Semi-Relaxed Plan Heuristics

Emil Keyder, Joerg Hoffmann, Patrik Haslum

Semi-Relaxed Plan Heuristics

Emil Keyder
INRIA
Nancy, France
emilkeyder@gmail.com

Jörg Hoffmann
Saarland University
Saarbrücken, Germany
hoffmann@cs.uni-saarland.de

Patrik Haslum
The Australian National University & NICTA
Canberra, Australia
patrik.haslum@anu.edu.au

Abstract

Heuristics based on the delete relaxation are at the forefront of modern domain-independent planning techniques. Here we introduce a principled and flexible technique for augmenting delete-relaxed tasks with a limited amount of delete information, by introducing special fluents that explicitly represent conjunctions of fluents in the original planning task. Differently from previous work in this direction, conditional effects are used to limit the growth of the task to be linear, rather than exponential, in the number of conjunctions that are introduced, making its use for obtaining heuristic functions feasible. We discuss how to obtain an informative set of conjunctions to be represented explicitly, and analyze and extend existing methods for relaxed planning in the presence of conditional effects. The resulting heuristics are empirically evaluated, and shown to be sometimes much more informative than standard delete-relaxation heuristics.

Introduction

Planning as heuristic search is one of the most successful approaches to planning. Some of the most informative heuristics in both the optimal planning setting, in which heuristics must be admissible, and the satisficing setting, in which there is no such requirement, are obtained as the estimated cost of the delete relaxation of the original planning task (Helmert and Domshlak 2009; Bonet and Geffner 2001; Hoffmann and Nebel 2001). This relaxation simplifies the task by assuming that every variable value, once achieved, persists during the execution of the rest of the plan.

While such heuristics are often informative, it is desirable to be able to take into account delete information. Some previous work in this direction has focused on local search for *low-conflict* relaxed plans, while still considering the underlying delete-relaxation problem (Baier and Botea 2009). We instead look for inspiration in the admissible h^m family of heuristics (Haslum and Geffner 2000), which rather than obtaining estimates by considering single fluents, consider the costs of *conjunctions* of fluents of size $\leq m$. The h^m heuristics provide the guarantee that there exists m such that $h^m = h^*$. Realizing this guarantee, however, comes at a large computational cost, as the number of conjunctions that must be considered is exponential in m. Furthermore, the required m is often large, as the cost of a conjunction (e. g., the goal) is estimated as the cost of its most costly subset of size $\leq m$, ignoring the cost of the remaining fluents.

The h^m heuristic has recently been recast as the $h^{\max}$ cost of a planning task Π^m with *no deletes* (Haslum 2009). This is achieved by representing conjunctions c of size $\leq m$ in the original task with new fluents π_c, called π-*fluents*, and modifying the operators of the planning task to have these fluents as preconditions or add effects as appropriate. The Π^m compilation is not useful, however, for obtaining more informative heuristic estimates, as $h^+(\Pi^m)$ is not admissible. The more recent Π^C construction (Haslum 2012) (hereafter HPC) fixes this issue, at the cost of growth *exponential* in the number of π-fluents.[1] Π^C also offers the possibility of a tradeoff between representation size and heuristic accuracy, by allowing the choice of an *arbitrary* set of conjunctions C and corresponding π-fluents, while treating the remaining fluents in the task as in the standard delete relaxation. This stands in contrast to the h^m heuristic and the Π^m compilation, in which *all* conjunctions of size $\leq m$ are represented.

While the Π^C task has the potential to yield extremely informative heuristics, its use in practice is precluded by its exponential growth in the size of C. This severely limits the number of conjunctions that can feasibly be considered. Here, we introduce a related construction Π^C_{ce} that is similar to Π^C, but that makes use of conditional effects to limit the growth of the task to be *linear* in $|C|$. This exponential gain in size comes at the price of some information loss relative to Π^C. However, as we show, Π^C_{ce} is still perfect in the limit: there always exists a set of conjunctions C such that $h^+(\Pi^C_{\mathsf{ce}}) = h^*$. Furthermore, there exist families of tasks for which Π^C_{ce} can represent the same heuristic function as Π^C in exponentially less space.

To determine whether these theoretical properties can translate into an improvement in planning performance, two issues must be addressed: how to choose the conjunctions in C so as to maximize the information gained from their addition to the planning task, and how to solve the resulting relaxed planning task with conditional effects so as to obtain an informative heuristic. We discuss our solutions to

[1] HPC focuses on solving Π^C optimally to obtain incremental lower bounds on the cost of an optimal plan, here we consider the use of these constructions to obtain heuristic functions.

these questions, and evaluate the resulting *partial relaxation heuristics* on a wide range of planning benchmarks, showing that for satisficing planning, they can significantly improve on the state of the art. For optimal planning, i. e., admissible approximations of $h^+(\Pi^C_{\mathsf{ce}})$, we discuss some major issues that arise from the presence of conditional effects; addressing these comprehensively is a topic for future work.

Background

Our planning model is based on the propositional STRIPS formalization, to which we add action costs and conditional effects. States and operators are defined in terms of a set of propositional variables, or fluents, with a *state* $s \subseteq F$ given by the set of fluents that are true in that state. A *planning task* is described by a 4-tuple $\Pi = \langle F, A, I, G\rangle$, where F is a set of such variables, A is the set of actions, $I \subseteq F$ is the initial state, and $G \subseteq F$ describes the set of goal states, given by $\{s \mid G \subseteq s\}$. Each action $a \in A$ consists of a 4-tuple $\langle \mathsf{pre}(a), \mathsf{add}(a), \mathsf{del}(a), \mathsf{ce}(a)\rangle$ and a cost $\mathsf{cost}(a) \in \mathbb{R}^+_0$. Here, $\mathsf{pre}(a)$, $\mathsf{add}(a)$, and $\mathsf{del}(a)$ are subsets of F; $\mathsf{ce}(a) = \{\mathsf{ce}(a)_1, \ldots, \mathsf{ce}(a)_n\}$ denotes a set of conditional effects, each of which is a triple $\langle \mathsf{c}(a)_i, \mathsf{add}(a)_i, \mathsf{del}(a)_i\rangle$ of subsets of F. If $\mathsf{ce}(a) = \emptyset$ for all $a \in A$, we say that Π is a *STRIPS planning task*.

An action a is *applicable* in s if $\mathsf{pre}(a) \subseteq s$. The result of applying it is $s[a] = (s \setminus (\mathsf{del}(a) \cup \bigcup_{\{i \mid \mathsf{c}(a)_i \subseteq s\}} \mathsf{del}(a)_i)) \cup (\mathsf{add}(a) \cup \bigcup_{\{i \mid \mathsf{c}(a)_i \subseteq s\}} \mathsf{add}(a)_i)$. A plan is a sequence of actions $\sigma = a_1, \ldots, a_n$ such that applying it in I results in a goal state. The *cost* of σ is $\sum_{i=1}^n \mathsf{cost}(a_i)$, with an *optimal* plan σ^* being a plan with minimal cost.

