Proceedings of the Eighth International Symposium on Combinatorial Search (SoCS-2015)

Planning with Always Preferences by Compilation into STRIPS with Action Costs

Luca Ceriani and **Alfonso E. Gerevini**[*]
Department of Information Engineering, University of Brescia, Italy
luca.ceriani@unibs.it, alfonso.gerevini@unibs.it ([*]corresponding author)

Abstract

We address planning with always preferences in propositional domains, proposing a new compilation schema for translating a STRIPS problem enriched with always preferences (and possibly also soft goals) into a STRIPS problem with action costs. Our method allows many STRIPS planners to effectively address planning with always preferences and soft goals. An experimental analysis indicates that such basic planners are competitive with current planners using techniques specifically developed to handle always preferences.

Introduction

Planning with preferences, also called "over-subscription planning" in (Briel et al. 2004; Do and Kambhampati 2004; Smith 2004), 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. A useful class of preferences than can be expressed in PDDL3 (Gerevini et al. 2009) consists of *always preferences*, requiring that a certain condition should hold in *every* state reached by a plan. As discussed in (Weld and Etzioni 1994; Bacchus and Kabanza 1998; Gerevini et al. 2009; Baier, Bacchus, and McIlraith 2009), 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".

We address propositional planning with always preferences by a compilation approach. Inspired by the work of Keyder and Geffner (2009) on compiling soft goals into STRIPS with action costs (here denoted with STRIPS+), we propose a compilation scheme for translating a STRIPS problem with always preferences into an equivalent STRIPS+ problem. Handling action costs is a practically important, basic functionality that is supported by many powerful planners, and the proposed complilation method allows them to support (through the compiled problems) always preferences with no change to their algorithms and code.

The most prominent existing planners supporting always preferences (as well as other types of PDDL3 preferences) are Hplan-P (Baier and McIlraith 2006; Baier, Bacchus, and McIlraith 2009), which won the "qualitative preference" track of IPC-5, MIPS-XXL (Edelkamp 2006; Edelkamp, Jabbar, and Nazih 2006) and the more recent LPRPG-P (Coles and Coles 2011). 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 (2004) and Rintanen (2000) are also somewhat related to ours, but with important differences. Their methods handle *hard* always goals instead of preferences. Rintanen's compilation considers only single literals in the always formulae (while we deal with arbitrary CNF formulas), and extending it to handle more general formulas requires substantial new techniques (Rintanen 2015). An implementation of Crosswell and Coddington's approach is unavailable, but Bayer and McIlaraith (2006) observed that their approach suffers exponential blow up problems and performs less efficiently than the approach of Hplan-P.

An experimental analysis indicates that solving propositional problems with always preferences by using our compiled problems in state-of-the-art STRIPS+ planners is competitive with, and can be more effective than, solving the original uncompiled problems with Hplan-P, MIPS-XXL or LPRPG-P, in terms of satisfied preferences and scalability.

In the following, after some background and preliminaries, we describe our compilation method, present some experimental results, and finally give the conclusions.

STRIPS+ with Always 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

[1]Another compilation scheme using numerical fluents is considered in (Gerevini et al. 2009) to study the expressiveness of PDDL3.0 (without an implementation).

Copyright © 2015, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

$\langle \mathit{Prec}(o), \mathit{Eff}(o)\rangle$, where $\mathit{Prec}(o)$ 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 $s_0 = I$, $\mathit{Prec}(a_i) \subseteq s_i$ 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.

Without loss of generality, we will assume that the condition of a preference A_i is expressed in CNF form, i.e., $A_i = ap_1 \wedge \dots \wedge ap_n$, where each ap_j ($j \in [1 \dots n]$) is a *clause of* A_i formed by literals over the problem fluents. We write $\pi \models_{\mathsf{a}} A_i$ to indicate that plan π satisfies A_i.

Definition 1. *A* STRIPS+ *problem with always preferences is a tuple $\langle F, I, O, G, AP, c, u\rangle$ where: $\langle F, I, O, G, c\rangle$ is a* STRIPS+ *problem; AP is a set of always preferences; u is an utility function mapping each $A_i \in AP$ to a value in $\mathbb{R}^+_o$.*

In the following, the class of STRIPS+ with always preferences is indicated with STRIPS+AP.

