Journal of Artificial Intelligence Research 50 (2014) 487-533 Submitted 12/13; published 06/14

Improving Delete Relaxation Heuristics Through Explicitly Represented Conjunctions

Emil Keyder EMILKEYDER@GMAIL.COM

Jörg Hoffmann HOFFMANN@CS.UNI-SAARLAND.DE
Saarland University
66123 Saarbrücken, Germany

Patrik Haslum PATRIK.HASLUM@ANU.EDU.AU
The Australian National University & NICTA
Canberra ACT 0200, Australia

Abstract

Heuristic functions based on the delete relaxation compute upper and lower bounds on the optimal delete-relaxation heuristic h^+, and are of paramount importance in both optimal and satisficing planning. Here we introduce a principled and flexible technique for improving h^+, by augmenting delete-relaxed planning tasks with a limited amount of delete information. This is done by introducing special fluents that explicitly represent conjunctions of fluents in the original planning task, rendering h^+ the perfect heuristic h^* in the limit. Previous work has introduced a method in which the growth of the task is potentially exponential in the number of conjunctions introduced. We formulate an alternative technique relying on conditional effects, limiting the growth of the task to be linear in this number. We show that this method still renders h^+ the perfect heuristic h^* in the limit. We propose techniques to find an informative set of conjunctions to be introduced in different settings, and analyze and extend existing methods for lower-bounding and upper-bounding h^+ in the presence of conditional effects. We evaluate the resulting heuristic functions empirically on a set of IPC benchmarks, and show that they are sometimes much more informative than standard delete-relaxation heuristics.

1. Introduction

Planning as heuristic search is one of the most successful approaches to planning. Some of the most informative heuristic functions for domain-independent planning are obtained as the estimated cost of the delete relaxation of the original planning task. The delete relaxation simplifies planning tasks by assuming that every variable value, once achieved, persists during the execution of the rest of the plan. The cost of an *optimal* plan for the resulting relaxed planning task, denoted h^+, is **NP**-complete to compute, however whether *some* plan for the delete-relaxed task exists can be checked in polynomial time (Bylander, 1994). For satisficing planning, where the heuristic does not have to be admissible, the latter fact can be exploited to upper-bound h^+, by generating some not necessarily optimal plan for the delete-relaxed task (Hoffmann & Nebel, 2001). For optimal planning, lower-bounding methods have been devised based on the analysis of landmarks, logical formulas over the set of actions that state necessary properties of delete-relaxed plans (Karpas &

Domshlak, 2009; Helmert & Domshlak, 2009). These cost estimates for the delete-relaxed task can then be used to guide heuristic search in the state space of the original task.

Since delete relaxation heuristics were first proposed (Bonet & Geffner, 2001), much work has been done to improve them. One approach focuses on better approximation schemes for h^+, obtaining tighter upper bounds and thus better non-admissible estimates (Hoffmann & Nebel, 2001; Keyder & Geffner, 2008, 2009), or tighter lower bounds that correspond to more informative admissible heuristics (Helmert & Domshlak, 2009; Bonet & Helmert, 2010). In many domains, however, it is important that the heuristic be able to take into account delete information (Hoffmann, 2005), and indeed there is a long tradition of works proposing heuristics that do so. Several of these extend the delete relaxation to capture strictly more information (Fox & Long, 2001; Helmert, 2006; Helmert & Geffner, 2008; Cai, Hoffmann, & Helmert, 2009; Katz, Hoffmann, & Domshlak, 2013), while some consider only the delete relaxation but attempt to find low-conflict relaxed plans (Baier & Botea, 2009), or generate modified heuristic values based on taking conflicts into account to some extent (Do & Kambhampati, 2001; Gerevini, Saetti, & Serina, 2003). Here, we approach this problem by taking inspiration from the admissible h^m family of heuristics (Haslum & Geffner, 2000). An important property of the heuristics we introduce, shared with some other recent work in this direction, is that our technique *renders h^+ the perfect heuristic h^* in the limit*. In other words, the technique offers a trade-off between the amount of delete information considered and the computational overhead of doing so. At one end of that continuum, delete-relaxed plans become plans for the original task.

The h^m heuristic function considers the cost of making true simultaneously *sets* of fluents of size $\leq m$. The cost of the planning task is then estimated by recursively taking the cost of a set of fluents, such as the goal or a set of action preconditions, to be the cost of its most costly subset of size $\leq m$, and ignoring the cost of achieving the remaining fluents in the set. As each possible subset of size m of the fluents in the task must be considered, the size of the representation required to compute h^m is exponential in m. The h^m heuristics provide the guarantee that there exists an m such that $h^m = h^*$ (trivially satisfied when m is the total number of fluents in the task). However, the value of m required to achieve this is usually so large as to make computing h^* with this method infeasible in practice.

The h^m heuristic has recently been recast as the $h^{\max} = h^1$ cost of a planning task Π^m with *no deletes* (Haslum, 2009). This is achieved by representing conjunctions of fluents c of size $\leq m$ in the original task with new fluents π_c, here called *π-fluents*, and modifying the initial state, goal, and operators of the planning task so as to capture the reasoning performed by h^m over these sets within the computation of $h^{\max}$. However, $h^+(\Pi^m)$ is not admissible (since a separate copy of the same action may be needed to establish each π-fluent), and thus the Π^m compilation is not useful for obtaining admissible estimates that are more informative than h^+. The more recent Π^C construction (Haslum, 2012) fixes this issue (introducing an action copy for every subset of π-fluents that may be established), at the cost of growing the task representation *exponentially* in the number of π-fluents rather than linearly as in the Π^m representation. On the other hand, Π^C offers the possibility of a more fine-grained tradeoff between representation size and heuristic accuracy, by allowing the choice of an *arbitrary* set of conjunctions C and corresponding π-fluents (which need not all be of the same size). This stands in contrast to the h^m heuristic and the Π^m compilation, in which *all* sets or conjunctions of size $\leq m$ are represented.

Haslum (2012) proposed to repeatedly solve Π^C optimally, within an iterative procedure that adds new conjunctions to the set C in each iteration. The relaxed plans that are computed therefore gradually become closer, in some sense, to being plans for the original task. We instead explore the idea of using this kind of construction for obtaining heuristic functions for guiding search.

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 worst case *linear*, rather than exponential, in $|C|$. This gain in size comes at the price of some information loss relative to Π^C. However, as we show, this information loss does not affect the fundamental property of tending towards the perfect heuristic h^* if enough conjunctions are introduced: Like Π^C, Π^C_{ce} is perfect in the limit, i. e. there always exists a set of conjunctions C such that $h^+(\Pi^C_{\mathsf{ce}}) = h^*$. Furthermore, while information may be lost, this is not always the case. Indeed, it is possible to construct families of planning tasks for which Π^C_{ce} can represent the same heuristic function as Π^C for the same set of conjunctions C, i. e., $h^+(\Pi^C_{\mathsf{ce}}) = h^+(\Pi^C)$, but for which the representation of Π^C_{ce} occupies exponentially less space.

Having said that, the theoretical advantage of Π^C_{ce} does not tend to materialize in practice (or at least in the commonly used benchmarks): While without further optimizations Π^C indeed grows too quickly to be practical, it turns out that *mutex pruning* techniques (eliminating compiled actions with conflicting preconditions) are extremely effective at keeping the size of Π^C at bay. We therefore consider both Π^C_{ce} and Π^C, evaluating their usefulness for devising improved heuristic functions. We focus on two main questions:

(A) How to obtain upper and lower bounds for h^+ in compiled tasks?

(B) How to choose a set of conjunctions C so as to maximize the information gained from their addition to the planning task?

