \cup_{i=1}^{|\mathscr{P}_\mathcal{AO}(I)|}\{P_i\text{-}\textit{seen}\}$, $\mathscr{P}_\mathcal{AO}(I) = \{P_i = \mathsf{AO}_\phi \mid P_i \in \mathscr{P}_\mathcal{AO}, I \models \phi\}$; this is the set of *seen* predicates (see Section) for those at-most-once preferences such that $I \models \phi$;
- $C' = \{P' \mid P \in \mathscr{P}\}$ and $\overline{C'} = \{\overline{P'} \mid P \in \mathscr{P}\}$;
- $\textit{forgo}(P_i) = \langle\{\textit{end-mode}, P_i\text{-}\textit{violated}, \overline{P_i'}\}, \{P', \neg\overline{P'}\}\rangle$;
- $\textit{end} = \langle\{\textit{normal-mode}\}, \{\textit{end-mode}, \textit{normal-mode}\}\rangle$;
- $\textit{comp}(o, \mathscr{P})$ which is the function which transforms an operator o into the corresponding compiled one, see Definition 15;

Forgo and Collect Actions For each preference P the transformation of Π into Π' adds a dummy hard goal P' to Π' which can be achieved by two ways: with action $\textit{collect}(P)$, that has cost 0 but requires P to be satisfied (i.e. P-*violated* is false in the goal state for all kinds of preferences except for sometime), or with action $\textit{forgo}(P)$, 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 preference, exactly one of $\textit{collect}(P)$ and $\textit{forgo}(P)$ appears in the plan.

Operator Compilation Function The function $\textit{comp}(o, \mathscr{P})$ which transforms an original operator o into the equivalent compiled one is splitted in two parts. If the operator is neutral ($o \in O_{neutral}$) then the function just extends *Pre*(o) with the predicate *normal-mode* in order to scorporate the execution of the domain operators and the evaluationof *forgo* and *collect* actions at the end of the planning.

If the operator o is not neutral ($o \in O - O_{neutral}$) where $|P_{\textit{affected}}(o)| = n$, then the compilation function $\textit{comp}(o, \mathscr{P})$ extends its effects, for each affected preference P_i, by adding a set of conditional effects denoted with $\mathcal{W}(o, P_i)$ whose definition depends by o, the class of preference P_i belongs to and the way how o interacts with P_i. In Subsections – we will detail the definition of $\mathcal{W}(o, P_i)$ for each class of preference.

But we want a compiling problem which belongs to STRIPS+ class and so we have to compile away the conditional effects (see Section).

Extension of the initial state Note that the original initial state I is extended to I' with the set of literals $V_\mathcal{ST}$, which contains the literal P_i-*violated* for each sometime preference of the problem. The literal P_i-*violated* states that the related preference P_i is (temporarily) violated in I until a support operator for P is applied.

Furthermore the original initial state I is extended with the set of literals $S_\mathcal{AO}$ which contains, for each at-most-once preference $P_i = \mathsf{AO}_{\phi_i}$ of the problem such that $I \models \phi_i$, the literal *seen*-P_i. This additional fluent *seen*-P_i states that the formula ϕ_i, involved in P_i, is satisfied in I. This precaution is necessary to correctly capture any possible violators of at-most-once preferences.

Definition 15. *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 $o \in O_{\textit{neutral}}$ then $\textit{Eff}(o') = \textit{Eff}(o)$.

Before defining the extending effects used to compile an operator o which affects an always preference, we introduce some useful notation in order to simplify the formalisation. For an operator o and a preference clause ϕ_i:

- $NA(o)_{\phi_i} = \{l_j \in L(\phi_i) \mid \neg l_j \in (\mathit{Eff}(o)^+ \cup \mathit{Eff}(o)^-)\}$ is the set of literals in $L(\phi_i)$ falsified by the effects of o;
- $AA(o)_{\phi_i} = L(\phi_i) \setminus NA(o)_{\phi_i}$ is the set of literals in $L(\phi_i)$ *not* falsified by the effects of o;
- $\overline{AA}(o)_{\phi_i}$ is the literal-complement set of $AA(o)_{\phi_i}$.

Compilation of an Always Preference

We now present the transformation of operators that threaten or violate a preference in class $\mathcal{A}$.

[FP: compilazione delle preferenze always con nuova notazione]

Definition 16. *Given an always preference A_ϕ and an operator o which affects P, the conditional effect set $\mathcal{W}(o, \mathsf{A}_\phi)$ in the compiled version o' of o (according to Definition 15) is defined as:*

$$\mathcal{W}(o, \mathsf{A}_\phi) = \begin{cases} \{\mathit{when}\ (\mathit{cond}(o, \mathsf{A}_\phi))\ (\mathsf{A}_\phi\text{-}\mathit{violated})\} \\ \quad \textit{if o is a threat for } \mathsf{A}_\phi \\ \{\mathit{when}\ (\top)\ (\mathsf{A}_\phi)\} \\ \quad \textit{if o is a violators for } \mathsf{A}_\phi \end{cases}$$

where:

- $\mathit{cond}(o, \mathsf{A}_\phi) = \bigvee_{\phi_i \in TC_{\mathcal{A}}(o, \mathsf{A}_\phi)} (l_1 \wedge \ldots \wedge l_q),\ \{l_1, \ldots, l_q\} = \overline{AA}(o)_{\phi_i}$.