Definition 2. *Let Π be a* STRIPS+AP *problem with a set of preferences AP. The utility $u(\pi)$ of a plan π solving Π is the difference between the total utility obtained by the plan and its cost, i.e., $u(\pi) = -\,c(\pi) + \sum_{A_i \in AP : \pi \models_{\mathsf{a}} A_i} u(A_i)$.*

The definition of plan utility for STRIPS+AP is similar to the one given for STRIPS+ with soft goals by Keyder and Geffner (2009). A plan π with utility $u(\pi)$ for a STRIPS+AP problem is optimal when no other plan π' has utility $u(\pi') > u(\pi)$. The next two definitions introduce some notation useful to simplify the following presentation.

Definition 3. *Given a preference clause $ap = l_1 \vee \dots \vee l_n$, the set $L(ap) = \{l_1, \cdots, l_n\}$ is the equivalent set-based definition of ap and $\overline{L}(ap) = \{\neg l_1, \cdots, \neg l_n\}$ is the literal-complement set of $L(ap)$.*

Definition 4. *Given an operator $o \in O$ of a* STRIPS+AP *problem, $Z(o)$ is the set of literals defined as:*
$Z(o) = (\mathit{Prec}(o) \setminus \{p \mid \neg p \in \mathit{Eff}(o)^-\}) \cup \mathit{Eff}(o)^+ \cup \mathit{Eff}(o)^-$.

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

In our compilation of a STRIPS+AP problem, it is important to distinguish three types of operators that are specified in the following Definitions 5,6,7.

Definition 5. *Given an operator o and a preference A of a* STRIPS+AP *problem, o is a* **violation** *of A if there is a clause ap of A such that $\overline{L}(ap) \subseteq Z(o) \wedge \overline{L}(ap) \not\subseteq \mathit{Prec}(o)$.*

If an operator violates a preference, the preference is unsatisfied in any state resulting from the application of the operator. The set of preferences in a STRIPS+AP problem that are violated by an operator o is denoted with $V(o)$.

Definition 6. *Given an operator o and a preference A of a* STRIPS+AP *problem, o is a* **threat** *for A if it is not a violation and there exists a clause ap of A, such that:*
$\overline{L}(ap) \cap Z(o) \neq \emptyset \wedge L(ap) \cap Z(o) = \emptyset \wedge \overline{L}(ap) \not\subseteq \mathit{Prec}(o)$.

A clause ap of A satisfying the condition of Definition 6 is a *threatened clause* of A. A threatened preference (clause) may be falsified by an operator depending on the state where the operator is applied. The set of preferences threatened by an operator o is denoted with $T(o)$; the set of clauses of a preference A threatened by o is denoted with $T_A(o)$. Note that conjunct $\overline{L}(ap) \not\subseteq \mathit{Prec}(o)$ in the conditions of Definitions 5-6 avoids that an operator o is considered a violation/threat when its preconditions require ap to be *already* violated in the state where it is applied.

Definition 7. *Given an operator o and a preference A of a* STRIPS+AP *problem, o is* **safe** *for A if: (i) for all clauses ap of A, $L(ap) \cap Z(o) \neq \emptyset$ or $\overline{L}(ap) \cap Z(o) = \emptyset$ holds, or (ii) there exists a clause ap such that $\overline{L}(ap) \subseteq \mathit{Prec}(o)$.*

If an operator o of a STRIPS+AP problem Π is safe for every always preference of Π, we say that *the operator is safe for* Π, and we write this property with $\mathit{Safe}(o, \Pi)$.