In response to question (A), we analyze and extend three state-of-the-art methods for estimating h^+. In the satisficing setting (upper-bounding h^+), we consider the problem of finding low-cost relaxed plans that can be scheduled so as to minimize the cost of the sequence of actions required to trigger a given set of conditional effects, avoiding unnecessary repeated applications of the same action. This problem has been only scantily addressed in previous work. Here we show that the problem of optimal action scheduling for a given set of effects is **NP**-complete, and generalize the approximation technique used in the FF planner (Hoffmann & Nebel, 2001).

In the optimal setting (lower-bounding h^+), we consider the *LM-cut* heuristic (Helmert & Domshlak, 2009) as well as admissible heuristics based on fluent landmarks (Karpas & Domshlak, 2009). For the former, our findings are mostly negative: First, we show that even though the introduction of π-fluents cannot decrease $h^{\max}$ or h^+, which lower- and upper-bound LM-cut, respectively, the LM-cut heuristic value can decrease. Second, we show that neither of the two straightforward adaptations of the LM-cut algorithm to problems with conditional effects maintains both admissibility and domination of $h^{\max}$.[1] For the latter, we show that Π^C_{ce} can be used to generate more informative fluent landmarks. Recent work (Keyder, Richter, & Helmert, 2010) extracts landmarks from the Π^m task.

1. A more sophisticated adaptation of LM-cut, based on the idea of "context splitting", has recently been proposed by ? (?). It maintains both properties.

This allows the discovery of π-fluent landmarks corresponding to conjunctive landmarks in the original task, but suffers due to the large number of π-fluents that must be considered. The Π^C_{ce} compilation offers the possibility of discovering interesting conjunctive landmarks of unbounded size, while avoiding growing the size of the compilation unnecessarily.

In response to question (B), we devise a range of strategies depending on the purpose for which the Π^C_{ce} or Π^C compilation is to be used. Most of these are parameterized in terms of the allowed growth of the compiled task relative to the original task, and thus allow a trade-off between the informativeness of the heuristic and its computational overhead. We evaluate the resulting heuristics on a wide range of benchmarks from the International Planning Competition, varying the relevant algorithm parameters to determine their individual effect on performance. As the results show, in several domains our heuristics are much more informative than previous ones, leading to significantly improved performance.

We next define the basic concepts (Section 2), before moving on to the formal definition of the Π^C_{ce} compilation and the previously introduced Π^m and Π^C compilations (Section 3). In Section 4, we analyze Π^C_{ce} and its relation to Π^m and Π^C from a theoretical perspective. Section 5 discusses the practical issues that arise in using the compilations for the purpose of satisficing planning, and describes the obtained experimental results, while Section 6 does the same for the case of optimal planning. Finally, Section 7 summarizes the main points of the paper and indicates some possible future research directions.

2. Preliminaries

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 F of propositional variables, or *fluents*. A *state* $s \subseteq F$ is 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 $\langle \mathsf{pre}(a), \mathsf{add}(a), \mathsf{del}(a), \mathsf{ce}(a) \rangle$, where $\mathsf{pre}(a)$, $\mathsf{add}(a)$, and $\mathsf{del}(a)$ are subsets of F. The action has a cost $\mathsf{cost}(a) \in \mathbb{R}^+_0$. By $\mathsf{ce}(a) = \{\mathsf{ce}(a)_1, \dots, \mathsf{ce}(a)_n\}$, we denote the set of conditional effects of action a, each of which is a triple $\langle \mathsf{c}(a)_i, \mathsf{add}(a)_i, \mathsf{del}(a)_i \rangle$ of subsets of F. To simplify some of our notations, we require that $\mathsf{add}(a) \cap \mathsf{del}(a) = \emptyset$; we do not need to impose any restrictions on the deletes $\mathsf{del}(a)_i$ of conditional effects, because conditional effects will be used only within the delete relaxation. If $\mathsf{ce}(a) = \emptyset$ for all $a \in A$, Π has no conditional effects, and we say that it is a *STRIPS planning task*.

An action a is *applicable* in s if $\mathsf{pre}(a) \subseteq s$. The result of applying it is given by

$$s[a] = (s \setminus (\mathsf{del}(a) \cup \bigcup_{\{i \mid \mathsf{c}(a)_i \subseteq s\}} \mathsf{del}(a)_i)) \cup (\mathsf{add}(a) \cup \bigcup_{\{i \mid \mathsf{c}(a)_i \subseteq s\}} \mathsf{add}(a)_i)$$

A *plan for* s is a sequence of actions $\sigma = a_1, \dots, a_n$ whose application in s results in a goal state. The *cost* of σ is $\sum_{i=1}^n \mathsf{cost}(a_i)$. σ is *optimal* if its cost is minimal among all plans for s; we will often denote optimal plans with σ^*. A plan for I is also called a *plan for* Π, or simply a *plan*.

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. It is sometimes useful to make explicit that h is a heuristic computed on Π itself; we will denote that by $h(\Pi)$. Note that the modified task Π' is used *only* for the computation of the heuristic function. In particular, the actual search for a plan is performed in the state space of the original planning task Π.

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

We denote the power set of F with $\mathcal{P}(F) = \{c \mid c \subseteq F\}$. In the context of h^m, Π^C, and Π^C_{ce}, we refer to fluent subsets $c \in \mathcal{P}(F)$ as *sets* or *conjunctions* interchangeably. Throughout the paper, we assume that conjunctions are *non-unit*, i. e., $|c| > 1$.

A *landmark* in a planning task is a logical formula ϕ over the set of fluents F such that every valid plan σ makes ϕ true in some state (Hoffmann, Porteous, & Sebastia, 2004). *Orderings* between landmarks are statements about the order in which these states occur. A *natural ordering* $\phi_1 \prec_{\mathrm{n}} \phi_2$ means that if a state s_j satisfies ϕ_2, there is some state s_i occurring before s_j in which ϕ_1 is satisfied. A *necessary ordering* $\phi_1 \prec_{\mathrm{nec}} \phi_2$ means that ϕ_1 is always true in the state immediately before the state in which ϕ_2 becomes true, while a *greedy necessary ordering* $\phi_1 \prec_{\mathrm{gn}} \phi_2$ means that this relationship holds the *first* time that ϕ_2 is made true. Note that the necessary ordering $\phi_1 \prec_{\mathrm{nec}} \phi_2$ implies the greedy necessary ordering $\phi_1 \prec_{\mathrm{gn}} \phi_2$, but not vice versa. A *landmark graph* G is a directed graph whose nodes are landmarks, and whose labelled edges correspond to the known orderings between these landmarks.

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

The Π^m compilation (Haslum, 2009) was the first technique proposed that made use of the idea of π-fluents that explicitly represent conjunctions in the original task. Given a conjunction $c \subseteq F$, π_c is a new fluent $\pi_c \notin F$ unique to c, i. e., if $c \neq c'$ then $\pi_c \neq \pi_{c'}$. In defining Π^m and the other compilations that we discuss, we use the shorthand $X^C = X \cup \{\pi_c \mid c \in C \wedge c \subseteq X\}$, where $X \subseteq F$ is a set of fluents, and $C \subseteq \mathcal{P}(F)$ is a set of conjunctions. In other words, X^C consists of the set of fluents X itself, together with new fluents π_c whose intention is to represent the conjunctions $c \in C$ that are contained in X, $c \subseteq X$.