For each always preference affected by an operator o the compiled operator o' contains a conditional effect whose effect is P-*violated*, while the conditions depends on how o affects A_ϕ. If o is a violator for A_ϕ, then the condition is always true. Instead, if o is a *threat* of A_ϕ, the condition which checks that exists at least a clause that will be certainly false in s'. This condition exists (subsists?) if there is at least a threatened clause whose literals which are not falsified in s' are false in the state s where o is applied.

Example. Consider the following operator $o = \langle \top, \{a, \neg c\}\rangle$ which is a threat of a preference A_ϕ where $\phi = \phi_1 \wedge \phi_2 = (a \vee b) \wedge (c \vee d \vee e)$. According to Definition 16, we have to define a condition $\mathit{cond}(o, \mathsf{A}_\phi))$ which has to to check that at least a threatened clause will be certainly false in s'. In this case we have a single threatened clause and we have to build a condition to checks it those literal which are not falsified in s', i.e. $\overline{AA}(o)_{\phi_2} = \{d, e\}$, are already false in s. So we have that the condition which captures the violation of A_ϕ $\mathit{cond}(o, \mathsf{A}_\phi) = \{\neg d, \neg e\}$.

Compilation of a Sometime Preference

We now present the transformation of an operator that is a potential supports of a preference P in class ST.

Definition 17. *Given a sometime preference $P = \mathsf{ST}_\phi$ and an operator o which potentially supports P, the conditional effect set $\mathcal{W}(o, P)$ in the compiled version o' of o (according to Definition 15) is defined as:*

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

where:

- $\mathit{cond}_{\mathcal{S}}(o, P) = \{\phi_i \mid\ \phi_i \notin C_\phi(o)\}$.

As described above, the general compilation scheme adds for each sometime preference $P = \mathsf{ST}_\phi$ a predicate P-*violated* to the compiled initial state I'.

These new violation predicates could be falsified by the potential supports, which are operators whose effects could make ϕ true in s'.

A potential support o could make a formula ϕ true when there exist at least a clause of ϕ that will surely be true in s'. So, if all clauses $\phi_i \notin C_\phi(o)$, where $C_\phi(o)$ is the set of clauses of ϕ that will be certainly true in s', hold in s, then ϕ will be true in s', thus falsifying the predicate P-*violated*.

Compilation of a Sometime-before Preference

We now present the transformation of operators that potentially support or threat a preference in $\mathcal{SB}$.

Definition 18. *Given a sometime-before preference $P = \mathsf{SB}_{\phi,\psi}$ and an operator o which affects P, the conditional effect set $\mathcal{W}(o, P)$ in the compiled version o' of o (according to Definition 15) is defined as:*

$$\mathcal{W}(o, P) = \begin{cases} \{\mathit{when}\ (\mathit{cond}_{\mathcal{S}}(o, P))\ (\mathit{seen}\text{-}\psi)\} \\ \quad \textit{if o is a potential support for P} \\ \{\mathit{when}\ (\mathit{cond}_{\mathcal{T}}(o, P))\ (P\text{-}\mathit{violated})\} \\ \quad \textit{if o is a threat for P} \\ \{\mathit{when}\ (\mathit{cond}_{\mathcal{S}}(o, P))\ (\mathit{seen}\text{-}\psi), \\ \ \mathit{when}\ (\mathit{cond}_{\mathcal{S}}(o, P))\ (P\text{-}\mathit{violated})\} \\ \quad \textit{if o is a both a threat and support for P} \end{cases}$$

where:

- $\mathit{cond}_{\mathcal{S}}(o, P) = \{\psi_i \mid \psi_i \notin C_\psi(o)\}$
- $\mathit{cond}_{\mathcal{T}}(o, P) = \{\neg \mathit{seen}\text{-}\psi\} \cup \{\phi_i \mid \phi_i \notin C_\phi(o)\}$.

The definition of the effects used to extend the effect of an operator o which affects a sometime-before preference P in Definition 18 depends on the class of operators which o belongs. We have to distinguish if o is a potential support, a threat or both for P.

If o is a potential support of P then this operator can behave in two different ways when it is executed. Recalling Definition 4, an operator o could make true a formula ϕ when there exists some clauses of ϕ that will surely be true in the state resulting from the application of o. So, if all $\psi_i \notin C_\psi(o)$ hold in s , i.e. $\mathit{cond}_{\mathcal{S}}(o, P)$ holds in s, then ψ will be true in s' and then we have to keep track of this fact by making the predicate $\mathit{seen}\text{-}\psi$ true.

The compilation if o is threat of P is similar. A threat for $\mathsf{SB}_{\phi,\psi}$ behaves as violators in the case that the operator makes ϕ true in s' (i.e., when all $\phi_i \notin C_\phi(o)$ holds in s, which is the condition of $\mathit{cond}_{\mathcal{T}(o,P)}$), and that the ψ has never been made true in the states preceeding s. If both these conditions, specified in $\mathit{cond}_{\mathcal{T}}(o, P)$, hold in s then predicate $P-\mathit{violated}$ is included in the effects of o'.