Compilation into STRIPS+

Given a STRIPS+AP problem, an equivalent STRIPS+ problem can be derived by a translation similar to the one of Keyder and Geffner (2009) for soft goals. Their compilation is considerable simpler than ours because for soft goals there is no need to consider threatening and violating operators. For the sake of simplicity, we don't consider soft goals, but their compilation into STRIPS+ can be easily added to our compilation for always preference using the same method of Keyder and Geffner. Moreover, we will assume that every preference is satisfied in the problem initial state I.[2]

Given a STRIPS+AP problem $\Pi = \langle F, I, O, G, AP, c, u\rangle$, the compiled STRIPS+ problem of Π is $\Pi' = \langle F', I', O', G', c'\rangle$ with:

- $F' = F \cup AV \cup D \cup C' \cup \overline{C'} \cup \{\mathit{normal\text{-}mode}, \mathit{end\text{-}mode}, \mathit{pause}\}$;
- $I' = I \cup \overline{C'} \cup \{\mathit{normal\text{-}mode}\}$;
- $G' = G \cup C'$;
- $O' = \{\mathit{collect}(A), \mathit{forgo}(A) \mid A \in AP\} \cup \{\mathit{end}\} \cup O_{comp}$;
- $c'(o) = \begin{cases} u(A) & \text{if } o = \mathit{forgo}(A) \\ c(o) & \text{if } o \in O_S \\ c_{tv}(o) & \text{if } o \in O_T \cup O_V \\ 0 & \text{otherwise} \end{cases}$

where:

- $AV = \bigcup_{i=1}^{k} \{A_i\text{-}\mathit{violated}\}, k = |AP|$;
- $D = \bigcup_{i=1}^{k} \{A_i\text{-}\mathit{done}_{o_1}, \dots, A_i\text{-}\mathit{done}_{o_n}\}$, $k = |AP|$ and $n =$ is number of operators threatening or violating A_i;
- $C' = \{A' \mid A \in AP\}$ and $\overline{C'} = \{\overline{A'} \mid A' \in C'\}$;
- $\mathit{collect}(A) = \langle\{\mathit{end\text{-}mode}, \neg A\text{-}\mathit{violated}, \overline{A'}\}, \{A', \neg\overline{A'}\}\rangle$;
- $\mathit{forgo}(A) = \langle\{\mathit{end\text{-}mode}, A\text{-}\mathit{violated}, \overline{A'}\}, \{A', \neg\overline{A'}\}\rangle$;
- $\mathit{end} = \langle\{\mathit{normal\text{-}mode}, \neg\mathit{pause}\}, \{\mathit{end\text{-}mode}, \neg\mathit{normal\text{-}mode}\}\rangle$;
- $O_S = \{\langle \mathit{Pre}(o) \cup \{\mathit{normal\text{-}mode}, \neg\mathit{pause}\}, \mathit{Eff}(o)\rangle \mid o \in O \text{ and } \mathit{Safe}(o, \Pi)\}$;
- $O_{comp} = O_S \cup O_T \cup O_V$;

[2]Unsatisfied preferences in I can be easily handled as we describe in a workshop paper (Ceriani and Gerevini 2014) including proofs of the main compilation properties.

– O_T and O_V are the operator sets generated by the operator transformation schema applied to the operators of Π that threaten and violate, respectively, a preference of Π.[3] Such sets will be defined after presenting the general idea for compiling an operator threatening a preference.

– $c_{tv}(o)$ is the cost of an operator o in $O_V \cup O_T$ that we define after describing how exactly these sets are formed.

For each preference A the transformation of Π into Π' adds a dummy hard goal A' to Π' which can be achieved in two ways: with action *collect(A)*, that has cost 0 but requires A to be satisfied (i.e. *A-violated* is false in the goal state), or with action *forgo(A)*, that has cost equal to the utility of A and can be performed when A is false (*A-violated* is true in the goal state). For each preference exactly one of $\{$*collect(A), forgo(A)*$\}$ appears in the plan.

We now present the transformation of operators that threaten or violate a preference. (Safe operators are compiled through O_S as indicated above.) The compilation of a threatening operator is not simple since it can violate multiple preferences and a preference is an arbitrary CNF formula that can be violated in different ways, depending on the operator definition and the state the operator is applied to.