A *heuristic* for Π is a function h mapping states of Π into $\mathbb{R}^+_0$. The *perfect heuristic* h^* maps each state s to the cost of an optimal plan for s. A heuristic h is *admissible* if $h(s) \leq h^*(s)$ for all s. By $h(\Pi')$, we denote a heuristic function for Π whose value in s is given by estimating the cost of the corresponding state s' in a modified task Π'. We specify Π' in terms of the transformation of $\Pi = \langle F, A, I, G\rangle$ into $\Pi' = \langle F', A', I', G'\rangle$; s' is obtained by applying to s the same transformation used to obtain I' from I. To make explicit that h is a heuristic computed on Π itself, we write $h(\Pi)$.

The delete relaxation Π^+ of a planning task is obtained by discarding the delete effects in all actions and conditional effects. Formally, $\Pi^+ = \langle F, A^+, I, G\rangle$, where $A^+ = \{\langle \mathsf{pre}(a), \mathsf{add}(a), \emptyset, \mathsf{ce}^+(a)\rangle \mid a \in A\}$, where $\mathsf{ce}^+(a) = \{\langle \mathsf{c}(a)_i, \mathsf{add}(a)_i, \emptyset\rangle \mid \mathsf{ce}(a)_i \in \mathsf{ce}(a)\}$, and each action $a^+ \in A^+$ has the same cost as the corresponding action a in A. The optimal relaxation heuristic h^+ for Π is defined as the cost $h^*(\Pi^+)$ of an optimal plan for Π^+.

We denote the powerset of F with $\mathcal{P}(F)$. As in the introduction, in the context of Π^C and Π^C_{ce} we often refer to the fluent subsets $c \in C$ as *conjunctions*.

The Π^m, Π^C and Π^C_{ce} Compilations

Π^m was the first compilation to consider the computation of heuristics similar to h^m in a delete relaxation context, and works by introducing fluents π_c for each set $\{c \in \mathcal{P}(F) \mid |c| \leq m\}$ (Haslum 2009). The fluents π_c are added to any fluent set in the task (like an action precondition) that contains the associated set c. Furthermore, *representatives* of each action a are added to the task to model the situation in which the elements of a set of fluents f of size $\leq m-1$ are already true when a is applied, and a adds the fluents in $c \setminus f$ while deleting no fluent in f, thereby making every fluent in c, and therefore π_c, true.

The non-admissibility of $h^*(\Pi^m) = h^+(\Pi^m)$ is due to the construction of these representatives. Sets of fluents that are simultaneously made true with a single application of an action a in Π may require several representatives of a to explicitly achieve the same effect in Π^m. Consider for example an action a adding a fluent p in a state in which q and r are already true. In Π this makes the fluents p, q, and r true simultaneously, whereas in Π^2, two different representatives of a are required: one with $f = \{q\}$ adding $\pi_{\{p,q\}}$, and one with $f = \{r\}$ adding $\pi_{\{p,r\}}$.

The Π^C compilation solves this problem by instead creating an *exponential* number of representatives of a, each of which corresponds to an application of a making a *set* of π-fluents true (HPC). In the above example, separate representatives of a are introduced for each of the π-fluent sets $\emptyset$, $\{\pi_{\{p,q\}}\}$, $\{\pi_{\{p,r\}}\}$, and $\{\pi_{\{p,q\}}, \pi_{\{p,r\}}\}$, and the representative resulting from the last of these can be applied to make the two π-fluents true simultaneously. Π^C also differs from Π^m in that it allows the *choice* of a set $C \subseteq \mathcal{P}(F)$, and introduces fluents π_c for only those $c \in C$ rather than for all subsets of size at most m. In what follows, given a set of fluents $X \subseteq F$, we use the shorthand $X^C = X \cup \{\pi_c \mid c \in C \wedge c \subseteq X\}$. In other words, X^C consists of the set of fluents X itself, together with all fluents π_c representing $c \in C$ such that $c \subseteq X$.

Definition 1 (The Π^C compilation) *Given a STRIPS planning task $\Pi = \langle F, A, I, G\rangle$ and a set of conjunctions $C \subseteq \mathcal{P}(F)$, Π^C is the planning task $\langle F^C, A^C, I^C, G^C\rangle$, where A^C contains an action $a^{C'}$ for each pair $a \in A$, $C' \subseteq C$ such that $\forall c' \in C'$,*

(1) $\mathsf{del}(a) \cap c' = \emptyset \wedge \mathsf{add}(a) \cap c' \neq \emptyset$, *and*

(2) $\forall c \in C((c \subseteq c' \wedge \mathsf{add}(a) \cap c \neq \emptyset) \implies c \in C')$,

and $a^{C'}$ is given by $\mathsf{del}(a^{C'}) = \emptyset$,[2] $\mathsf{ce}(a^{C'}) = \emptyset$, *and*

$$\mathsf{pre}(a^{C'}) = (\mathsf{pre}(a) \cup \bigcup_{c' \in C'} (c' \setminus \mathsf{add}(a)))^C$$

$$\mathsf{add}(a^{C'}) = (\mathsf{add}(a) \cup (\mathsf{pre}(a) \setminus \mathsf{del}(a)))^C \cup \{\pi_{c'} \mid c' \in C'\}$$

The representatives $a^{C'}$ of a enforce that no element of the sets $c' \in C'$ is deleted, and require that the fluents that are elements of any $c' \in C'$ but that are not added by a be true before $a^{C'}$ can be executed. Constraint (2) ensures a form of non-redundancy: if $a^{C'}$ adds a π-fluent $\pi_{c'}$, then it also adds all π-fluents π_c such that $c \subseteq c'$, as all fluents in c necessarily become true with the application of the action.

[2] As defined by HPC, the actions in Π^C also have delete effects, ensuring that real (non-relaxed) plans correspond to plans in the original task. Since we only consider delete relaxations here, this does not concern us.