We have also to consider the case in which an operator o is both a threat and a support for P. In this case o can behaves in the following ways: making ϕ true in s', making ψ true in s' and making both ϕ and ψ true in s'. In order to handle these situations, the compiled operator o' contains both conditional effects of o as threat of P and as support of P. Note that this correctly captures the violators of P determined by ϕ and ψ becoming simultaneously true by execution of o.

Compilation of an At-Most-Once Preference

We now present the transformation of an operator that threats a preference in $\mathcal{AO}$.

Definition 19. *Given an at-most-once preference $P = \mathsf{AO}_\phi$ and an operator o which affects P, the conditional effect set $\mathcal{W}(o, P)$ in the compiled version o' of o (according to Definition 15) is defined as:*

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

where:

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

The semantic of a preference in class $\mathcal{AO}$ (Figure 1) requires that a preference $P = \mathsf{AO}_\phi$ is satisfied by the state-trajectory generated by a plan π if ϕ *becomes* true in a state s' at most once during the execution of the plan. A formula ϕ becomes true in a state s' due to the execution of an operator o applied in a state s iif $s \models \neg\phi$ and $s' \models \phi$.

If ϕ holds in the problem initial state I, then it is required that either it stay true until the end of the plan. or it becomes false at some successor state of I in the state-trajectory generated by the plan and it then stays always false.

If an operator o can make ϕ true for the first time in the plan trajectory of a plan, then it behaves as a neutral operator for P. On the other hand, if o can make ϕ true after having been true and become false in past states of the trajectory, then o behaves as a threat for P.

In order to correctly capture the possible violators of P, the set of effects extending o $\mathcal{W}(o, P)$ has two conditional effects. The first one is a neutral effect that is used to catch the behavior of o as neutral operator for P. Condition $\mathrm{cond}_N(o, P)$ requires that ϕ has been never changed truth value from true to false in preceeding state using the negated predicate $\neg$*seen-*ϕ and that ϕ will be true in s' using the condition $\{\phi_i \mid \phi_i \notin C_\phi(o)\}$ similarly to what done in the compilation schemes previously presented. If the conditions specified in $\mathrm{cond}_N(o, P)$ hold in the state s where o is applied, then we take into account that ϕ becomes true in s' for the first time stating in the effects of the neutral conditional effect predicate *seen-*ϕ.

The second conditional effect is a violating effect that is used to catch the the behavior of of o as a threat for P. This happens when the following conditions, specified in $\mathrm{cond}_{T(o,P)}$ hold: (specified in $\mathrm{cond}_{\mathcal{V}(o,P)}$):

- ϕ has alreay been made true in a state preceding s; this is expressed by the predicate *seen-*ϕ when o is applied;
- ϕ is made true in the resulting state s'; this is guaranteed by the conditions $\{\phi_i \mid \phi_i \notin C_\phi(o)\}$;
- ϕ is false in the state s where o is applied; this is specified by requiring that at least a clause in $C_\phi(o)$ is false in s.

If all these conditions hold in the state where o is applied then P will be violated in the state resulting from the application of o.

Compilation of Conditional Effects

As described before, given an operator o of a STRIPS+P problem which affects a set of n preferences, the corresponding compiled operator should have in its effects a set $\{(\textit{when } c_i\ \ e_i) \mid i = 1 \ldots m\}$ of $m \leq 2n$ conditional effects, which are built using the compilation schema described in the previous subsections.

In order to keep the compiled problem in the STRIPS+ class, the conditional effects of o' should be compiled away by replacing o' with an equivalent set of operators without conditional effects. In the literature, there are two main general methods for generating this equivalent set of unconditional operators.

The first method, introduced by Gazen and Knoblock [**?**], works by recursively splitting o' for each conditional effect *(when* c_i e_i*)* into a couple of new operator, o'' and $\overline{o}''$, such that:

$$\begin{aligned} &pre(o'') = pre(o') \cup \{c_i\} \\ &\mathit{eff}(o'') = \mathit{eff}(o') \cup e_i \\ &pre(\overline{o}'') = pre(o') \cup \{\neg c_i\} \\ &\mathit{eff}(\overline{o}'') = \mathit{eff}(o'). \end{aligned}$$

This method is not practicable because it leads to an exponential blow up of operators (in our case $O(2^m)$ for each operator affecting n preferences), but the compiled plan preserves exactly the length of the original plan.