Given an operator o threatening k preferences $A_{1\ldots k}$, o can be compiled by adding k conditional effects *(when c_i A_i-violated)*, for $i = 1 \ldots k$, where c_i is a formula that, when satisfied in the state where o is applied, makes o violate A_i. Condition c_i can be derived by regression of A_i over o (it is analogous to precondition $l_1 \wedge \ldots \wedge l_q$ in the $\overline{o}$-operators of Definition 8). However, with this method, in order to keep the compiled problem in the STRIPS+ class, the conditional effects have to be compiled away, which leads to an exponential blow up of operators (in our example 2^k for o) (Gazen and Knoblock 1997). Since an original operator can threaten many preferences (e.g., $k = 20$), in the following we propose a more elaborated transformation generating only a linear number of operators ($2k$ instead of 2^k).

Each operator o such that $T(o) \neq \emptyset$ is compiled into a set of new operators (one for each threatened preference). If $T(o) = \{A_1, \ldots, A_m\}$, o is compiled into a set of $2m$ operators $O_{T(o)} = \{o_{A_1}, \overline{o}_{A_1}, \ldots, o_{A_m}, \overline{o}_{A_m}\}$ such that, in any state s where o can be applied violating a set of preferences in $T(o)$, the *sequence $\omega_{T(o)}$ of m operators in $O_{T(o)}$* defined as follows can be applied: $\omega_{T(o)} = \langle o'_{A_1}, \ldots, o'_{A_m} \rangle$, where $o'_{A_i} = \overline{o}_{A_i}$ if o violates A_i when applied in s, $o'_{A_i} = o_{A_i}$ if o does not violate A_i when applied in s, and $\overline{o}_{A_i}$ and o_{A_i} are mutually exclusive, for $i \in [1 \ldots m]$.

The cost $c_{tv}(o^t)$ of an operator $o^t \in O_{T(o)}$ is $c(o)$ if $o \in \{o_{A_1}, \overline{o}_{A_1}\}$, 0 otherwise (exactly one of the operators in ω_T has cost equal to the cost of the original domain operator o compiled into $O_{T(o)}$, while the others have cost zero).

Before defining the operators of $O_{T(o)}$ and $\omega_{T(o)}$, we introduce some notation useful to simplify their formalisation. For an operator o and a preference clause ap:

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

[3]Note that if an operator is both a violation and a threat for the same preference, the generated compiled operators are only in O_T.

Definition 8. *The operators $O_{T(o)}$ of an operator o threatening a set of preferences $T(o) = \{A_1, \ldots, A_m\}$ are:*