Π^C enumerates all possible subsets of C and therefore grows exponentially in $|C|$. This exponentiality is reminiscent of the canonical conditional effects compilation used to convert planning tasks with conditional effects into classical STRIPS planning tasks with exponentially more actions (Gazen and Knoblock 1997). The Π^C_{ce} compilation that we introduce here is the result of applying roughly the reverse transformation to Π^C, resulting in a closely related planning task that has a number of conditional effects *linear* in $|C|$:

Definition 2 (The Π^C_{ce} compilation) *Given a STRIPS planning task $\Pi = \langle F, A, I, G\rangle$ and a set of conjunctions $C \subseteq \mathcal{P}(F)$, Π^C_{ce} is the planning task $\langle F^C, A^C_{\mathsf{ce}}, I^C, G^C\rangle$ where*

$$A^C_{\mathsf{ce}} = \{\langle \mathsf{pre}(a^C), \mathsf{add}(a^C), \mathsf{del}(a^C), \mathsf{ce}(a^C)\rangle \mid a \in A\},$$

and a^C is given by

$$\begin{aligned}
\mathsf{pre}(a^C) &= \mathsf{pre}(a)^C \\
\mathsf{add}(a^C) &= (\mathsf{add}(a) \cup (\mathsf{pre}(a) \setminus \mathsf{del}(a)))^C \\
\mathsf{del}(a^C) &= \emptyset \\
\mathsf{ce}(a^C) &= \{\langle (\mathsf{pre}(a) \cup (c \setminus \mathsf{add}(a)))^C, \{\pi_c\}, \emptyset\rangle \\
&\quad \mid c \in C \wedge c \cap \mathsf{del}(a) = \emptyset \wedge c \cap \mathsf{add}(a) \neq \emptyset\}
\end{aligned}$$

Rather than enumerating the sets of π-fluents that may be made true by an action, Π^C_{ce} uses conditional effects to implicitly describe the conditions under which *each* is made true. The only information lost in doing so is the information encoded by *cross-context* π-fluents in preconditions, which appear in action representatives in Π^C, but not in the preconditions or conditions of the corresponding actions in Π^C_{ce}. For action representatives $a^{C'}$ in Π^C, these are π-fluents $\pi_y \in \mathsf{pre}(a^{C'})$ such that there is no *single* $c \in C'$ for which $y \subseteq (c \setminus \mathsf{add}(a)) \cup \mathsf{pre}(a)$. Considering our example above, $\pi_{\{q,r\}}$ is a precondition for the action representative that adds both $\pi_{\{p,q\}}$ and $\pi_{\{p,r\}}$ in Π^C, but does not appear as a condition in any of the conditional effects of the action in Π^C_{ce}. Since effect conditions are determined individually for each π_c, such conditions are never included. We will return to this below when discussing the theoretical relationship between Π^C and Π^C_{ce}.

Example 1 *Consider the STRIPS planning task (adapted from Helmert and Geffner (2008)) with variables $\{x_0, \ldots, x_n, y\}$, initial state $I = \{x_0, y\}$, goal $G = \{x_n\}$, and actions*

$$a : \langle \emptyset, \{y\}, \emptyset, \emptyset\rangle \qquad b_i : \langle \{x_i, y\}, \{x_{i+1}\}, \{y\}, \emptyset\rangle$$

for $i = 0, \ldots, n-1$.

The optimal solution to this planning task takes the form $b_0, a, b_1, a, \ldots, b_{n-1}$, and has cost $2n-1$. In the delete relaxation of the task, the fact that y is deleted after each application of b_i is ignored, and the optimal plan has cost n.

When a π-fluent $\pi_{x_i,y}$ is introduced in the Π^C_{ce} compilation, it is added to the precondition of the action b_i, and a new conditional effect $\mathsf{ce}(a)_i$ of the form $\langle\{x_i\}, \{\pi_{\{x_i,y\}}\}, \emptyset\rangle$ is created for action a. No conditional effects are added to any of the actions b_i, as each deletes y and therefore cannot be an achiever of the π-fluent. This increases the optimal delete relaxation cost of the task by 1, as an instance of a must be added to the relaxed plan to achieve the newly introduced precondition of b_i. If all π-fluents of the form $\pi_{\{x_i,y\}}$ are introduced, the delete relaxation cost of Π^C_{ce} becomes $2n-1$, the optimal cost. While h^2 would also give the optimal cost of this problem, its computation would require the consideration of $\Theta(n^2)$ fluent pairs rather than the linear number of π-fluents used here.

An important practical optimization for both Π^C and Π^C_{ce} concerns mutex information. If such information about the original planning task is available, then action representatives and conditional effects created by the compilation that have unreachable preconditions or conditions can be discarded with no loss of information.[3]

Theoretical Properties of Π^C_{ce}

Here we outline some theoretical properties of Π^C_{ce}, considering the cost $h^+(\Pi^C_{\mathsf{ce}})$ of its optimal solutions instead of more practical approximations (note that for Π^C_{ce} and the version of Π^C considered here, $h^+ = h^*$ as no delete effects are present). We first show two fundamental properties:

Proposition 1 (Consistency and admissibility) *$h^+(\Pi^C_{\mathsf{ce}})$ is consistent and admissible.*

Proof sketch: Consistency follows from the fact that Π^C_{ce} defines a state space in which the cost of a state $h^+(\Pi^C_{\mathsf{ce}})(s')$ is necessarily associated with a plan $\sigma^*(s')$. Given s, a such that $s[a] = s'$, $a \cdot \sigma^*(s')$ then necessarily constitutes a plan for s, and therefore $h^+(\Pi^C_{\mathsf{ce}})(s) \leq cost(a) + h^+(\Pi^C_{\mathsf{ce}})(s')$. Admissibility follows from consistency. ■

Proposition 2 ($h^+(\Pi^C_{\mathsf{ce}})$ dominates $h^+(\Pi)$) *Given a planning task Π and a set of conjunctions C, $h^+(\Pi^C_{\mathsf{ce}}) \geq h^+(\Pi)$. There are cases where the inequality is strict.*

Proof sketch: This follows from the fact that any plan for Π^C_{ce} is also a plan for Π^+, yet the inverse is not the case. ■

We now consider the relationship between the Π^C and Π^C_{ce} compilations. As mentioned above, information encoded by *cross-context* preconditions is lost when moving from the exponential Π^C to the linear Π^C_{ce}. Estimates obtained from Π^C_{ce} may therefore be inferior to those obtained from Π^C:

Proposition 3 ($h^+(\Pi^C)$ dominates $h^+(\Pi^C_{\mathsf{ce}})$) *Given a planning task Π and a set of conjunctions C, $h^+(\Pi^C) \geq h^+(\Pi^C_{\mathsf{ce}})$.*

Proof sketch: The claim follows directly from the fact that the standard conditional effects compilation of Π^C_{ce} (Gazen and Knoblock 1997) is equivalent to Π^C except for the presence of the cross-context preconditions discussed above. ■

[3] If enough π-fluents were added to the compilation, this mutex information would be detected during the heuristic computation itself, as the relevant π-fluents would become unreachable. Including all sets of fluents of size 2, for example, would lead to all h^2 mutexes being found. Exploiting available mutex information allows us to avoid the addition of unnecessary π-fluents, and thus helps to keep the compilation small.