Definition 1 (The Π^m compilation) *Given a STRIPS planning task* $\Pi = \langle F, A, I, G\rangle$ *and a parameter* $m \in \mathbb{Z}^+$, Π^m *is the planning task* $\langle F^C, A^C, I^C, G^C\rangle$, *where* $C = \{c \mid c \subseteq F \wedge 1 < |c| \leq m\}$, *and* A^C *contains* A *as well as an action* a^c *for each pair* $a \in A$, $c \in C$ *such that* $\mathsf{del}(a) \cap c = \emptyset$ *and* $\mathsf{add}(a) \cap c \neq \emptyset$, *and* a^c *is given by* $\mathsf{del}(a^c) = \emptyset$, $\mathsf{ce}(a^c) = \emptyset$, *and*

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

The parameter m here indicates the maximum size of the conjunctions to be represented explicitly in the resulting compiled task. A π-fluent is inserted (by definition of F^C, cf. above) for each $c \subseteq F$ where $1 < |c| \leq m$. These π_c are then added to all fluent sets in the task (such as the initial state, action preconditions, and goals) containing the associated set c. Furthermore, a *linear* (in $|C|$) number of *representatives* of each action a are added to the task to model the situation in which the elements of c that are not made true by a are already true before a is applied, and a adds the remaining fluents in c while deleting none of them, thereby making every fluent in c, and therefore π_c, true. This compilation allows the admissible h^m cost of the original task to be computed as the $h^{\max}$ cost of this compiled task.

The non-admissibility of $h^*(\Pi^m) = h^+(\Pi^m)$ is due to the construction of the action representatives a^c: 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 $c = \{p, q\}$ adding $\pi_{\{p,q\}}$, and one with $c = \{p, r\}$ adding $\pi_{\{p,r\}}$.

The Π^C compilation solves this problem by instead creating a number of representatives of a exponential in the number of π-fluents which may be made true by a. Each of these representatives corresponds to an application of a that makes a *set* of π-fluents true (Haslum, 2012). Following the above example, separate representatives of a would be 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 could 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:[2]

Definition 2 (The Π^C compilation) *Given a STRIPS planning task $\Pi = \langle F, A, I, G \rangle$ and a set of non-unit 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* $\mathsf{del}(a^{C'}) = \emptyset$, $\mathsf{ce}(a^{C'}) = \emptyset$, *and*

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

2. There are three differences between our definition and Haslum's (2012) definition of the the actions in Π^C. First, Haslum's definition features delete effects, ensuring that real (non-relaxed) plans correspond to plans in the original task. Since we only consider delete relaxations of the compiled task, we can safely omit these. Second, we allow the sets C' used in the construction of actions to contain conjunctions c with $c \subseteq \mathsf{add}(a) \cup (\mathsf{pre}(a) \setminus \mathsf{del}(a))$; and third, $\mathsf{add}(a^{C'})$ contains the π-fluents π_c where $c \subseteq \mathsf{pre}(a) \setminus \mathsf{del}(a)$. These latter two differences keep our definitions simpler. The redundant action representatives and redundant add effects that they cause are easily pruned in practice.

The representatives $a^{C'}$ of a enforce, for every $c' \in C'$, that no part of c' is deleted, and that the non-added part of c' is true already before $a^{C'}$ is 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. Note that, differently than Π^m, add effects in Π^C include π-fluents representing conjunctions of fluents added by the action with prevail fluents (non-deleted preconditions). This is necessary for admissibility of h^+ (the primary purpose of Π^C), but not needed for the computation of h^1 (the primary purpose of Π^m).

Π^C enumerates all possible subsets of C in constructing the representatives of each action 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 & 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 with a *linear* (in $|C|$) number of conditional effects:

Definition 3 (The Π^C_{ce} compilation) *Given a STRIPS planning task $\Pi = \langle F, A, I, G\rangle$ and a set of non-unit 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

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

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 effect conditions of the corresponding actions in Π^C_{ce}. For action representatives $a^{C'}$ in Π^C, these are π-fluents $\pi_y \in \mathsf{pre}(a^{C'})$ where there exists no $c \in C'$ s.t. $y \subseteq (c \setminus \mathsf{add}(a)) \cup \mathsf{pre}(a)$. In the situation discussed above, for example, $\pi_{\{q,r\}}$ is a precondition for the action representative that adds both $\pi_{\{p,q\}}$ and $\pi_{\{p,r\}}$ in Π^C, but does not appear in the condition of any conditional effects of the corresponding 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 & Geffner, 2008) with variables $\{x_0, \ldots, x_n, y\}$, initial state $I = \{x_0, y\}$, goal $G = \{x_n\}$, and unit-cost actions*

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

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

The optimal solution to this planning task takes the form $b_0, a, b_1, a, \dots, b_{n-1}$, 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 new conditional effects $\mathsf{ce}(a)_i$ of the form $\langle \{x_i\}, \{\pi_{\{x_i,y\}}\}, \emptyset \rangle$ are created for action a. No conditional effects are added to any of the b-actions, as each deletes y and therefore cannot be an achiever of any π-fluent. This increases the optimal delete relaxation cost of the task by 1, as a new 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.

The same set of conjunctions renders the delete relaxation cost of Π^C perfect (i. e., $2n-1$). However, the size of Π^C given that conjunction set is exponential in n: As action a may in principle achieve any subset of the conjunctions, every such subset C' induces a separate representative $a^{C'}$ in A^C.

Regarding the Π^m compilation, $h^2 = h^{\max}(\Pi^2)$ also gives the optimal cost of this task. However, its computation requires the consideration of $\Theta(n^2)$ fluent pairs, rather than the linear number of π-fluents that need to be introduced in Π^C_{ce}. As we shall see below (Theorem 6), the example can be easily extended so that m must scale with n for h^m to become perfect, thus showing an exponential separation between Π^C_{ce} and both Π^C and Π^m.

An important practical optimization for both Π^C and Π^C_{ce} is *mutex pruning*. If mutex information about the original planning task is available, specifically if we are given (some) m-tuples of fluents that are not reachable in conjunction, then we can discard from the compiled task any action representatives and conditional effects which require such an m-tuple, without losing admissibility of the compilation. Namely, the value of $h^+(\Pi^C)$ (respectively $h^+(\Pi^C_{\mathsf{ce}})$) after this mutex pruning is bounded from above by the value of $h^+(\Pi^{C'})$ (respectively $h^+(\Pi^{C'}_{\mathsf{ce}})$) for a larger set $C' \supseteq C$ of conjunctions: If we include all π-fluents of size at most m, then all h^m mutexes are found, i. e., none of the respective π-fluents is reachable in the compiled task. Exploiting available mutex information allows us to make the compilation more informed without having to add all these additional π-fluents, helping to keep the compilation small.

Another optimization we use is to *eliminate dominated preconditions*. Whenever we add a fluent π_c to the precondition of an action, or to the condition of a conditional effect, we remove from that condition all fluents $p \in c$ and π-fluents $\{\pi_{c'} \mid c' \subseteq c\}$. That is because achieving π_c implies achieving these fluents as well, and methods that count their cost separately (such as, for example, h^{add} and related heuristics) would incur an overestimation.

Note, however, that this does not eliminate the duplication caused by π-fluents representing different fluent sets that have a non-empty intersection. 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 in the h^{add} estimate of the cost of applying a. As a possible solution, we considered replacing overlapping π-fluents $\pi_c, \pi_{c'}$ with $\pi_{c \cup c'}$. This, however, did not consistently improve any of the heuristics that we compute from the compiled tasks.