- *For o_{A_1} and $\overline{o}_{A_1}$:*
  $\mathit{Prec}(o_{A_1}) = \mathit{Prec}(o) \cup \{\neg \mathit{pause}\} \cup$
  $\quad \bigcup_{ap \in T_{A_1}(o)} \{(l_1 \vee \ldots \vee l_p) \mid \{l_1, \ldots, l_p\} = AA(o)_{ap}\}$
  $\mathit{Eff}(o_{A_1}) = \{A_1\text{-}\mathit{done}_o, \mathit{pause}\}$
  $\mathit{Prec}(\overline{o}_{A_1}) = \mathit{Prec}(o) \cup \{\neg \mathit{pause}\} \cup$
  $\quad \{\bigvee_{ap \in T_{A_1}(o)} (l_1 \wedge \ldots \wedge l_q) \mid \{l_1, \ldots, l_q\} = \overline{AA}(o)_{ap}\}$
  $\mathit{Eff}(\overline{o}_{A_1}) = \{A_1\text{-}\mathit{done}_o, \mathit{pause}, A_1\text{-}\mathit{violated}\}$
- *For o_{A_k} and $\overline{o}_{A_k}$ with $k \in [2 \ldots m-1]$:*
  $\mathit{Prec}(o_{A_k}) = \{A_{k-1}\text{-}\mathit{done}_o, \mathit{pause}\} \cup$
  $\quad \bigcup_{ap \in T_{A_k}(o)} \{(l_1 \vee \ldots \vee l_p) \mid \{l_1, \ldots, l_p\} = AA(o)_{ap}\}$
  $\mathit{Eff}(o_{A_k}) = \{A_k\text{-}\mathit{done}_o, \neg A_{k-1}\text{-}\mathit{done}_o\}$
  $\mathit{Prec}(\overline{o}_{A_k}) = \{A_{k-1}\text{-}\mathit{done}_o, \mathit{pause}\} \cup$
  $\quad \{\bigvee_{ap \in T_{A_k}(o)} (l_1 \wedge \ldots \wedge l_q) \mid \{l_1, \ldots, l_q\} = \overline{AA}(o)_{ap}\}$
  $\mathit{Eff}(\overline{o}_{A_k}) = \{A_k\text{-}\mathit{done}_o, A_k\text{-}\mathit{violated}, \neg A_{k-1}\text{-}\mathit{done}_o\}$
- *For o_{A_m} and $\overline{o}_{A_m}$:*
  $\mathit{Prec}(o_{A_m}) = \bigcup_{ap \in T_{A_m}(o)} \{(l_1 \vee \ldots \vee l_p) \mid \{l_1, \ldots, l_p\} =$
  $\quad AA(o)_{ap}\} \cup \{A_{m-1}\text{-}\mathit{done}_o, \mathit{pause}\}$
  $\mathit{Eff}(o_{A_m}) = \mathit{Eff}(o) \cup \{\neg A_{m-1}\text{-}\mathit{done}_o, \neg \mathit{pause}\}$
  $\mathit{Prec}(\overline{o}_{A_m}) = \{\bigvee_{ap \in T_{A_m}(o)} (l_1 \wedge \ldots \wedge l_q) \mid \{l_1, \ldots, l_q\} =$
  $\quad \overline{AA}(o)_{ap}\} \cup \{A_{m-1}\text{-}\mathit{done}_o, \mathit{pause}\}$
  $\mathit{Eff}(\overline{o}_{A_m}) = \mathit{Eff}(o) \cup \{\neg A_{m-1}\text{-}\mathit{done}_o, A_m\text{-}\mathit{violated}, \neg \mathit{pause}\}$.

The operators of $O_{T(o)}$ can be applied only in a sequence $\omega_{T(o)}$ defined above. The preconditions of any operator $\overline{o}_{A_i}$ in $\omega_{T(o)}$ require $\overline{AA}(o)_{ap}$ to hold in the state where $\omega_{T(o)}$ is applied for at least one $ap \in T_{A_i}(o)$. If this happens, then $\overline{o}_{A_i} \in \omega_{T(o)}$ and preference A_i is violated by $\omega_{T(o)}$. Predicate A_i*-violated* is made true by $\overline{o}_{A_i}$ and is never falsified.

After the *end* action is applied, A_i*-violated* serves as a precondition of the operator *forgo*(A_i) that has cost equal to the utility of A_i. The A_i*-done*$_o$ predicates force the planner to strictly follow the order in $\omega_{T(o)}$, avoiding repetitions. Once the planner starts sequence $\omega_{T(o)}$ for some o, no other operator $o' \notin O_{T(o)}$ is enabled before the application of $\omega_{T(o)}$ is completed. Predicate *pause* serves to this purpose, and only the last action in $\omega_{T(o)}$ (o_{A_m} or $\overline{o}_{A_m}$) falsifies it.

Each domain operator o violating a preference ($V(o) \neq \emptyset$) must be compiled as well. The compilation schema for o is simpler than the one for a threatening operator. It is sufficient to add to *Eff*(o) a *A-violated* predicate for each preference $A \in V(o)$. If the modified operator is applied, all the preferences it violates are falsified (making their *violated*-predicates true) and corresponding *forgo* actions must later be selected by the planner for every valid plan. Note that an operator o can simultaneously be a violation and a threat of different preference sets. If an operator o threatening a set of preferences also violates a preference A, effect *A-violated* is added to the first pair of operators $(o_{A_1}, \overline{o}_{A_1})$ in $O_{T(A)}$. The cost $c_{tv}(o^v)$ of an operator o^v derived by compiling an original operator o that violates a preference (and does not threat any other preference) is the the same cost $c(o)$ of o.