Proposition 4 ($h^+(\Pi^C)$ may strictly dominate $h^+(\Pi^C_{\mathsf{ce}})$) *There exist planning tasks Π and sets of conjunctions C such that $h^+(\Pi^C) > h^+(\Pi^C_{\mathsf{ce}})$.*

Proof sketch: Consider the planning task with fluent set $F = \{p_1, p_2, r, g_1, g_2\}$, initial state $I = \{p_1\}$, goal $G = \{g_1, g_2\}$, and actions

$$a_{p_2} : \langle\{p_1\}, \{p_2\}, \{r, p_1\}, \emptyset\rangle \quad a_r : \langle\emptyset, \{r\}, \emptyset, \emptyset\rangle$$
$$a_{g_1} : \langle\{p_1, r\}, \{g_1\}, \emptyset, \emptyset\rangle \quad a_{g_2} : \langle\{p_2, r\}, \{g_2\}, \emptyset, \emptyset\rangle$$

Let C be the set of all subsets of F of size 2. The only optimal plan for both Π and Π^C is the sequence $\langle a_r, a_{g_1}, a_{p_2}, a_r, a_{g_2}\rangle$. However, $\langle a_{p_2}, a_r, a_{g_1}, a_{g_2}\rangle$ is a plan of lower cost for Π^C_{ce}. This plan takes advantage of the fact that $\pi_{\{p_1,r\}}$ and $\pi_{\{p_2,r\}}$ can be simultaneously achieved by the action a_r, using two different conditional effects which have the conditions p_1 and p_2, without making true the (unreachable) cross-context π-fluent $\pi_{\{p_1,p_2\}}$. ∎

The choice of C as all conjunctions of size 2 in Proposition 4 implies that there exist tasks in which it is necessary to consider strictly larger conjunctions in order to obtain equally good heuristic estimates with Π^C_{ce} as are obtained with Π^C. This is not necessarily problematic however, as differently from h^m, the sizes of Π^C_{ce} and Π^C are not exponential in the size of the conjunctions considered.

The advantage of Π^C_{ce} over Π^C is that it is exponentially smaller in $|C|$; the above "domination" therefore must be qualified against this reduction in size. Furthermore, Π^C_{ce} preserves the ability to compute a perfect heuristic given a sufficiently large set C of conjunctions. We first consider the equivalent result for Π^C (HPC):

Theorem 1 ($h^+(\Pi^C)$ is perfect in the limit) *Given a planning task Π, there exists C such that $h^+(\Pi^C) = h^*(\Pi)$.*

Proof sketch: It is known that $h^m(\Pi) = h^*(\Pi)$ for sufficiently high values of m (Haslum and Geffner 2000), and as shown by Haslum (2009), $h^1(\Pi^m) = h^m(\Pi)$. It can easily be demonstrated that for $C = \{c \in \mathcal{P}(F) \mid |c| \leq m\}$, $h^1(\Pi^C) = h^1(\Pi^m)$. Choosing an appropriate m and the corresponding C, we then have that $h^*(\Pi) = h^m(\Pi) = h^1(\Pi^m) = h^1(\Pi^C) \leq h^+(\Pi^C) \leq h^*(\Pi)$, with the last inequality following from the admissibility of Π^C. ∎

This proof is different from that given (implicitly) in HPC. We use it here as it can be conveniently adapted to show that Π^C_{ce} preserves this property:

Theorem 2 ($h^+(\Pi^C_{\mathsf{ce}})$ is perfect in the limit) *Given a planning task Π, there exists C such that $h^+(\Pi^C_{\mathsf{ce}}) = h^*(\Pi)$.*

Proof sketch: To show this, we first show that for any set of conjunctions C, $h^1(\Pi^C) = h^1(\Pi^C_{\mathsf{ce}})$. This is because a minimum cost h^1 path in Π^C need not make use of any action that adds more than one π-fluent, since the critical path passes through single fluents in the task. Therefore, cross-context π-fluents do not play a role. The claim follows from this fact and the proof of Theorem 1. ∎

The proofs of Theorems 1 and 2 rely on obtaining perfect h^m, which is clearly unfeasible in general since this involves enumerating all subsets of fluents (and hence all possible states). However, Π^C and Π^C_{ce} offer *flexibility*, in allowing us to choose the set C: while selecting *all* subsets guarantees a perfect heuristic, this may be achieved with much less effort, especially when using Π^C_{ce}, whose growth in $|C|$ is linear. Indeed, there are task families for which obtaining h^* takes exponential effort with h^m, and requires exponentially-sized Π^C, yet for which Π^C_{ce} remains small:

Proposition 5 (Expressive power of Π^C_{ce} vs. h^m and Π^C) *There exist parametrized task families Π_k such that (a) if $h^m(\Pi_k) = h^*(\Pi_k)$ then $m \geq k$; (b) $h^+((\Pi_k)^C) = h^*(\Pi_k)$ implies $(\Pi_k)^C$ has $\geq 2^k$ action representatives; (c) for any k there exists C_k such that $|C_k| \leq k \cdot \alpha$, where α is a constant for the family Π_k, and $h^+((\Pi_k)^C_{\mathsf{ce}}) = h^*(\Pi_k)$.*

Proof sketch: Members Π_k of one such family are given by the combination of k planning tasks of the type in Example 1, each of size n. Π_k has k goals, and $h^m = h^*$ iff $m \geq k + 1$. For both $(\Pi_k)^C$ and $(\Pi_k)^C_{\mathsf{ce}}$ to be optimal, n π-fluents for each of the individual subtasks, and therefore kn π-fluents in total, must be introduced. The number of conditional effects in $(\Pi_k)^C_{\mathsf{ce}}$ is then linear in k, but the number of action representatives in $(\Pi_k)^C$ is exponential. ∎

In practice, of course, our heuristic is not usually perfect, and we instead try to select a set C that yields an informative heuristic with a reasonably sized representation.

Practical Aspects of Using Π^C_{ce}

We now turn to the practical issues involved with using the Π^C_{ce} compilation to obtain a heuristic for the original planning task. There are two questions to be answered: How to obtain heuristic estimates from delete-free planning tasks with conditional effects, and how to choose the set C.

LM-cut

We first consider optimal planning with admissible approximations of $h^+(\Pi^C_{\mathsf{ce}})$. The state-of-the-art approach to approximating h^+ is the LM-cut algorithm (Helmert and Domshlak 2009). However, it cannot be directly applied to the Π^C_{ce} task due to the presence of conditional effects, for which its behaviour is undefined. This turns out to be a formidable obstacle, as there is no straightforward extension to the algorithm that preserves its two fundamental properties, (i) admissibility and (ii) domination of $h^{\max}$.