4. Theoretical Properties of Π^C_{ce}

We now discuss 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). Where only proof sketches are shown, the full proofs can be found in Appendix A. We first show the fundamental and expected property:

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

Proof: Regarding consistency, given s, a such that $s[a] = s'$ in Π, we need to show that $h^+(\Pi^C_{\mathsf{ce}})(s) \leq \mathit{cost}(a) + h^+(\Pi^C_{\mathsf{ce}})(s')$. Let $\sigma^*(s'^C)$ be an optimal plan for s'^C in Π^C_{ce}. Then $a^C \cdot \sigma^*(s'^C)$ is necessarily a plan for s^C in Π^C_{ce}, as $s'^C \subseteq s^C[a^C]$ and Π^C_{ce} is a task with no deletes. Admissibility follows from consistency together with the fact that $h^+(\Pi^C_{\mathsf{ce}})(s) = 0$ on goal states s. ∎

Furthermore, the (ideal) delete relaxation lower bound can only improve as we add π-fluents:

Theorem 2 ($h^+(\Pi^C_{\mathsf{ce}})$ grows monotonically with C) *Given a planning task* Π *and sets* $C \supseteq C'$ *of non-unit conjunctions,* $h^+(\Pi^C_{\mathsf{ce}}) \geq h^+(\Pi^{C'}_{\mathsf{ce}})$.

Proof: This follows from the fact that given any plan $\sigma = a_1^C, \ldots, a_n^C$ for Π^C_{ce}, $\sigma' = a_1^{C'}, \ldots, a_n^{C'}$ constitutes a plan for $\Pi^{C'}_{\mathsf{ce}}$. We show by induction that $I^{C'}[a_1^{C'}] \ldots [a_i^{C'}] \supseteq I^C[a_1^C] \ldots [a_i^C] \setminus \{\pi_c \mid c \in C \setminus C'\}$, which shows the result since the goal of $\Pi^{C'}_{\mathsf{ce}}$ is $G^{C'} = G^C \setminus \{\pi_c \mid c \in C \setminus C'\}$, and $G^C \subseteq s^C[\sigma]$ if σ is a valid plan.

For $i = 0$, the induction hypothesis holds since $I^{C'} = I^C \setminus \{\pi_c \mid c \in C \setminus C'\}$ by definition. For $i > 0$, $a_i^{C'}$ is applicable in $I^{C'}[a_1^{C'}] \ldots [a_{i-1}^{C'}]$ since $\mathsf{pre}(a_i^{C'}) = \mathsf{pre}(a_i^C) \setminus \{\pi_c \mid c \in C \setminus C'\}$, and $I^{C'}[a_1^{C'}] \ldots [a_{i-1}^{C'}] \supseteq I^C[a_1^C] \ldots [a_i^C] \setminus \{\pi_c \mid c \in C \setminus C'\}$ by the induction hypothesis. For $\{\pi_c \mid \pi_c \in (I^C[a_1^C] \ldots [a_i^C] \setminus I^C[a_1^C] \ldots [a_{i-1}^C]) \wedge c \in C'\}$, either $\pi_c \in \mathsf{add}(a_i)^C$, which implies $\pi_c \in \mathsf{add}(a_i^{C'})$ due to the definition of $\Pi^{C'}_{\mathsf{ce}}$, or there exists some conditional effect $\mathsf{ce}_j(a_i^C) = \langle (\mathsf{pre}(a) \cup (c \setminus \mathsf{add}(a)))^C, \{\pi_c\}, \emptyset \rangle$. Since $c \in C'$, there must exist a corresponding conditional effect in $\Pi^{C'}_{\mathsf{ce}}$ by definition, and its condition must be true in $I^{C'}[a_1^{C'}] \ldots [a_{i-1}^{C'}]$ by the induction hypothesis. ∎

For the special case where $C' = \emptyset$, Theorem 2 gives us:

Corollary 1 ($h^+(\Pi^C_{\mathsf{ce}})$ dominates $h^+(\Pi)$) *Given a planning task* Π *and a set of non-unit conjunctions* C, $h^+(\Pi^C_{\mathsf{ce}}) \geq h^+(\Pi)$.

The domination can be strict, as follows trivially from convergence to h^* (Theorem 5 below).

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:

Theorem 3 ($h^+(\Pi^C)$ dominates $h^+(\Pi^C_{\mathsf{ce}})$) *Given a planning task* Π *and a set of non-unit conjunctions* C, $h^+(\Pi^C) \geq h^+(\Pi^C_{\mathsf{ce}})$. *There are cases in which the inequality is strict.*

Proof sketch: The standard conditional effects compilation into STRIPS (Gazen & Knoblock, 1997), applied to Π^C_{ce}, is equivalent to Π^C except for the presence of cross-context preconditions in Π^C. Given this, any plan for Π^C is also a plan for Π^C_{ce}, yet the inverse is not the case. To show the first part, we show by induction that $I^C[a_1^{C_1}, \dots, a_n^{C_n}] \subseteq I^C[a_1^C, \dots, a_n^C]_{\mathsf{ce}}$, where $I[\dots]_{\mathsf{ce}}$ denotes the result of applying a sequence of actions to the initial state I^C in Π^C_{ce}. Since the goal of both tasks is defined as G^C, this shows the desired result.

The strictness result follows from the fact that it is possible to construct tasks in which the cross-context preconditions discussed above play a role, leading to situations in which there exist plans for Π^C_{ce} that are shorter than the minimum-length plans for Π^C. ■

In the proof of strictness (Appendix A), we show a planning task for which the $h^+(\Pi^C)$ value is strictly larger than the $h^+(\Pi^C_{\mathsf{ce}})$ value when C is chosen to be *all* conjunctions of size 2. This implies that there exist tasks in which it is necessary to consider *strictly larger* conjunctions in Π^C_{ce} to obtain equally good heuristic estimates as are obtained with Π^C. This is not necessarily problematic however, as differently from h^m, Π^C_{ce} and Π^C do not introduce *all* conjunctions of a given size, and are therefore not exponential in the maximum size of the conjunctions considered.

The advantage of Π^C_{ce} over Π^C is that it is potentially 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, already proved by Haslum (2012). We provide an alternative proof here that can be conveniently adapted to show the same property for Π^C_{ce}. The key to our proof for Π^C is the following equivalence between $h^1(\Pi^m)$ and $h^1(\Pi^C)$:

Lemma 1 *Given a planning task* Π*, and* $C = \{c \in \mathcal{P}(F) \mid 1 < |c| \leq m\}$*,* $h^1(\Pi^m) = h^1(\Pi^C)$*.*

Proof sketch: Π^m and Π^C are identical except for their action sets. h^1 values are computed by considering only a single add effect at a time. The inequality $h^1(\Pi^m) \leq h^1(\Pi^C)$ is then easy to see by verifying that, for every add effect π_c of an action $a^{C'}$ in Π^C (unless $\pi_c \in \mathsf{pre}(a^{C'})$ and thus is redundant), the action a^c in Π^m *dominates* it, i. e., $\pi_c \in \mathsf{add}(a^c)$ and $\mathsf{pre}(a^c) \subseteq \mathsf{pre}(a^{C'})$. The proof is similar for the inequality $h^1(\Pi^C) \leq h^1(\Pi^m)$, observing that for any action a^c in Π^m and non-redundant add effect, there exists a dominating action $a^{C'}$ in Π^C. ■

Theorem 4 ($h^+(\Pi^C)$ is perfect in the limit) *Given a planning task* Π*, there exists* C *such that* $h^+(\Pi^C) = h^*(\Pi)$*.*

Proof: It is known that $h^*(\Pi) = h^m(\Pi)$ for sufficiently high values of m (Haslum & Geffner, 2000), and as shown by Haslum (2009), $h^m(\Pi) = h^1(\Pi^m)$. By Lemma 1, for $C = \{c \in \mathcal{P}(F) \mid 1 < |c| \leq m\}$, we have $h^1(\Pi^m) = h^1(\Pi^C)$. Choosing an appropriate m and the corresponding C, we thus have that $h^*(\Pi) = h^m(\Pi) = h^1(\Pi^m) = h^1(\Pi^C)$. Together with the fact that $h^1(\Pi^C) \leq h^+(\Pi^C)$, and since $h^+(\Pi^C) \leq h^*(\Pi)$ by admissibility of $h^+(\Pi^C)$, the claim follows. ■

To show the same claim for Π^C_{ce}, all that remains is to relate $h^1(\Pi^C)$ to $h^+(\Pi^C_{\mathsf{ce}})$:

Lemma 2 *Given a planning task* Π *and a set of non-unit conjunctions* C*,* $h^1(\Pi^C) \leq h^+(\Pi^C_{\mathsf{ce}})$*.*

Proof sketch: Consider a planning task $\Pi^C_{\text{no-cc}}$ identical to Π^C except that it drops cross-context π-fluents from preconditions. We show that (A) $h^1(\Pi^C) \leq h^1(\Pi^C_{\text{no-cc}})$, and (B) $h^1(\Pi^C_{\text{no-cc}}) \leq h^+(\Pi^C_{\mathsf{ce}})$.