The compilation of a threatening operator can generate operators with negated literals and disjunctions of (conjunction of) literals in the preconditions. Such operators can then be translated into STRIPS+ operators by the method described in (Gazen and Knoblock 1997).

Rovers	**Mercury**		**LAMA**		**LPRPG-P**		**HPlan-P**	
	%G	**%A**	**%G**	**%A**	**%G**	**%A**	**%G**	**%A**
P01 (0/3)	-	100	-	100	-	100	-	100
P02 (0/3)	-	100	-	100	-	100	-	100
P03 (0/5)	-	100	-	100	-	100	-	100
P04 (0/5)	-	100	-	100	-	100	-	?
P05 (0/5)	-	100	-	100	-	60	-	100
P06 (0/5)	-	60	-	60	-	60	-	60
P07 (0/6)	-	80	-	80	-	80	-	100
P08 (0/6)	-	100	-	83	-	100	-	100
P09 (0/6)	-	67	-	83	-	67	-	100
P10 (0/6)	-	100	-	100	-	100	-	100
P11 (0/6)	-	83	-	83	-	67	-	?
P12 (0/6)	-	100	-	100	-	67	-	100
P13 (0/6)	-	100	-	100	-	100	-	100
P14 (0/6)	-	100	-	100	-	83	-	?
P15 (0/6)	-	100	-	100	-	100	-	?
P16 (0/6)	-	67	-	100	-	83	-	?
P17 (0/8)	-	100	-	100	-	100	-	?
P18 (0/8)	-	100	-	100	-	88	-	?
P19 (0/8)	-	75	-	75	-	100	-	?
P20 (0/10)	-	80	-	90	-	?	-	?

TPP	**Mercury**		**LAMA**		**LPRPG-P**		**HPlan-P**	
	%G	**%A**	**%G**	**%A**	**%G**	**%A**	**%G**	**%A**
P01 (1/2)	100	100	100	100	100	100	100	100
P02 (6/6)	83	100	83	100	83	83	83	100
P03 (9/8)	67	100	67	100	67	63	?	?
P04 (12/8)	83	100	83	75	83	50	?	?
P05 (20/14)	85	86	85	93	75	79	?	?
P06 (24/14)	75	93	75	86	63	71	58	64
P07 (28/14)	75	86	75	86	61	64	54	64
P08 (32/16)	75	88	75	88	66	63	56	63
P09 (45/22)	82	68	82	68	73	68	64	68
P10 (50/42)	80	81	80	81	76	76	74	83
P11 (55/42)	80	79	80	83	69	60	?	?
P12 (60/45)	80	76	80	80	70	62	63	76
P13 (78/57)	83	81	83	81	76	67	?	?
P14 (84/54)	83	81	83	83	75	81	?	?
P15 (90/57)	83	86	84	75	74	81	?	?
P16 (96/60)	83	73	83	82	76	72	?	?
P17 (119/69)	86	80	86	81	76	75	?	?
P18 (126/72)	86	78	86	76	?	?	?	?
P19 (133/72)	86	76	86	79	?	?	?	?
P20 (140/75)	?	?	85	75	?	?	?	?

Trucks	**Mercury**		**LAMA**		**LPRPG-P**		**HPlan-P**	
	%G	**%A**	**%G**	**%A**	**%G**	**%A**	**%G**	**%A**
P01 (2/1)	100	100	100	100	100	0	100	100
P02 (3/10)	67	50	67	50	100	30	100	40
P03 (1/12)	?	?	?	?	0	8	?	?
P04 (2/15)	100	40	100	53	100	20	?	?
P05 (2/17)	0	24	50	53	0	53	?	?
P06 (2/20)	0	35	50	55	100	30	?	?
P07 (6/31)	0	19	0	71	83	58	?	?
P08 (8/53)	0	49	13	71	88	46	?	?
P09 (9/38)	0	47	11	66	89	50	?	?
P10 (7/42)	0	43	0	67	57	38	?	?
P11 (7/45)	0	31	0	69	29	42	?	?
P12 (12/49)	0	47	8	71	42	41	?	?
P13 (11/52)	0	40	0	71	45	46	?	?
P14 (11/56)	9	41	9	70	64	38	?	?
P15 (12/59)	0	42	0	71	58	46	?	?
P16 (15/81)	0	27	7	77	60	56	?	?
P17 (18/85)	11	42	22	75	44	42	?	?
P18 (16/90)	19	53	13	77	69	52	?	?
P19 (16/94)	6	41	0	76	44	45	?	?
P20 (17/99)	0	39	0	77	47	51	?	?