The second method, proposed by Nebel [**?** , see proof of Theorem 20], generates a polynomial number of new operators, but it increases polynomially the plan length. The main idea is to simulate the parallel behaviour of the conditional effects of an operator by a replacing it with an equivalent sequence of unconditional operators. For each operator o' with m conditional effects, Nebel's schema introduces m pairs of new operators, that separately evaluate the condition c_i of each conditional effect *(when* c_i e_i*)* and possibly "activates" the corresponding effect e_i. One of the operators in the pair for *(when* c_i e_i*)* contains precondition c_i and an effect indicating that e_i is activated, while the other, that is mutex with the first, contains precondition $\neg c_i$ and does not activate e_i. In order to avoid possible (positive or negative) interference in the sequentialisation of the conditional effects through the new operators (e.g., if *(when* c_1 e_1*)* and *(when* c_2 e_2*)* are conditional effects of o' and $e_1 \models c_2$), the activated effects in the operator sequence are not made immediately true, but they are deferred to the end of the sequence (i.e., after all conditional effects conditions have been evaluated). This is done by using an additional set of operators, called "copying operators" in [?], which copy the activated effects to the state description after all operators in the sequence have been executed (For more details the reader is referred to [?].)

In order to deal with the conditional effects generated by our compilation of PDDL3 preferences, we have implemented and used Nebel's compilation method because a compiled operator can in principle contain many conditional effects, which makes Gazen and Knoblock's method impractical given its exponential complexity. Moreover, the conditional effects needed to compile PDDL3 preference have a particular structure that allows us to simplify and optimize Nebel's general method. In particular we would like avoid the so-called "copying operators" operators maintaining the semantics of conditional effects (which requires that all conditions are evaluated at the same time). First of all, note that the conditional effects that refer to different affected preferences can not interfere with each other because they involve different fluents.

After that we have to pay attention to those class of preferences whose compilation introduces more than one conditional effect, because, affecting the same fluents, they can generate interference. In our previous discussion there are two classes of preferences having this feature, i.e. sometime-before and at-most-once preferences (see Subsections and). In the first case, the set of problematic conditional effects refers to those operators which threats and supports a sometime-before preference at the same time, in the second case to those operators that threats at-most-once preference.

Concerning at-most-once preferences, an interference could arise through the predicate $seen\text{-}\phi$, that is both an effect of the first conditional effect and a condition of the second. However, the conditional effect interference disappears, if in the compilation, the pair of unconditional operators for $(\mathit{when}\ (\mathit{cond}_{\mathcal{V}}(o,P))\ (P{-}\mathit{violated}))$ is constrained to be ordered before the other pair. If we evaluated these conditional effects without following this order, the execution would be equivalent to not evaluating all conditional effects of the same operator simultaneously, thus risking to recognize a violators even if this does not happen. Indeed, given a preference $P = \mathsf{AO}_\phi$ and a threat operator o, if ϕ becomes true for the first time in s' after the application of o, an we check the condition $\mathit{cond}_{\mathcal{N}}(o,P)$ before $\mathit{cond}_{\mathcal{V}}(o,P)$, then we could detect a violation that may not have happened.

We can expose similar considerations for sometime-before preferences. In this case, if we do not check the condition $\mathit{cond}_{\mathcal{V}}(o,P)$ before $\mathit{cond}_{\mathcal{T}}(o,P)$, we risk to not correctly identifying a possible violators if ϕ and ψ become true at the same time.

Consequently, starting from the previous observations which show that it is possible to eliminate any interferences within our context, Nebel's copying operators are not needed in the compilation of our conditional effects; furthermore, in the compiled problem, we can force an arbitrary total order of the unconditional operator pairs, paying attention that the ordering constraints dealing with the potential interference between the conditional effects arising from at-most-once and sometime-before preferences are satisfied.

These changes to Nebel's schema simplify it, and have some beneficial consequences: (1) the compiled problem has smaller size in terms of number of operators, and (2) the search effort of a planner can be reduced because the solution plans are shorter without the coping actions,[3] and because the sequence of the unconditional operators can be explicit in the compiled problem, while with the original compilation it is built at search time by the planner.

Since our compilation of conditional effects can be easily derived from Nebel's method, instead of giving all the formal details of the translation, we illustrate the compilation with an example, referring the reader to [?] for a formal description. Moreover, in [?] we give a detailed description of the final (without conditional effects) compiled problem for any STRIPS+ problem with always preferences.

Compilation Example

[NOTA FP: ho usato ψ e ψ per le formule riferito alla preferenza always ed at-most-once dell'esempio per evitare i doppi indici nei pedici delle clausole e renderle piu' facilmente distinguibili]

Consider the operator $o = \langle \mathit{Pre}(o), \mathit{Eff}(o)\rangle = \langle \top, \{a, \neg c\}\rangle$ with cost κ, which affects two preferences A_ϕ and AO_ψ, where $\phi = \phi_1 \wedge \phi_2 = (a \vee b) \wedge (c \vee d)$ and $\psi = \psi_1 \wedge \psi_2 = (\neg c \vee e) \wedge (d \vee f)$.

Compilation of A_ϕ Using the conditions specified in Definition 6, we can classify o as a **threat** of A_ϕ because there exists a clause $\phi_2 = (c \vee d)$ of ϕ such that:

1. it contains at least a literal, i.e. c, that is negated by the effects of o: $|\overline{L}(\psi_1) \cap Z(o)| = |\overline{L}(c \vee d) \cap Z(o)| = |\{\neg c, \neg d\} \cap \{a, \neg c\}| = |\{\neg c\}| > 0$;
2. the other literal which does not satisfy the previous point, i.e. d, will not be certainly true in the resulting state from the application of o: $|L(\psi_1) \cap Z(o)| = |L(c \vee d) \cap Z(o)| = |\{c, d\} \cap \{a, \neg c\}| = 0$;
3. (the clause) is not false in the state where o is applied because the negation of its literals are not implied by the precondition of o: $\overline{L}(c \vee d) = \{\neg c, \neg d\} \not\subseteq \mathit{Pre}(o)$.