Similarly to the proof of Lemma 1, (A) is easy to see by showing that every add effect π_c of an action $a^{C'}$ in $\Pi^C_{\text{no-cc}}$ is dominated by an action $a^{C''}$ in Π^C: we simply set C'' to the minimal subset of C' that contains c and satisfies condition (2) of Definition 2 (in other words, we reduce C' to get rid of any cross-context π-fluents).

For (B), it suffices to show that $h^+(\Pi^C_{\text{no-cc}}) \leq h^+(\Pi^C_{\mathsf{ce}})$. This holds because, for any action a in a relaxed plan for Π^C_{ce}, if C' is the set of conjunctions that are added by conditional effects of a when it is applied in the plan, then the action representative $a^{C'}$ in $\Pi^C_{\text{no-cc}}$ has the same preconditions as a, and can be used to achieve the same set of fluents. ∎

Theorem 5 ($h^+(\Pi^C_{\mathsf{ce}})$ is perfect in the limit) *Given a planning task* Π*, there exists* C *such that* $h^+(\Pi^C_{\mathsf{ce}}) = h^*(\Pi)$*.*

Proof: Choosing an appropriate m and C, we have $h^*(\Pi) = h^m(\Pi) = h^1(\Pi^m)$, and, by Lemma 1, $h^1(\Pi^m) = h^1(\Pi^C)$. With Lemma 2, we get $h^1(\Pi^C) \leq h^+(\Pi^C_{\mathsf{ce}})$. Since, by Theorem 1, $h^+(\Pi^C_{\mathsf{ce}}) \leq h^*(\Pi)$, this shows the claim. ∎

Note that, with Theorem 3, Theorem 4 is actually a corollary of Theorem 5. Our presentation is chosen to make the relation between the two results, and the role of the two lemmas, clearer.

The proofs of Theorems 4 and 5 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) in the worst case. 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, which is especially beneficial 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:

Theorem 6 (Expressive power of Π^C_{ce} vs. h^m and Π^C) *There exist parameterized task families* Π_k *such that*

1. *if* $h^m(\Pi_k) = h^*(\Pi_k)$ *then* $m \geq k$*,*
2. $h^+(\Pi^C_k) = h^*(\Pi_k)$ *implies that the number of action representatives in* Π^C_k *is exponential in* k*, and*
3. *for any* k *there exists* C_k *such that* $|C_k|$ *(and therefore the number of conditional effects in* $(\Pi_k)^C_{\mathsf{ce}}$*) is polynomial in* k*, and (b)* $h^+((\Pi_k)^C_{\mathsf{ce}}) = h^*(\Pi_k)$*.*

Proof: Members of one such family are given by the combination of k planning tasks of the type shown in Example 1, each of size k, that share among them the action a and the fluent y that needs to be made true after each step. Π_k then has k goals, and $h^m = h^*$ iff $m \geq k$.

For both Π^C_k and $(\Pi_k)^C_{\mathsf{ce}}$ to be perfect, k π-fluents $\{x_{i1}, y\}, \ldots, \{x_{ik}, y\}$ must be introduced for each of the individual subtasks i, leading to a total of k^2 π-fluents. If any one of these π-fluents is not present, then the precondition for the action b_{ij} in $(\Pi_k)^C_{\mathsf{ce}}$, or similarly its representative with $C' = \{\}$ in Π^C_k, have only the individual fluent preconditions y, x_{ij}, and in consequence one of the a actions reestablishing y can be left out of the plan. The number of conditional effects created in $(\Pi_k)^C_{\mathsf{ce}}$ is linear in the number of π-fluents added. However, the number of action representatives in $(\Pi_k)^C$ is exponential in k: the action a adds the fluent y, that belongs to all π-fluents, and hence a has one representative for each subset of the π-fluents. ■

When using $h^+(\Pi^C_{\mathsf{ce}})$ in practice, we will not typically be able to choose a C that results in a perfect heuristic. Instead, we try to pick a set C that yields an informative heuristic without making the size of the representation impractical to work with.

5. Heuristics for Satisficing Planning

We now consider the practical issues involved in using Π^C_{ce} for satisficing planning. Section 5.1 deals with the extraction of relaxed plans, and Section 5.2 deals with strategies for choosing the set of conjunctions C. Section 5.3 presents our experiments with the resulting setup.

5.1 Relaxed Planning with Conditional Effects

Techniques for extracting relaxed plans in the presence of conditional effects have long been known (Hoffmann & Nebel, 2001). Here, we refine and extend those techniques. They are particularly important in our context as, unlike in most IPC benchmarks, the structure of the conditional effects in Π^C_{ce} can be rather complex, involving multiple dependencies between different actions, and even between different executions of the same action.[3]

Non-admissible delete-relaxation heuristics are typically obtained with a *relaxed plan extraction algorithm* (Keyder & Geffner, 2008). The different variants of this algorithm are characterized by the *best-supporter* function $\mathsf{bs} : F \mapsto A$ they use. In all cases, $\mathsf{bs}(p)$ is an action adding p that minimizes some estimate of the cost of making p true. When no conditional effects are present, the algorithms compute a *set* of actions σ that can be scheduled to form a relaxed plan for the planning task. Formally, the algorithms construct a relaxed plan σ according to the following equations (Keyder & Geffner, 2008):

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

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

3. We remark that similar issues arise in approaches compiling uncertainty into classical planning with conditional effects (Palacios & Geffner, 2009; Bonet, Palacios, & Geffner, 2009), so our techniques may turn out to be useful there as well.

action. In particular, this is the method employed, using $h^{\max}$, to compute the FF heuristic function (Hoffmann & Nebel, 2001). More precisely, 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 $\mathsf{add}(a_i) = \mathsf{add}(a) \cup \mathsf{add}(a)_i$ and precondition $\mathsf{pre}(a_i) = \mathsf{pre}(a) \cup \mathsf{c}(a)_i$ is created. The set of effects $\sigma(G)$ as defined by the rules above then forms a relaxed plan. The presence of conditional effects, however, implies that there is a problem of *how to schedule that relaxed plan*: different schedules may require different numbers of action applications, as multiple applications of a single action a can be avoided by making the conditions of multiple desired effects true before a given application of a.

For illustration, consider a planning task where an action *move-briefcase* has n conditional effects, each of which conditionally transports an object from location A to location B if it is inside the briefcase. Using the representation above, a distinct moving action is generated for each conditional effect. So one possible schedule of the relaxed plan repeatedly puts an object in the briefcase, applies *move-briefcase*, then proceeds to the next object. This plan is $n-1$ steps longer than the optimal relaxed plan, which first places all of the objects in the briefcase and then applies *move-briefcase* once.