Table 1: Performance comparison about solving uncompiled and compiled STRIPS+AP problems. The values in brackets of the 1st column are the total *numbers of soft goals/always preferences*. **%G** is % of satisfied *soft goals*; **%A** is % of satisfied *always preferences*; **?** indicates *unsolved problem.*

Experimental Results

We implemented the proposed compilation schema and compared the performance of the two state-of-the-art STRIPS+ planners Mercury (Katz and Hoffmann 2014) and LAMA (Richter and Westphal 2010) with Hplan-P and MIPS-XXL and LPRPG-P.[4] For brevity, results for MIPS-XXL are not reported since this planner performed generally worse than both Hplan-P and LPRPG-P. We considered the five domains, and corresponding test problems involving always preferences (below abbreviated with AP), of the qualitative preference track of IPC5 (Gerevini et al. 2009). Here we focus the presentation on three of them (`Rovers`, `TPP` and `Trucks`), while for the others (`Openstack` and `Storage`) we give general results for the STRIPS+ planners and Hplan-P. For `Openstacks` LPRPG-P generates no plans (apparently because these test problems don't have hard goals), and for `Storage` the problems use some PDDL features that are not supported by the available version of LPRPG-P.

For each original IPC5 problem, all soft goals were kept, while all types of preferences different from APs were removed. Additional domain-specific APs were added to make the problems more challenging. Specifically, in `Trucks` it is requested that 50% of the packages should be delivered by half of the originally specified deadline (using discrete levels of time). In `Rovers`, each rover should always avoid a specified set of locations; moreover, we used APs to specify the preference that at least one rover store remains empty in every state reached by a plan. Finally, in `TPP` APs are used to request that each type of goods should be carried by a specific truck, which has to buy the total goal quantity of the good and unload it visiting a deposit no more than once. The association between goods and trucks is randomly decided. For each test problem, the utility of every AP is one, and the cost of every domain action is zero. The CPU-time limit for each run of each planner was 30 minutes.

Table 1 shows results for `Rovers`, `TPP` and `Trucks` obtained by running Hplan-P and LPRPG-P over the original uncompiled problems, and Mercury and LAMA over the equivalent compiled problems. For each considered problem, the table indicates the percentages of soft goals and always preferences satisfied by each of the compared planners.

In general, we observe that Hplan-P solves many less problems than the STRIPS+ planners and LPRPG-P. Concerning the relative performance of LPRPG-P and the considered STRIPS+ planners, we have the following results. For `Rovers`, in terms of satisfied always preferences, both STRIPS+ planners satisfy more preferences in several problems, while LPRPG-P performs better only in very few problems, but fails to solve the largest one. Note that in `Rovers` there are no soft goals. For `TPP`, LAMA performs generally best in terms of both satisfied always preferences and soft goals. Moreover, the largest problems are solved only by the STRIPS+ planners. For `Trucks`, in general LAMA performs better than the other compared planners in terms of always preferences, but LPRPG-P satisfies more soft goals. Overall, in terms of problem coverage (regardless plan quality), Hplan-P solved 21 problems, LPRPG-P 56, Mercury 58, and LAMA 59.

Finally, for both `Openstack` and `Storage` we observed that the considered STRIPS+ planners perform generally better than Hplan-P. In particular, LAMA solves 70% more `Openstack` problems than Hplan-P, and for the 7 problems of this domain that both planners solve, on average LAMA satisfies 95% of the preferences while Hplan-P 82%.