For (ii), consider a planning task Π with a single action a that has two conditional effects $\mathsf{ce}(a)_1 = \langle\{p\}, \{q\}, \emptyset\rangle$ and $\mathsf{ce}(a)_2 = \langle\{q\}, \{r\}, \emptyset\rangle$, initial state $\{p\}$, and goal $\{r\}$. We have $h^+(\Pi) = h^{\max}(\Pi) = 2$ due to the critical path $\langle a, a\rangle$, and the justification graph considered by LM-cut consists of this same sequence. The first cut found is $\{a\}$. When the cost of a is reduced, the remaining task has $h^{\max}$ cost 0, resulting in a cost estimate of 1.

The problem here is that different conditional effects of an action may be part of the same critical path. A natural approach is therefore to reduce costs per individual conditional effect, rather than for all of the effects of the action at once. Unfortunately, it turns out that this does not preserve admissibility. Indeed, it is possible to construct a STRIPS task Π

whose Π^C_{ce} compilation has the following properties. All except a single action a have no conditional effects, and a has exactly two. Reducing the cost of a globally when it is first encountered in a cut leads to a heuristic estimate that is less than $h^{\max}(\Pi^C_{\mathsf{ce}})$, while treating the effects separately leads to an estimate greater than $h^+(\Pi^C_{\mathsf{ce}})$=$h^*(\Pi)$. There is therefore no strategy based on considering effects individually that preserves both (i) and (ii) on all planning tasks. Since admissibility cannot be sacrificed, we must reduce costs globally. This means that despite Theorem 2, $h^{\text{LM-cut}}(\Pi^C_{\mathsf{ce}})$ does *not* converge to $h^*(\Pi)$. This could of course be fixed by using $\max(h^{\max}, h^{\text{LM-cut}})$ as the heuristic value, yet as $h^{\max}$ is typically not informative, this strategy is not useful in practice.

Even if every action in the original task need be applied at most once, in the Π^C_{ce} compilation critical paths may contain multiple occurences of the same action. This can lead to situations in which the addition of a π-fluent *decreases* the LM-cut estimate. Consider a task with goal $G = \{p, q\}$, initial state $I = \emptyset$, and unit-cost actions $A = \{a, b\}$ which add p and q respectively, with no preconditions. When LM-cut is used on Π^+, both landmarks $\{a\}$ and $\{b\}$ are easily discovered, giving the correct estimate 2. After the introduction of the fluent $\pi_{\{p,q\}}$, however, its $h^{\max}$ value of 2 is higher than that of each of p and q (1), and $\pi_{\{p,q\}}$ is the goal fluent selected by the *precondition choice function* in LM-cut. Each of a and b can achieve $\pi_{\{p,q\}}$, leading to the cut $\{a, b\}$. The cost of both actions is then reduced to 0, resulting in the overall cost estimate 1.

On the IPC benchmarks, optimal planning performance is worse with $h^{\text{LM-cut}}(\Pi^C_{\mathsf{ce}})$ than with $h^{\text{LM-cut}}(\Pi)$ in all but a few cases. It remains an open question whether this can be improved.

Non-Admissible Approximations

The problem of finding sub-optimal relaxed plans for planning tasks with conditional effects has previously been considered (Hoffmann and Nebel 2001). Here, we refine and extend those techniques. This is especially important because (unlike in most IPC benchmarks) the structure of the conditional effects introduced in Π^C_{ce} can be rather complex, with multiple dependencies between different actions and even between different executions of the same action.[4]

Non-admissible delete-relaxation heuristics are typically obtained from a *best-supporter* function $\mathsf{bs} : F \mapsto A$ with the intuition that $\mathsf{bs}(p)$ is an action adding p that minimizes the cost of making p true. This function is used in combination with a *relaxed plan extraction algorithm*, which when no conditional effects are present, computes a *set* of actions σ that form a relaxed plan for the planning task, as defined by the following rules (Keyder and Geffner 2008):

$$\sigma(p) = \begin{cases} \{\} & \text{if } p \in s \\ \mathsf{bs}(p) \cup \sigma(\mathsf{pre}(\mathsf{bs}(p))) & \text{otherwise} \end{cases}$$

$$\sigma(P) = \bigcup_{p \in P} \sigma(p)$$

[4]We remark that similar issues arise in approaches compiling uncertainty into classical planning with conditional effects (Palacios and Geffner 2009; Bonet, Palacios, and Geffner 2009).

Existing methods for choosing best supporters, such as h^{add} or $h^{\max}$, can easily be extended to conditional effects by treating each conditional effect in the task as a separate action. In particular, this is the method employed (using $h^{\max}$) by the heuristic used in FF (Hoffmann and Nebel 2001): for each relaxed conditional effect $\mathsf{ce}(a)^+_i$ with condition $\mathsf{c}(a)_i$ and add $\mathsf{add}(a)_i$, an action a_i with the same add effect $\mathsf{add}(a_i) = \mathsf{add}(a)_i$ and precondition $\mathsf{pre}(a_i) = \mathsf{pre}(a) \cup \mathsf{c}(a)_i$ is created. The resulting set of effects $E = \sigma(G)$ then form a relaxed plan. However as a single action execution may trigger several of its conditional effects, there may exist a relaxed plan that uses fewer occurrences of an action than implied by E. The question then arises of how to optimally schedule the plan so as to minimize the number of action applications required. FF uses a simple heuristic solution to this problem that we outline and improve upon below, but we first note that the problem is actually **NP**-complete:

Proposition 6 (Scheduling conditional relaxed plans) *Let Π^+ be a relaxed planning task with conditional effects and E a set of effects that, viewed as a set of independent actions, constitutes a plan for Π^+. Deciding whether there exists a sequence of actions of length $\leq k$ such that all conditional effects in E are triggered is **NP**-complete.*

Proof sketch: Membership is obvious. Hardness follows by reduction from the shortest common supersequence problem (Garey and Johnson 1979). The fluents encode the current position within each sequence. Each conditional effect moves forward from one particular position, and each action groups together all effects whose position bears one particular symbol in the respective sequence. ∎

Note that Proposition 6 does *not* relate to the (known) hardness of optimal relaxed planning: we wish only to schedule effects that we have already selected and which we know to form a relaxed plan. This source of complexity has, as yet, been overlooked in the literature.