In other words, as a single action execution may trigger several conditional effects at once, there may exist a relaxed plan of length less than $|\sigma(G)|$. The question then arises of how to *optimally* schedule the relaxed plan, minimizing the number of action applications required. FF uses a simple approximate solution to this problem, that we outline and improve upon below. But we first note that the problem of scheduling conditional relaxed plans (SCRP) is actually **NP**-complete:

Theorem 7 (Scheduling conditional relaxed plans) *Let Π^+ be a relaxed planning task with conditional effects and $\sigma(G)$ 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 $\sigma(G)$ are triggered is **NP**-complete.*

Proof: Membership follows from the fact that given a sequence of k actions, it can easily be checked in polynomial time whether all conditional effects in $\sigma(G)$ are triggered. Hardness follows by reduction from the shortest common supersequence problem (SCS) (Garey & Johnson, 1979). A supersequence of a string $x = d_0 \dots d_m$ over the alphabet Σ is a string over the same alphabet that belongs to the language $L = \Sigma^* d_0 \Sigma^* \dots \Sigma^* d_m \Sigma^*$. Given an instance of the SCS problem with strings $x_0, \dots, x_n$ over the alphabet $\{0,1\}$ that asks whether there exists a supersequence of these strings with length $\leq k$, we construct a planning task with conditional effects $\Pi = \langle F, A, I, G \rangle$, where

- $F = \bigcup_{i=0}^{n} \{y_{ij} \mid 0 \leq j \leq |x_i|\}$
- $A = \{a_0, a_1\}$, where $a_z = \{\emptyset, \emptyset, \emptyset, \mathsf{ce}(a_z)\}$, and $\mathsf{ce}(a_z)$ is given by the set of conditional effects
$$\bigcup_{i=0}^{n} \bigcup_{j=0}^{|x_i|-1} \{\langle y_{ij}, y_{ij+1}, \emptyset \rangle \mid x_{ij} = z\}$$
- $I = \{y_{00}, \dots, y_{n0}\}$
- $G = \{y_{0|x_0|}, \dots, y_{n|x_n|}\}$

The two actions a_0 and a_1 correspond to the addition of the symbols 0 and 1 respectively to the supersequence that is implicitly being constructed, and a fluent y_{ij} encodes the fact that the current string constitutes a supersequence for a prefix $x_{i0}, \ldots, x_{ij-1}$. It can then be seen that a valid plan for the planning task must trigger all of the conditional effects of the task, yet such a sequence of actions with length $\leq k$ exists iff there is a common supersequence of $x_0, \ldots, x_n$ with length $\leq k$. This transformation of the SCS problem into a planning task with conditional effects is polynomial, which shows the claim. ∎

Note that Theorem 7 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.

Given this hardness result, we employ a greedy minimization technique that we call *conditional effect merging*. Starting with the trivial schedule containing one action execution for each effect in $\sigma(G)$, we consider pairs of effects $e, e' \in \sigma(G)$ that are conditional effects 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 their add effects. FF's approximation method applies similar reasoning, but captures only a special case in which this condition holds: when both e and e' appear in the same layer of the relaxed planning graph, which trivially implies that the conditions of these effects are independently achievable. However, the same may also be the case for effects in *different* layers of the relaxed planning graph. Here we devise a strictly more general technique, capturing this form of independence between effects using what we call the *best supporter graph* (BSG) representation of the relaxed plan (for simplicity, we assume here that the task has a single goal fluent G', which if needed can be achieved by introducing a new action END whose preconditions are the original goals, and that adds G'):

Definition 4 (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,* $E = \{\langle v, v'\rangle \mid \exists p \in \mathsf{pre}(v') \wedge v = \mathsf{bs}(p)\}$*, each vertex is labeled with the action whose conditional effect it represents, and each edge is labelled with the set of preconditions* $\{p \mid p \in \mathsf{pre}(v') \wedge v = \mathsf{bs}(p)\}$*.*

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'.[4] bs being a valid best supporter function (i. e., the relaxed plan $\sigma(G)$ generated by bs being sound) is a sufficient condition for ϕ being acyclic, and it can easily be shown that any topological sort of ϕ is a sound relaxed plan. This implies that, if there is no path in ϕ 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. The algorithm runs in polynomial time and is sound in that it results in a BSG of which any topological sort constitutes a relaxed plan for Π. It does not, however, guarantee an optimal scheduling of the original plan.

4. The edge labels will be used in our procedure choosing the conjunction set C, described in Section 5.2.

For example, consider again the task where *move-briefcase* has n conditional effects transporting an object from location A to location B if it is inside the briefcase. The nodes in the BSG are n *put-into-briefcase*(o_i) actions (one for each object o_i), as well as n copies of *move-briefcase*(A, B) (one for the conditional effect regarding each object o_i). There is one edge from *put-into-briefcase*(o_i) to the respective copy of *move-briefcase*(A, B), labeled with *in-briefcase*(o_i). There is therefore no path in the graph from any *move-briefcase*(A, B) node to another, and they will be merged into a single node by the conditional effect merging algorithm. All topological sorts of that merged BSG correspond to optimal relaxed plans.

5.2 Choosing C for Relaxed Planning

Algorithm 1 shows the main procedure for computing the set of conjunctions C used to form the Π^C_{ce} task. The algorithm is applied once, before the start of the search, to the initial state of the planning task. The resulting Π^C_{ce} (or Π^C) task is then used for all subsequent heuristic evaluations. Conditional effect merging is not used during the conflict extraction phase in any configuration discussed below, i. e., we use the original "non-merged" BSG as stated in Definition 4.

Algorithm 1: Choosing C for relaxed plan heuristics.

```
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)
```

Algorithm 1 is, at a high level, very similar to the procedure previously introduced for computing incremental cost lower bounds based on the Π^C construction (Haslum, 2012). The algorithm repeatedly generates relaxed plans for the initial state of the current compiled task. It adds new conjunctions to C based on the *conflicts* that are found in the current plan, i. e., based on how the current relaxed plan fails when executed in the original planning task Π. The process stops when either no further conflicts can be found, implying that the current relaxed plan for Π^C_{ce} is a plan for the original planning task, or when a user-specified bound on the size of Π^C_{ce} is reached. We will express this bound in terms of the size of Π^C_{ce} compared to Π (see below). We will also sometimes impose a bound on the runtime of the algorithm.

If no bound is specified, and if $\mathsf{FindConflicts}(\sigma)$ returns at least one new conjunction as long as σ is not a plan for Π, Algorithm 1 is a complete planning algorithm in its own right. We report results for this usage of the algorithm in our experiments below. If the relaxed plan generated in each iteration is optimal, Algorithm 1 can be used to compute a sequence of admissible cost estimates that converges to the optimal plan cost (Haslum, 2012). Our focus, however, is on the use of Π^C_{ce} for generating inadmissible heuristic functions. We therefore use a tractable, non-optimal, relaxed planning procedure, and impose a bound that typically stops Algorithm 1 before a plan for the original task has been found.

It remains to specify the $\mathsf{FindConflicts}$ procedure: Given a relaxed plan that fails to execute in the original planning task Π, how to select the set of new conjunctions C? One answer to this question has been provided by the previous use of Algorithm 1 to compute plan cost lower bounds (Haslum, 2012). Because our aim is different – computing heuristics for satisficing search-based planning – we make a number of changes to the previously proposed version of FindConflicts. Section 5.2.1 summarizes the original procedure, and Section 5.2.2 describes the changes we make to it.