Conclusions

We have presented a compilation method for handling always preferences in propositional planning. The compiled problems use only STRIPS and action costs, which makes the compilation method usable by many existing powerful planners. A preliminary experimental analysis shows their good behaviour in terms of scalability and quality of the generated plans (satisfied preferences) compared to existing state-of-the-art planners supporting always preferences.

In addition to a deeper experimental analysis, current work concerns the compilation of other types of PDDL3 preferences into STRIPS+, although always preferences in the problem model are already practically useful by themselves.

[4]Planner Mercury was 2nd best planner in IPC8. We tried also the available version of IbACop2 (winner of IPC8), but we could not properly test it due to several internal crash of it during testing.

References

Bacchus, F., and Kabanza, F. 1998. Planning for temporally extended goals. *Annals of Mathematics and Artificial Intelligence* 22(1-2):5–27.

Baier, J., and McIlraith, S. 2006. Planning with first-order temporally extended goals using heuristic search. In *Proc. of the 21st National Conference on Artificial Intelligence AAAI*, 788–795. AAAI Press.

Baier, J.; Bacchus, F.; and McIlraith, S. 2009. A heuristic search approach to planning with temporally extended preferecens. *Artificial Intelligence* 173:593–618.

Briel, M.; Sanchez, R.; Do, M.; and Kambhampati, S. 2004. Effective approaches for partial satisfaction (over-subscription) planning. In *Proc. of 19th National Conf. on Artificial Intelligence (AAAI'04)*.

Ceriani, L., and Gerevini, A. E. 2014. Planning with preferences by compiling soft always goals into strips with action costs. In *Proc. of the 5th Workshop on Knowledge Engineering for Planning and Scheduling (ICAPS-2014)*, 23–30.

Coles, A. J., and Coles, A. 2011. LPRPG-P: Relaxed plan heuristics for planning with preferences. In *Proceedings of 21st Internaltional Conference on Automated Planning and Scheduling ICAPS'11)*.

Cresswell, S., and Coddington, A. M. 2004. Compilation of LTL goal formulas into PDDL. In *Proc. of the 16th European Conference on Artificial Intelligence ECAI*, 985–986.

Do, M., and Kambhampati, S. 2004. Partial satisfaction (over-subscription) planning as heuristic search. In *Proc. of 5th Int. Conf. on Knowledge Based Computer Systems (KBCS'04)*.

Edelkamp, S.; Jabbar, S.; and Nazih, M. 2006. Large-scale optimal pddl3 planning with mips-xxl. In *In 5th International Planning Competition Booklet (ICAPS-06)*.

Edelkamp, S. 2006. On the compilation of plan constraints and preferences. In *In Proceedings of ICAPS-06*, 374–377.

Gazen, B. C., and Knoblock, C. A. 1997. Combining the expressiveness of UCPOP with the efficiency of Graphplan. In Steel, S., and Alami, R., eds., *Recent Advances in AI Planning: 4th European Conference on Planning, ECP'97*. New York: Springer-Verlag.

Gerevini, A.; Haslum, P.; Long, D.; Saetti, A.; and Dimopoulos, Y. 2009. Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners. *Artificial Intelligence* 173(5-6):619–668.

Katz, M., and Hoffmann, J. 2014. Mercury planner: Pushing the limits of partial delete relaxation. In *In 8th International Planning Competition Booklet (ICAPS-14)*.

Keyder, E., and Geffner, H. 2009. Soft goals can be compiled away. *J. Artif. Intell. Res. (JAIR)* 36:547–556.

Richter, S., and Westphal, M. 2010. The lama planner: Guiding cost-based anytime planning with landmarks. *J. Artif. Int. Res.* 39(1):127–177.

Rintanen, J. 2000. Incorporation of temporal logic control into plan operators. In *Proc. of the 14th European Conference on Artificial Intelligence ECAI*, 526–530. IOS Press.

Rintanen, J. 2015. Personal communication.

Smith, D. 2004. Choosing objectives in over-subscription planning. In *Proc. of 14th Int. Conf. on Automated Planning and Scheduling (ICAPS'04)*.

Weld, D., and Etzioni, O. 1994. The first law of robotics (a call to arms). In *Proc. of AAAI-94*, 1042–1047.