Due to this hardness result, we use a greedy minimization technique that nevertheless gives good results. Starting with the trivial schedule containing one action execution for each effect in E, we considers pairs of effects $e, e' \in E$ of the same action a. The two effects are merged into a single execution of a if their conditions can be achieved without the use of either of the effects. In contrast, FF merges e and e' when they appear in the same layer of the relaxed planning graph. This criterion is sound as the conditions of such effects are necessarily independently achievable without using either, yet is less general than the technique that we use here, as the same may also be the case for effects in *different* layers of the relaxed planning graph. We capture this independence between effects with the *best supporter graph* (BSG) representation of the relaxed plan, assuming a single goal fluent G' and, if necessary, an action END that has as preconditions the original goals of the task and adds G':

Definition 3 (Best supporter graph) *Given a relaxed planning task Π^+ and a best supporter function* bs*, the* best supporter graph *is a directed acyclic graph $\phi = \langle V, E \rangle$, where $V = \sigma(G)$, with $\sigma(G)$ as above,*

$E = \{\langle v, v' \rangle \mid \exists p \in \mathsf{pre}(v') \wedge v = \mathsf{bs}(p)\}$, *and each edge is labelled with the precondition* p *that gives rise to the edge.*

The nodes of this graph represent conditional effects that appear in the relaxed plan, and there exists an edge $\langle v, v' \rangle$ between two nodes if the effect represented by v is the best supporter of a (pre)condition of the effect represented by v'. ϕ is an acyclic graph and therefore has at least one valid topological sort, and it can easily be shown that any such sort of ϕ is a valid relaxed plan. This implies that, if there is no path in the BSG between two conditional effects of the same action, they can occur as the result of the same action application, and therefore can be merged into a single occurrence of the action. These nodes are then removed from the BSG, and a new node is added that represents both effects, combining their incoming and outgoing edges. This process can be repeated until no further node merges are possible. This algorithm is sound in that it results in a relaxed plan for Π; it is suboptimal, and necessarily so due to Proposition 6.

Finally, an important optimization is *eliminating dominated preconditions*. When a fluent π_c is introduced as the precondition of an action or a condition of a conditional effect, all fluents $p \in c$ and π-fluents $\{\pi_{c'} \mid c' \subseteq c\}$ are removed from that precondition: achieving π_c implies that they are necessarily made true as well, and counting their cost separately would lead to an overestimation. A particular case is that where the fluent sets represented by several different π-fluents have a non-empty intersection, yet none is a subset of the other. Consider for example an action a with $\mathsf{pre}(a) = \{p, q, r\}$. If $C = \{\{p, q\}, \{q, r\}\}$, then $\mathsf{pre}(a) = \{\pi_{\{p,q\}}, \pi_{\{q,r\}}\}$, and the cost of achieving q will implicitly be counted twice when calculating the cost of applying a. (We experimented with a fix treplacing overlapping π-fluents $\pi_c, \pi_{c'}$ with $\pi_{c \cup c'}$, yet found this to not generally improve performance.)

Strategies for Choosing C

Our general strategy for choosing a set of conjunctions is shown in Algorithm 1: Relaxed plans are repeatedly generated for the initial state of the current Π^C_{ce} and new conjunctions are added to C based on how the current relaxed plan fails on the original planning task Π. The process stops either when no further conflicts can be found, implying that the current relaxed plan is also a plan for the original planning task, or when a user-specified bound on the size of Π^C_{ce} is reached. In our experiments, we will express this bound in terms of the size of Π^C_{ce} compared to Π. We will also sometimes consider a bound in the runtime taken by the algorithm. Algorithm 1 is complete if no bound is specified and FindConflicts(σ) returns at least one conjunction unless σ already is a plan for Π. Algorithm 1 is also optimal when the relaxed planning method used is optimal.

In choosing the conjunctions to be added to C, we adapt the strategy outlined in HPC to our purposes. They consider only optimal plans and represent them by means of the *relaxed plan dependency graph* (RPDG), which is similar to the BSG considered above except in a few particulars. The RPDG graph encodes only *necessary* orderings between actions: There is a path from a node v_a to a node v_b if and only if a precedes b in *every* valid sequencing of σ. Disjunctive dependencies in which one of several actions adding a precondition must be applied before another are therefore not captured, and there may exist topological sorts of RPDGs that are not valid relaxed plans. In contrast, the BSG encodes information about the "intentions" of the relaxed plan heuristic in the form of the chosen best supporters, sometimes imposing orderings that need not be respected in every valid sequencing of the plan (for example, when a fluent p is added by an action in the plan that is not its best supporter). The property of introducing only necessary orderings in the graph is not required by the conflict detection method discussed below, so we use the BSG instead.

Algorithm 1: Generating C for Π^C_{ce}.

```
C = ∅
σ = RelaxedPlan(Π^C_ce)
while σ not a plan for Π and size(Π^C_ce) < bound do
    C = C ∪ FindConflicts(σ)
    σ = RelaxedPlan(Π^C_ce)
```

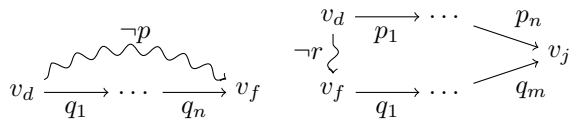

Figure 1: Relaxed plan failure scenarios. Wavy edges show deletions of a precondition.

As all preconditions of actions in a relaxed plan σ are made true at some point, the failure of σ implies that some action d, the *deleter*, deletes the precondition of some other action f, the *failed* action. There are two possibilities for this to happen, depicted in Figure 1. One corresponds the case in which there is a path in the BSG from d to f, and the other to the case in which there is no such path. HPC show that the addition of the set of π-fluents $\bigcup_{i=1}^{n} \{\pi_{\{p,q_i\}}\}$ in the first case, and the addition of the set $\bigcup_{i=1}^{n} \bigcup_{j=1}^{m} \{\pi_{\{p_i,q_j\}}\}$ in the second, ensures that the current relaxed plan σ no longer constitutes a relaxed plan for Π^C. If either or both of the fluents p, q are π-fluents, the resulting fluent π_c represents the union of the fluents represented by both, and possibly has size $|c| > 2$. This notion of progress is important in their setting as computing the cost $h^+(\Pi^C)(I)$ of an optimal plan after each addition is very costly.

There are a number of differences between our setting and that of HPC. In particular, "progress" is not as well-defined, since we use Π^C_{ce} to generate heuristic estimates for all states during search, not just for the initial state in which the π-fluents are collected. We have observed that it is highly beneficial to instead add conjunctions one at a time, introducing just a single π-fluent in each iteration of Algorithm 1. This fluent is $\{p, q_n\}$ in the case depicted in Figure 1a and $\{p_n, q_m\}$ in that of (b). Intuitively, this works better because conflicts found in the same round tend to be redundant and needlessy grow the size of the task, leading to slow evaluation times without much gain in informativeness.

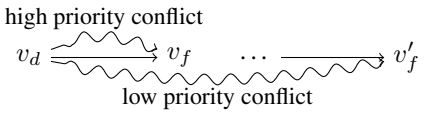

Figure 2: High/low priority conflicts. Wavy edges show deletions of a precondition.