5.2.1 Conflict Extraction for Incremental Plan Cost Lower Bounds

Given an optimal relaxed plan that is not a plan for the original planning task, Haslum's (2012) version of FindConflicts returns a set of conjunctions C that prevents the same relaxed plan from being a solution in the next iteration. This ensures "progress", in the sense that the cost of the relaxed plan will eventually increase, or prove to be the real plan cost. To describe the conflict extraction procedure, we need two definitions:

Definition 5 (Relaxed Plan Dependency Graph) *Let σ be a non-redundant plan for the relaxed planning task Π^+. Construct a directed graph $G^{\prec}(\sigma)$ with one node v_a for each action a in σ, plus a node v_G representing the goal. Let $\mathsf{pre}(v)$ denote the precondition of node v, which is $\mathsf{pre}(a)$ for a node v_a and G for node v_G. $G^{\prec}(S)$ has a directed edge from v_a to v' iff $\mathsf{pre}(v')$ is not relaxed reachable using the set of actions in σ minus $\{a\}$. The edge is labelled with the subset of $\mathsf{pre}(v')$ that is relaxed unreachable with these actions. The relaxed plan dependency graph,* RPDG(σ), *is the transitive reduction of $G^{\prec}(\sigma)$.*

The RPDG is similar to the BSG above (Definition 4), but encodes the *necessary* dependencies between actions in a relaxed plan. A path from a node v_a to a node v_b in the RPDG implies that a precedes b in *every* valid sequencing of σ; in this case, v_b is said to be *ordered after* v_a. In contrast, the BSG encodes the "intentions" of the relaxed plan heuristic, in the form of the chosen best supporters, and may impose orderings that need not be respected in every valid sequencing of the plan (e. g. if a fluent p is added by another action in the relaxed plan that is not the best supporter of p). Because the relaxed plan is non-redundant, meaning that no action can be removed without invalidating it, there is a path from every action node in the RPDG to the goal node.

Definition 6 (Dependency Closure) *Let σ be a non-redundant plan for the relaxed planning task Π^+, and let v and v' be nodes in* RPDG(σ), *where v' is ordered after v. A* simple dependency path *is a path $v \xrightarrow{q_1} v_1 \xrightarrow{q_2} \dots \xrightarrow{q_m} v'$ from v to v' in* RPDG(σ), *where each edge is labelled by one fluent, chosen arbitrarily, from the edge label in* RPDG(σ). *(Whenever v' is ordered after v, a simple dependency path from v to v' exists.) A* dependency closure *from v to v' is a minimal, w.r.t. subset, union of paths, such that (1) it contains a simple dependency path from v to v', and (2) if q is the fluent that labels an edge from a node v'' in the closure, and a is an action with $q \in \mathsf{add}(a)$, where a is not the action associated with v'', then the closure contains a simple dependency path from v to the node corresponding to a. (Such a path is guaranteed to exist.)*

Recall that input to FindConflicts is a plan, σ, that is valid for the delete relaxation Π^+ but not for the original planning task Π when delete effects are considered. Because σ is valid for Π^+, the preconditions of all actions in σ, as well as all goals, must be made true at

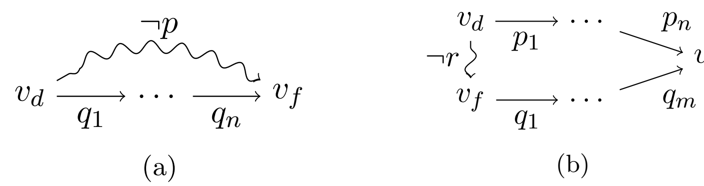


Figure 1: Relaxed plan failure scenarios. Wavy edges show deletions of a precondition.

some point. Thus, if σ fails to solve the original task Π it must be the case that some action d, which we call the *deleter*, deletes a precondition of some other action f, which we called the *failed* action. Note that the failed "action" here can also be the goal. Let $p \in pre(f)$ be the deleted fluent. The procedure distinguishes two cases, based on the relation between nodes v_d and v_f in the RPDG:

In the first case, illustrated in Figure 1 (a), v_f is ordered after v_d. Choose a dependency closure from v_d to v_f, and let L be the set of fluents labelling the edges in this closure: the set of conflicts generated is $\{\{p, q\} \mid q \in L\}$. (Note that $p \notin L$, and thus each conflict is a proper conjunction.)

If the first case does not hold, then v_d and v_f are unordered. They must have a nearest common descendant node, v_j, in the RPDG, so we are in the situation illustrated in Figure 1 (b). Choose a dependency closure from v_d to v_j, and let L_1 be the set of fluents labelling the edges in this closure. Likewise, choose a dependency closure from v_f to v_j, and let L_2 be the set of fluents labelling the edges in this closure. The set of conflicts generated is then $\{\{q, q'\} \mid q \in L_1, q' \in L_2 \cup \{p\}\}$.

Theorem 8 (Haslum, 2012, Theorem 6) *Let $\sigma = a_1, \ldots, a_n$ be a non-redundant plan for the delete relaxed task Π^+ that is not valid for the original task Π, and let C be a set of conjunctions extracted by the procedure described above. No action sequence $\sigma' = a'_1, \ldots, a'_n$ such that each a'_i is a representative of a_i is a valid plan for Π^C.*

5.2.2 Changes to Conflict Extraction for Satisficing Planning

There are a number of differences between our setting and that of Haslum (2012). In particular, although π-fluents are collected only in the initial state, the resulting Π^C_{ce} task will be used for heuristic evaluations of all states encountered in the search, and growth in the size of the Π^C_{ce} task will incur an overhead on each heuristic evaluation. Thus, our objective is to find a set C that will make the heuristic more accurate across all states, while keeping the size of C limited. On the other hand, computing non-optimal relaxed plans is computationally far cheaper than optimal relaxed planning, so we can afford more iterations in Algorithm 1.

Therefore, we make the following modifications to the strategy: First, we use the BSG instead of the RPDG. The necessity of orderings in the latter does not extend beyond the current (initial) state, and therefore is not useful for our purpose. The BSG is more representative of the relaxed plans found by the non-optimal relaxed planning procedure. Second, we introduce just a single π-fluent in each iteration of Algorithm 1. Most of the time, this does cause a new relaxed plan to be found, which allows the algorithm to focus on finding a small number conflicts that are useful in a wide range of states. The chosen conflict is $\{p, q_n\}$ in the case depicted in Figure 1 (a), and $\{p_n, q_m\}$ in that of Figure 1 (b).

Intuitively, this works better in our setting because the set of all conflicts generated by the same plan failure tends to be redundant, and thus needlessly grows the size of the task leading to slow evaluation times without much gain in informativeness.

These changes do not affect the fundamental property of Algorithm 1, that it converges to a real plan. To show convergence, the only property that FindConflicts must have is that it returns at least one new conjunction whenever σ fails to solve the original task. Our variant still gives that guarantee:

Lemma 3 *Assume we eliminate dominated preconditions* [5] *in* Π^C*. Let* $\sigma = a_1, \ldots, a_n$ *be a non-redundant plan for* Π^C *that is not valid for the original task* Π*, and let* c *be the conjunction extracted by the procedure described above. Then* $c \notin C$*.*

Proof: This is simply because, in both possible relaxed plan failure scenarios (Figure 1), the chosen conjunction $c = \{x, y\}$ ($\{x, y\} = \{p, q_n\}$ respectively $\{x, y\} = \{p_n, q_m\}$) is contained in the precondition of the failed action f. Assuming that $c = \{x, y\} \in C$, as we eliminate dominated preconditions, no action precondition in Π^C contains both x and y. Hence, in that case, c cannot be the chosen conjunction. ■

Theorem 9 (Convergence of conflict extraction) *Assume we eliminate dominated preconditions in* Π^C*, and Algorithm 1 is run without a size bound. Then eventually* σ *will be a plan for* Π*.*

Proof: Follows from Lemma 3 as the set of possible conjunctions is finite. ■

Contrasting Theorem 9 with Haslum's variant (Theorem 8), the latter gives a stronger convergence guarantee (only) in the sense that it guarantees a certain "minimum progress" is made in each iteration.