Operator o threatens only one clause of A_ϕ, i.e. $TC_{\mathcal{A}}(o, \mathsf{A}_\phi) = \{\phi_2\} = \{c \vee d\}$ (remember that $TC_{\mathcal{A}}(o, \mathsf{A}_\phi)$ is the set of threatened clauses of A_ϕ by o), which could be falsified if o is applied.

Using the preliminary definitions provided in Section , we define the set $\overline{AA}(o)_{\phi_2} = \{\neg d\}$ which contains those literal of the threatened clause ϕ_2 which are not falsified by the o execution.

Starting from these considerations and using Definition 16 we define the conditional effect $\mathcal{W}(o, P1)$ to add to

[3]Given a plan π for a problem with conditional effects and the corresponding plan π' for the compiled problem, we have $|\pi'| \leq |\pi| * m$ while with Nebel's general method this bound is $|\pi'| \leq |\pi| * (3 + m)$ [?].

Eff(o):

$$\mathcal{W}(o, P1) = \{\textit{when}\ ((\neg d),\ \{\neg d\} = \overline{AA}(o)_{\phi_2})\ (\mathsf{A}_\phi\textit{-violated})\} = \{\textit{when}\ (\neg d)\ (\mathsf{A}_\phi\textit{-violated})\}.$$

Compilation of AO_ψ According to Definition 12, we can classify o as a *threat* of AO_ψ because it could make true ψ in the state resulting from the application of o; indeed:

- there exists a clause of ψ, i.e. $\psi_1 = c \vee e$, which contains at least a literal, i.e. $\neg c$, that will be surely true in s': $|L(\phi_1) \cap Z(o)| = |L(\neg c \vee e) \cap Z(o)| = |\{\neg c, e\} \cap \{a, \neg c\}| = |\{\neg c\}| > 0$;

- while the other clause, i.e. $\psi_2 = d \vee f$, will not be certainly false in the state resulting the application of o: $|\overline{L}(\psi_2) \cap Z(o)| = |\overline{L}(d \vee f) \cap Z(o)| = |\{\neg d, \neg f\} \cap \{a, \neg c\}| = 0$

We denote with $C_\psi(o) = \{\psi_1\} = \{\neg c \vee e\}$ the set of clauses of ψ that will be surely true in the resulting state s'. Starting from these considerations and using Definition we define the conditional effect $\mathcal{W}(o, \mathsf{AO}_\psi)$ to add to *Eff*(o):

$$\mathcal{W}(o, \mathsf{AO}_\psi) = \{ \textit{when}\ (\textit{cond}_N(o, \mathsf{AO}_\psi))\ (\textit{seen-}\psi)), \textit{when}\ (\textit{cond}_T(o, \mathsf{AO}_\psi)))\ (\mathsf{AO}_\psi\textit{-violated})\}$$

where:

$$\textit{cond}_N(o, \mathsf{AO}_\psi) = \{\neg\textit{seen-}\psi\} \cup \{\psi_i \mid \psi_i \notin C_\psi(o)\} = = \{\neg\textit{seen-}\psi\} \cup \{\psi_2\} = \{\neg\textit{seen-}\psi, d \vee f\}$$

$$\textit{cond}_T(o, \mathsf{AO}_\psi) = \{\textit{seen-}\psi\} \cup \{\psi_i \mid \psi_i \notin C_\psi(o)\} \cup \{ \bigvee_{\psi_i \in C_\psi(o)} (\neg l_1 \wedge ... \wedge \neg l_q) \mid \{l_1, ..., l_q\} = L(\psi_i)\} = = \{\textit{seen-}\psi\} \cup \{\psi_2\} \cup \{\{\neg\neg c \wedge \neg e\} \mid \{\neg c, e\} = L(\psi_1)\} = = \{\textit{seen-}\psi, d \vee f, c \wedge \neg e\}$$

[FP: spiegazione commentata effetto condizionale della preferenza at-most-once, necessaria?]

Experimental Results

Experiments Description

We implemented the proposed compilation scheme and we have evaluated it by two sets of experiments with different purposes. On the one hand we have evaluated the scheme in a satisficing planning context in which we focused on the search for sub-optimal plans using different planning systems, while in the other we focuse on the search of optimal plans using admissible heuristics.

Regarding the comparison in the context of the satisficing planning we have considered the following STRIPS+ planning system LAMA[10], Mercury [7], MIPlan [8], IBaCoP2 [3], which are some of the best performing planning system in IPC8 [11], and Fast Downward Stone Soup 2018[**?**], Fast Downward Remix[**?**] (abbreviated with FDRemix), which are some of the best performing planning system in the last IPC9 [1] which have been compared with LPRPG-P [4], which is one of the performing planner which supports PDDL3 preferences. Moreover we have considered our specifically enhanced version of LAMA for planning with soft goal, which is LAMA$_\mathrm{P}(h_\mathrm{R})$, which makes use of admissibile heuristic h_R to test the reachability of the soft goals of the problem [9].