Experiments

We evaluated the performance of relaxed plan heuristics obtained from h^{add} best supporters for different growth bounds x on the size of Π^C_{ce}. When $x = 1$, no π-fluents or conditional effects are introduced and $\Pi^C_{\text{ce}} = \Pi^+$, resulting in a standard relaxed plan heuristic. For growth bounds $x > 1$, π-fluents are introduced until the number of conditional effects in the task is $\geq (x-1) \cdot |A|$. The compilation and associated heuristic were implemented in the Fast Downward planner (Helmert 2006), and used in conjunction with greedy best-first search with lazy evaluation and a second open list for states resulting from preferred operators, the operators in the relaxed plan applicable in the current state. Action costs were ignored, as taking them into account tends to be detrimental to coverage (Richter and Westphal 2010), the maximization of which is the primary aim in satisficing planning. The resulting planners were tested on the 14 domains of the 2011 International Planning Competition (IPC). All experiments were run on Intel Xeon 2.67 GHz computers with the settings used in the competition: a memory limit of 6Gb and a time limit of 30 minutes.

Π^C vs. Π^C_{ce}. We have not implemented Π^C, but the number of actions that would be induced by a set of π-fluents C in Π^C can be inferred from the conditional effects found in Π^C_{ce} compilations. Even with $x = 1.5$, this number runs into the millions or billions for larger problems in several domains, and there are 3 domains out of the 14 considered in which it causes overflow in a 32-bit integer for at least one task. In the barman domain for example, 19 of the problems would have more than 10^7 actions, and 13 would have more than 10^8. This confirms our hypothesis that the conditional effects representation is in general necessary for scaling to large numbers of π-fluents.

Conflict selection. The informativeness of the Π^C_{ce} compilation is very sensitive to the choice of π-fluents added. Our strategy is to first introduce π-fluents for conflicts that lie along a single path in the relaxed plan (Figure 1 (a)), and to only consider conflicts arising from two parallel paths if none of the former are present (Figure 1 (b)). We also prioritize conflicts by distance: The shorter the path from an action deleting a precondition to the action whose precondition is deleted, the higher the conflict priority (Figure 2).

Improved informativeness. Relaxed plans obtained from Π^C_{ce} with $x > 1$ are more informative than standard relaxed plans in several domains, most notably in Barman, Floortile, Parcprinter, and Woodworking (Table 1). The difference is most striking in the Floortile domain, in which the relaxed plan obtained with $x = 2.5$ is 27000 times more informative than the standard relaxed plan heuristic, as measured by the median ratio of heuristic evaluations, and $x = 2.5$ results in the solution of all 20 instances within 5 seconds. In comparison, only 7 instances are solved with $x = 1$, requiring the evaluation of several hundred thousand nodes, and the maximum number of instances solved in this domain by *any* planner participating in the IPC is 9. The added information leads to a drastic decrease in the number of heuristic evaluations required in order to find a plan, rendering it trivial (Figure 3). A possible explanation is that delete effects impose a fairly fixed ordering on the goals in this domain, and this ordering can be discovered with the addition of comparatively few π-fluents. Coverage also increases in the Barman and Parcprinter domains, but not in the Woodworking domain, in which the standard relaxed plan heuristic is already able to solve all 20 tasks. In most of the remaining domains, informativeness is slightly increased or does not significantly change, with the Tidybot domain being the lone exception.

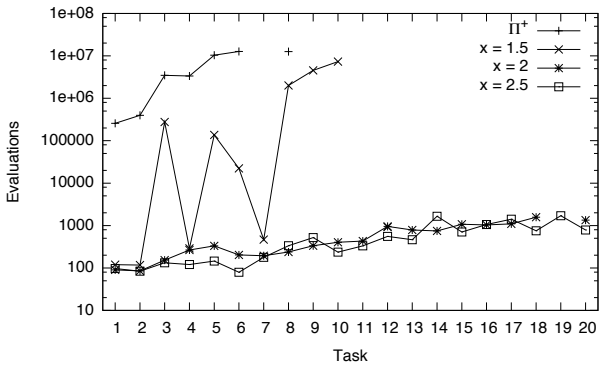

Figure 3: Heuristic evaluations in the Floortile domain.

Performance on older benchmarks. We have evaluated the performance of our planner with $x = 2$ on the full set of benchmarks from the previous IPCs. Its performance is generally similar to that seen on the domains shown here in detail, being more informative in most domains and slightly less informative in a few. In the Mystery domain, it is able to solve all 19 of the solvable instances with less than 25 evaluations and (soundly) prove the remaining 11 unsolvable in the initial state, making it the only heuristic we know of that is able to achieve this.

Computational overhead. Increasing the number of actions in the relaxed task comes at a computational cost. The slowdown in heuristic evaluation for the $x = 2.5$ case is shown in column **S** in Table 1. When this is not accompanied by an increase in heuristic informativeness, coverage with $\pi(\Pi^C_{\text{ce}})$ suffers, sometimes significantly. This behavior can be seen in the Openstacks, Parking, and Visitall domains, in which neither heuristic is informative and node evaluations in the ten thousands up to the millions are the norm.

The time required to select conflicts and iteratively construct Π^C_{ce} is usually negligible, but can be large in domains with very large relaxed plans in which recomputing a relaxed plan after introducing a conflict takes a long time, or each π-fluent induces few conditional effects. For $x = 2.5$, the time dedicated to the construction of Π^C_{ce} exceeds 60 seconds in Openstacks, Scanalyzer, Transport, and Visitall. In Visitall, the procedure does not terminate within 30 minutes for some of the larger tasks. The **+60s** column in Table 1 shows the

	Coverage							% fewer evals			Median			S
Domain	**1**	**1.5**	**2**	**+60s**	**2.5**	**+60s**	∞	**1.5**	**2**	**2.5**	**1.5**	**2**	**2.5**	**2.5**
Barman	18	**20**	19	19	**20**	**20**	0	88	94	94	10.56	7.34	31.53	2.22
Elevators	18	16	**19**	**19**	16	16	0	60	53	53	1.04	1.00	1.15	1.97
Floortile	7	11	19	19	**20**	**20**	8	100	100	100	571	22709	27702	4.07
Nomystery	**9**	**9**	8	8	**9**	**9**	6	55	57	63	1.27	2.00	2.23	1.76
Openstacks	**20**	18	16	18	17	18	0	55	50	35	1.35	1.09	0.94	2.86
Parcprinter	13	15	17	17	19	19	**20**	33	27	58	0.92	0.82	1.04	1.04
Parking	13	**14**	13	13	10	10	0	67	73	67	1.07	1.47	1.31	4.36
Pegsol	**20**	**20**	**20**	**20**	**20**	**20**	0	40	60	55	0.95	1.55	1.57	1.06
Scanalyzer	**20**	**20**	**20**	**20**	**20**	**20**	0	70	75	85	1.79	1.98	3.11	1.33
Sokoban	**18**	**18**	**18**	**18**	**18**	**18**	1	71	47	47	1.08	0.98	0.86	1.32
Tidybot	15	**16**	**16**	**16**	14	14	0	38	28	29	0.77	0.22	0.24	0.69
Transport	**10**	8	8	8	7	8	0	67	57	29	1.20	1.14	0.70	1.26
Visitall	**18**	14	12	17	12	17	0	64	58	58	1.06	1.09	1.07	2.02
Woodworking	**20**	**20**	**20**	**20**	**20**	**20**	**20**	95	95	95	216.95	220.94	224.80	4.31
Total	220	219	225	**232**	222	229	55							