As benchmark we have considered the five domains of the qualitative preference track of IPC5 [6] which involve always, sometime, sometime-before, at-most-once and soft goal preferences, i.e Rovers, TPP, Trucks, Openstacks and Storage. Storage and TPP are particular domains without hard goals.

For each original problem all preferences and each original utility were kept. The classical planners were runned on the compiled problems while LPRPG-P was runned on the original problems of the competition. 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 8GiB respectively. We have tested 8 planners for 5 domains each of which consists of 20 instances for a total of 800 runs.

In order to realize a quality comparison we have considered two different quality metrics. The first one is the IPC quality score, a popular metric used in the IPC competitions [**?**], of which a brief description follows.

Given a planner p and a task i we assign, if p solves i, the following score 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 know solution for the task i found by any planner, and $\textit{cost}(p, i)$ is the cost of the solution found by the considered planner p in 30 minutes. In our case our reference for $\textit{cost}_{\textit{best}}(i)$ is equal to the cost of the best solution among the tested planners within 30 minutes. If p did not find a solution within the time assigned, then $\textit{score}(p, i)$ is equal to 0 in order to reward both quality and coverage.

We also considered another quality metrics, that we have denoted as α_{cost}, which is useful for understanding what class of preferences have been achieved and how important they are to achieve a good quality plan. A description follows. Given a planner p and a task i we assign, if p solves i, the following score to p:

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

where $\textit{cost}(p, i)$ is the cost of the solution found by planner p for the task i within 30 minutes and $\textit{cost}_{\textit{total}}(i)$ is the sum of the costs of all the preferences involved in the task i (note that $\mathscr{P}(i)$ denote the set of the preferences of the task i).

If we want to restric the calculation of $\alpha_{\textit{cost}}$ to a single type of preference, for example just always preferences, we denote the cost as $\alpha_{cost}(\mathcal{A})$ while if nothing is indicated, it means that we have considered all the classes of preferences.

From the previous definition, $\alpha_{cost}(p, i)$ could vary between 0 and 1. If $\alpha_{cost}(p, i) = 0$, then it means that the numerator $cost(p, i)$ is equal to 0 and that p has found an optimal plan for i which satisfies all the preferences of the problem. On the contrary, if $\alpha_{cost}(p, i) = 1$, then it means that p has found the worst plan for i where all the preferences of the problem are violated.

More generally given an instance i, the ratio $\alpha_{cost}(p, i)$, comparing plans produced by different systems, tell us which planner has achieved the satisfaction of the most useful subset of preferences in absolute terms. In particular, the planner with the lowest ratio is the planner who got the best performance on that particular instance.

In Tables 1 and 2 we have reported the performances of the considered planning system in term of IPC score quality aggregating the scores by domain

In particular in Table 1 we have reported the quality comparison considering all class of preferences together in the IPC calculation, while in 2 we splitted the table into six subtables where we have considered each class of preferences separetely in the IPC calculation (which is indicated in the header of each subtable).

Figures 2—6 instead show the α_{cost} comparison aggregated by domain. In this case each bar is associated to a planner whose value is equal to α_{cost} expressed as a percentage and calculated by adding up all the quality scores obtained in each instance. Each level of the stacked histogram represents the aggregated $\alpha_{\text{cost}}(\text{class})$ restricted to a specific class of preferences in order to show how much each class of violated preferences contributes to the total cost of the plans.

Note that in Table 1 we have reported the results about all the tested planners while in Table 2 and Figures 2—6 we have restricted the analysis, in order to synthetize the explanationy, to the following planners: LAMA$_{\text{P}}(h_{\text{R}})$ which realize the best IPC score performance, Fast Downward Stone Soup 2018 which is the best IPC score performing planners in IPC9, MIPlan which realizes a lowest IPC performance than LPRPG-P and the latter which the planning system which natively supports PDDL3 preferences. We have done some exceptions in the stacked histograms Figures for avoiding redundancy.

Concerning the optimal evaluation, similarly to what to what has been done in [12], we have tested our scheme using some admissible heuristics which are h^{blind}, which assign 1 to all states except for goal states to which assign 0, the maximum heuristic $h^{\max}$ [**?**], and the canonical pattern database heuristic h^{cpdb} [**?**].

These heuristitcs used with A* algorithm guarantee the optimality of the solution found. Starting from the IPC5 domains we have generated, likewise what was done in [12], simpler instances by randomly sampling subsets of the soft trajectory constraints. Starting from each instance we have generated five new instances with 1%, 5%, 10%, 20% and 40% of (grounded) soft trajectory constraints while the hard goals have remained unchanged if they exist.

Since we do not have the same instances that have been used in the aforementioned paper, we have generated, for each percentage of sampling preferences (except for 100 %), 3 sampled instances in order to average the obtained results in order to have a better comparison with their approach.

All these experiments about the optimality 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 8GiB respectively, while the experiments reported in [12] are conducted on a Intel(R) Xeon(R) E5-2650v2 2.60GHz processors with 64GiB with one hour of CPU-time for the search and then our approach is penalized.

The results about these experiments and the comparison with the automata approach are shown Table 3. The results inherent to Openstacks have been excluded because it was not possible to find optimal plans even for the simplest instances. Overall we have tested 3 admissible heuristics for 4 domains, each of which consists of 320 instances, the 20 original ones plus 300 sampled instances $(20 * \text{sampling\_rates} * \text{sample})$, where $\text{sampling\_rates} = 5$ and $\text{rate} = 3)$, for a total of 4800 runs.

Satisficing Planning Results

The results obtained comparing the satisficing context show that the compilative approach is almost always preferable since the tested classical planners obtain an higher IPC score than LPRPG-P except for Mercury and MIPlan.

With reference to Table 1 the compilative approach seems at glance to be particularly preferable in Rovers, Trucks and Storage. In these domains each classical planner performs better or at least comparable than LPRPG-P (except for Mercury in Trucks); IBaCoP2 performs particularly well in Rovers, FDRemix in Trucks and 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 planning system from the more recent IPC9, FDRemix and Fast Downward Stone Soup 2018, perform overall better in this benchmark than the planning system from the previous IPC8 but the enhanced version LAMA$_{\text{P}}(h_{\text{R}})$ is better than everyone else indeed it improves the performance of LAMA in all the considered domains except in Rovers (where there is no soft goal).

Looking at Figure 2 we note that the IPC gap between the classical planners and LPRPG-P in Rovers is due to the LPRPG-P violation of sometime-before preferences. As regards the remaining classes preferences, the violation cost settles down on similar level except for IBaCoP2, which achieves more sometime-before and sometime preferences than the others planning violating more at-most-once preferences but succeeding to obtaining better quality plans.

In TPP the compilative approach seems to be very ineffective, indeed each classical planner achieves an extremely lower quality performance compared to LPRPG-P. The bad performances in this domain are due to the many soft goals and sometime preferences because, as shown in [9], the compilation of soft goals can be sometime problematic and neither the use of the reachability heuristic h_{R} in LAMA$_{\text{P}}(h_{\text{R}})$ can compensate this weakness. Indeed looking at Table 2 we note that LPRPG-P gets an high IPC score for these two preference classes than all the classical planners. Looking at Table 1 we can also observe that the clas-

Planner	Rovers	TPP	Trucks	Openstacks	Storage	TOTAL
LAMA$_{\mathrm{B}}$(h_{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.

sical planners achieves a better result of always, sometime-before and at-most-once preferences compared to LPRPG-P in term of IPC score, but this is not very relevant for the plan quality because apparently it happens at the expense of soft-goal and sometime preferences which are clearly more expensive to violate (or equivalently more useful to satisfy), indeed looking to Figure 3 we note that the decisive preferences in this domain are mainly soft goals which are more achieved by LPRPG-P. Note that LAMA$_{\mathrm{P}}$(h_{R} helps a bit LAMA to achieves more soft goals.

Looking at Figure 4 we observe that the classical planners, except MIPlan, achieves a comparable performance with LPRPG-P and in particular LAMA$_{\mathrm{P}}$(h_{R} and Fast Downward Stone Soup 2018 get higher quality plan because they manage to achieve almost all the always preferences and more sometime-before preferences than their competitor.

Regarding Opentacks and looking at Table 1 all the tested planners achieve a comparable performance in term of IPC score even if the classical planners are slightly penalized compared to LPRPG-P. Looking at Figure 5 we can assert that the only relevant classes of preferences in this domain are soft goals and always preferences and every planner performs a similar performance (except for IBaCoP2 amd MIPlan that do worse).

Looking at Figure 6 we observe that all the classical planners, achieves a better performance than LPRPG-P and in particular LAMA$_{\mathrm{P}}$(h_{R} and MIPlan get higher quality plan because they manage to achieve almost the sometime-before preferences than their competitor.

NOTA (FP): scrivi commento Storage.

$\mathscr{P}_{\mathcal{A}}$						
Planner	Rovers	TPP	Trucks	Openstacks	Storage	TOTAL
LAMA$_{\mathrm{P}}$(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
$\mathscr{P}_{\mathcal{SG}}$						
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
LAMA$_{\mathrm{P}}$(h_{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
LAMA$_{\mathrm{P}}$(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
$\mathscr{P}_{\mathcal{SB}}$						
Planner	Rovers	TPP	Trucks	Openstacks	Storage	TOTAL
LAMA$_{\mathrm{P}}$(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
$\mathscr{P}_{\mathcal{ST}}$						
Planner	Rovers	TPP	Trucks	Openstacks	Storage	TOTAL
LAMA$_{\mathrm{P}}$(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.

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.

Domain	h^{blind}		$h^{\max}$		h^{cpdb}	
	WRB	PG	WRB	PG	WRB	PG
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 (PG) 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.