Table 1: **Coverage** and heuristic evaluations comparison for IPC'11 domains containing 20 instances each, for different values of x. **+60s** indicates that the conflict detection phase was bounded by whichever was reached first, the bound on task size, or a maximum runtime of 60 seconds. The columns **% fewer evals** and **Median** compare to $x = 1$, and show the percentage of tasks solved with fewer evaluations, and the median of the ratios of heuristic evaluations for tasks solved with both heuristics, respectively. The last column shows the median **S**lowdown per heuristic evaluation.

effects on coverage of setting a time bound of 60 seconds on this procedure (in addition to the x bound).

Conditional effect merging. The impact of conditional effect merging is mixed. Overall, it increases coverage and informativeness, but in some domains it results in a less informative heuristic, especially in the Elevators and Transport domains where it decreases coverage by 1 and 2 problems respectively with $x = 2$.

The overhead of the procedure is quite small, as the transitive closure operation required to check whether there is a path between two nodes of the BSG can be implemented very efficiently when the graph is known to be directed acyclic, as is the case here.

Finding plans with no search. The ∞ column in Table 1 shows the number of tasks solved when no limit is imposed on the growth of Π^C_{ce} and the generated relaxed plan becomes a valid plan for Π. Of these domains, Parcprinter and Woodworking share the feature that plans can be decomposed into many smaller subplans which have little interaction with one another, in the sense that it rarely occurs that actions from one subplan delete preconditions in another. The flaws in each of the subplans can then be fixed independently of the others. The maximum growth in task size required to obtain a valid plan for any task in the Woodworking domain is $x = 1.38$,[5] while this value in Parcprinter is $x = 17$.

Contribution to the state of the art. Effective domain-independent planners increasingly rely on combinations of heuristics and planning techniques in order to obtain superior performance. The winner of the most recent IPC, LAMA-2011, uses two different heuristics in conjunction, and 3 of the other planners in the top 5 are portfolio planners. In this context, we note that the best possible portfolio, for IPC 2011 using LAMA and the techniques proposed here, would run LAMA for 1000 seconds and our compilation with $x = 2.5$ for 800, resulting in 267 of 280 tasks solved compared to LAMA's 250. Any one of the planners entered in the competition would gain at least 11 tasks in coverage from running search with Π^C_{ce} and $x = 2.5$ for just 10 seconds, from the Floortile domain alone.

[5]Informativeness continues to increase beyond $x = 1.5$ in Table 1 as, for the purpose of these experiments, we continued to add conjunctions even after a valid plan was found.

Conclusion

We have demonstrated a principled and flexible method for improving delete-relaxation heuristics with a limited amount of delete information. Similarly to the previously proposed Π^C compilation, our method generalizes both relaxed plan heuristics and the h^m family of heuristics, yet guarantees that the size of the task representation grows linearly, rather than exponentially, in the number of conjunctions that are explicitly represented. Our results show that the computational investment is worthwhile in certain domains, in one of which the heuristic leads to solutions for tasks that are not solved by any other planner.

Our work suggests a number of directions for future research, including more principled methods for selecting informative or optimal sets of conjunctions C to define Π^C_{ce}, and alternative search algorithms that make use of the conflict learning mechanism only when heuristic plateaus are encountered. The Π^C_{ce} compilation could also be used to encode domain-specific information about planning tasks obtained from other sources, such as goal orderings. An ordering $g_1 \prec g_2 \prec g_3$, for example, can be expressed by having all effects adding $\pi_{\{g_1,g_2,g_3\}}$ have $\pi_{\{g_1,g_2\}}$ as a condition, and those adding $\pi_{\{g_1,g_2\}}$ have g_1 as a condition.

Acknowledgments. Work performed while Jörg Hoffmann was employed by INRIA, Nancy, France.

References

Baier, J. A., and Botea, A. 2009. Improving planning performance using low-conflict relaxed plans. In Gerevini, A.; Howe, A.; Cesta, A.; and Refanidis, I., eds., *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*. AAAI Press.

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence* 129(1–2):5–33.

Bonet, B.; Palacios, H.; and Geffner, H. 2009. Automatic derivation of memoryless policies and finite-state controllers using classical planners. In Gerevini, A.; Howe, A.; Cesta, A.; and Refanidis, I., eds., *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*. AAAI Press.

Garey, M. R., and Johnson, D. S. 1979. *Computers and Intractability—A Guide to the Theory of NP-Completeness*. San Francisco, CA: Freeman.

Gazen, B. C., and Knoblock, C. 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)*, volume 1348 of *Lecture Notes in Artificial Intelligence*, 221–233. Toulouse, France: Springer-Verlag.

Haslum, P., and Geffner, H. 2000. Admissible heuristics for optimal planning. In Chien, S.; Kambhampati, R.; and Knoblock, C., eds., *Proceedings of the 5th International Conference on Artificial Intelligence Planning Systems (AIPS-00)*, 140–149. Breckenridge, CO: AAAI Press, Menlo Park.

Haslum, P. 2009. $h^m(P) = h^1(P^m)$: Alternative characterisations of the generalisation from $h^{\max}$ to h^m. In Gerevini, A.; Howe, A.; Cesta, A.; and Refanidis, I., eds., *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*, 354–357. AAAI Press.

Haslum, P. 2012. Incremental lower bounds for additive cost planning problems. In Bonet, B.; McCluskey, L.; Silva, J. R.; and Williams, B., eds., *Proceedings of the Twenty-second International Conference on Automated Planning and Scheduling (ICAPS 2012)*. AAAI Press.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In Gerevini, A.; Howe, A.; Cesta, A.; and Refanidis, I., eds., *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*, 162–169. AAAI Press.

Helmert, M., and Geffner, H. 2008. Unifying the causal graph and additive heuristics. In Rintanen, J.; Nebel, B.; Beck, J. C.; and Hansen, E., eds., *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling (ICAPS 2008)*, 140–147. AAAI Press.

Helmert, M. 2006. The Fast Downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14:253–302.

Keyder, E., and Geffner, H. 2008. Heuristics for planning with action costs revisited. In Ghallab, M., ed., *In Proc. ECAI 2008*, 588–592. Wiley.

Palacios, H., and Geffner, H. 2009. Compiling uncertainty away in conformant planning problems with bounded width. *Journal of Artificial Intelligence Research* 35:623–675.

Richter, S., and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research* 39:127–177.