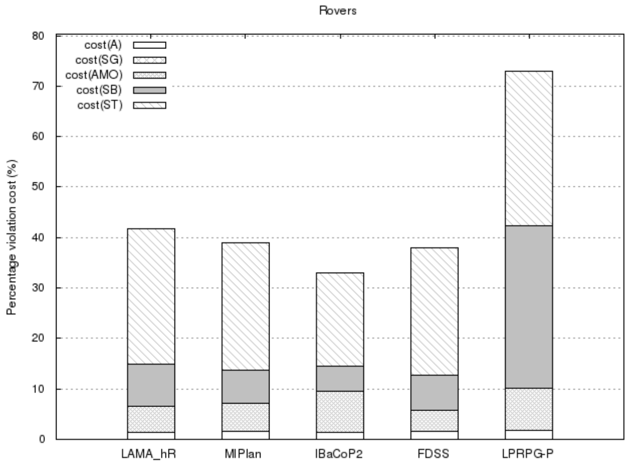

Figure 2: α_{cost} comparison for Rovers domain.

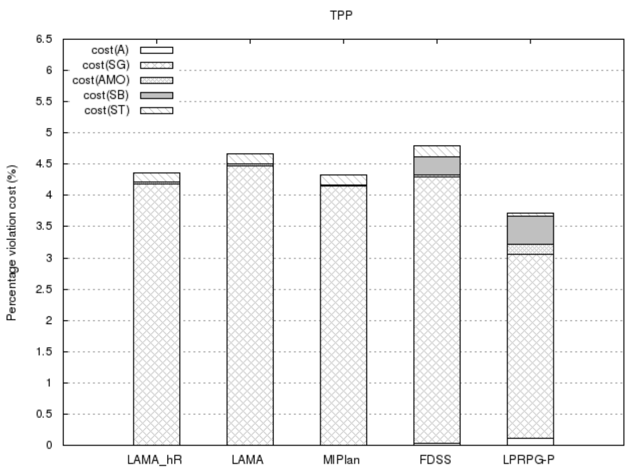

Figure 3: α_{cost} comparison for TPP domain.

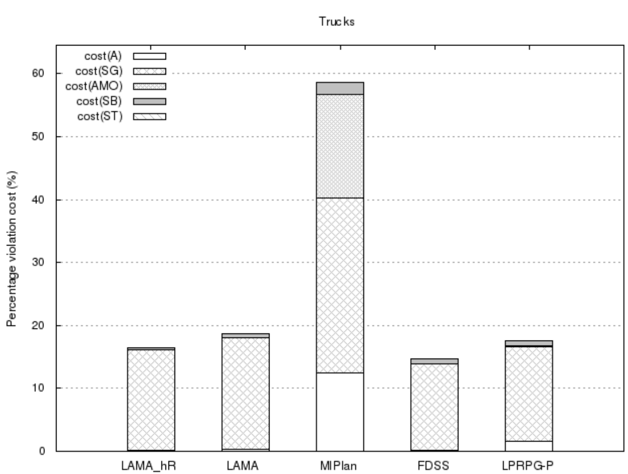

Figure 4: α_{cost} comparison for Trucks domain.

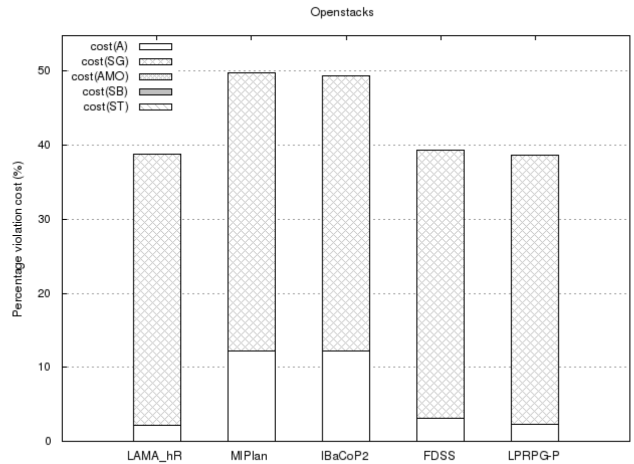

Figure 5: α_{cost} comparison for Openstacks domain.

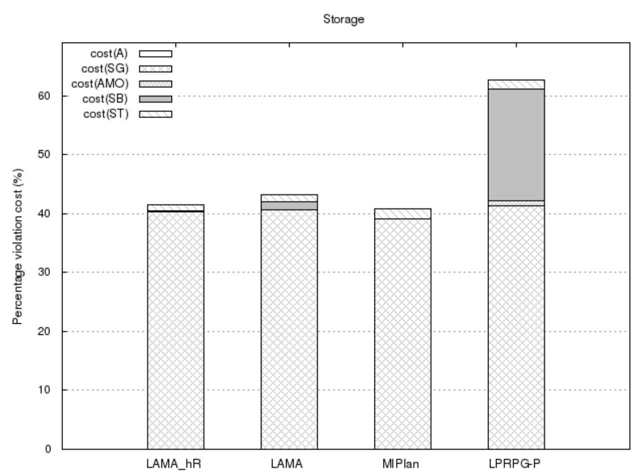

Figure 6: α_{cost} comparison for Storage domain.

[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 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.