Lemma 3 (and thus Theorem 9) holds in the same way for Π^C_{ce}, i. e., when σ is a sequence of conditional effects in Π^C_{ce}, that, viewed as a set of independent actions, constitutes a non-redundant plan for Π^C_{ce}. We rely on eliminating dominated preconditions here as that makes the proof very simple, and we use the technique in practice anyway. We did not verify whether or not convergence holds also if dominated preconditions are not eliminated; we conjecture that it does.

Since there can be multiple conflicts in the BSG of a relaxed plan, in our experiments we choose (arbitrarily) one that minimizes the number of conditional effects (or STRIPS actions, in the case of Π^C) created. We place a bound on the factor x by which Π^C_{ce} exceeds the size of the original planning task Π. Precisely, when $x = 1$, no π-fluents or conditional effects are added, and $\Pi^C_{\mathsf{ce}} = \Pi^+$, resulting in a standard relaxed plan heuristic. For growth bounds $x > 1$, π-fluents are added until the number of conditional effects in the task reaches $(x - 1) \cdot |A|$. For Π^C, x limits the total number of actions in the task as a multiple of $|A|$.

5. Recall that eliminating dominated preconditions means that, whenever we add a fluent π_c to the precondition of an action, or to the condition of a conditional effect, we remove from that condition all fluents $p \in c$ and π-fluents $\{\pi_{c'} \mid c' \subseteq c\}$.

Example 2 *Consider again the STRIPS planning task from Example 1, with variables* $\{x_0, \ldots, x_n, y\}$*, initial state* $I = \{x_0, y\}$*, goal* $G = \{x_n\}$*, and unit-cost actions*

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

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

As previously discussed, setting $C = \{\pi_{x_1,y}, \ldots, \pi_{x_{n-1},y}\}$ *renders the delete relaxation perfect, i. e., results in the relaxed plan having to re-establish* y *in between every two b-actions. Exactly that set* C *is iteratively selected by our procedure.*

Assuming a best supporter function based on either of h^{add} *or* h^{max}*, in the first iteration of Algorithm 1 the BSG will be:*

$$b_0 \xrightarrow[x_1]{\qquad} b_1 \xrightarrow[x_2]{\qquad} b_2 \qquad \cdots \qquad b_{n-2} \xrightarrow[x_{n-1}]{} b_{n-1}$$

The relaxed plan fails to execute when trying to apply the second action, b_1*. The corresponding failure scenario matches Figure 1 (a):*

$$\begin{tikzcd} b_0 \arrow[r, "x_1"'] \arrow[r, bend left, squiggly, "\neg y"] & b_1 \end{tikzcd}$$

The chosen conflict thus is $\{y, x_0\}$*. With the now non-empty set of conjunctions* C *containing just that single conjunction, the precondition of* b_1 *contains* $\pi_{\{x_1,y\}}$ *which must be established using action* a*, so that the BSG now takes the form (note that the dominated preconditions* y *and* x_i *of* b_1 *are eliminated):*

$$b_0 \xrightarrow[x_1]{\qquad} a \xrightarrow[\pi_{\{x_1,y\}}]{\qquad} b_1 \xrightarrow[x_2]{\qquad} b_2 \qquad \cdots \qquad b_{n-2} \xrightarrow[x_{n-1}]{} b_{n-1}$$

The relaxed plan now fails to execute when trying to apply the fourth action, b_2*. The corresponding failure scenario is:*

$$\begin{tikzcd} b_1 \arrow[r, "x_2"'] \arrow[r, bend left, squiggly, "\neg y"] & b_2 \end{tikzcd}$$

The chosen conflict now is $\{y, x_2\}$*. Iterating the procedure will, in the same manner, select exactly the set* C *above one-by-one, at the end of which the relaxed plan will solve the original planning task.*

5.3 Experiments

We evaluate the impact of using the Π^C_{ce} compilation in a relaxed plan heuristic in the context of a greedy search. The expected impact of using a heuristic based on the improved relaxation is two-fold. On the one hand, it should make the heuristic more informative, enabling the search to find plans with fewer node evaluations. On the other hand, there is a computational overhead associated with the growth of the problem, slowing down heuristic evaluations. We examine both of these effects individually, as well as their combined influence on *coverage*, or the set of problems that the planner is able to solve within given time and memory bounds, which we take to be the main measure of performance.

In this study, we do not consider the objective of producing plans of high quality (as measured by plan length or cost). That is not because plan quality is unimportant. Rather, the rationale for this decision is methodological: Seeking a high quality plan is not the same problem as seeking to find a plan with minimum search effort – particularly when quality is measured by non-unit action costs – and the requirements on heuristics for the two problems are quite different. Here, we have chosen to focus on one, viz. search efficiency, as measured by coverage and node evaluations, rather than conflate the two. The choice of a plain greedy search algorithm is also motivated by this decision. As a consequence, we treat all actions as having a unit cost of 1. Previous experiments have shown that in the context of greedy search, distinguishing action costs in heuristic calculation tends to result in lower coverage (Richter & Westphal, 2010). However, to at least assess the impact of our heuristics on plan quality, we do report data regarding plan length.

We next describe the experiment setup and baseline. We then discuss heuristic informativeness, computational overhead, the impact of conditional effect merging, the impact on plan length of using Π^C_{ce} heuristics, a comparison with other state-of-the-art heuristics for the same problem, the difference between using the Π^C and Π^C_{ce} compilations, and finding plans with no search.

5.3.1 Experiment Setup and Baseline

The compilation and associated heuristics were implemented in the Fast Downward planner (Helmert, 2006), and used in a greedy best-first search, with lazy evaluation and a second open list (with boosting) for states resulting from preferred operators. The planners were tested on all of the STRIPS domains from the 1998–2011 editions of the International Planning Competition (IPC). For domains from the last two IPCs, only the most recent sets of instances were used. All experiments were run on Opteron 2384 processors with the settings used in the competition: a memory limit of 2Gb and a time limit of 30 minutes.

The baseline planner configuration uses the relaxed plan heuristic, with best supporters identified by h^{add}, on the unmodified planning task (i. e., with the growth bound $x = 1$). It is a known fact that greedy search, and in particular greedy search with lazy evaluation and a strong bias towards preferred operators, can be highly sensitive to small changes in the relaxed plan, even changes that do not alter the heuristic value but rather only the operators that are preferred. Unfortunately, this fact is very rarely taken into account when heuristics are compared in the context of greedy search. Since the introduction of π-fluents alters the structure of the relaxed plan, we believe it is particularly important to determine whether the resulting differences in planner performance are really due to the relaxed plan being more (or less) informative.

Therefore, as a first step towards accounting for the "brittleness" of experiments with greedy heuristic search, we introduce a simple variance measure and use it to decide when the results of our experiments should be considered significant. Variance in the performance of the baseline planner is measured by randomizing the choice of supporters with equal h^{add} values in the construction of the relaxed plan and measuring the *maximum deviation* from the results of the baseline planner over five repeated runs. The results are shown in columns labeled "MaD" (Tables 1 and 4). For each domain and for the problem set as a whole, the deviation is defined by the differences in coverage and in the median number of heuristic evaluations. Note that we are not interested in whether randomization "helps" or "hurts" the search, but rather in the *magnitude of the variation* that it causes. When comparing the results of the planner using heuristics based on Π^C_{ce} or Π^C under different growth bounds with the results of the baseline planner, we consider the difference between them to be *significant* if it is greater in magnitude than the maximum deviation observed with randomization of the baseline. This should not be interpreted as "significance" in the statistical sense (although, if we assumed that randomization affects all heuristics equally, we could estimate the probability of the hypothesis of no difference), but simply as setting a reasonable threshold for what counts as a substantial difference in search performance.

5.3.2 Heuristic Informativeness

The comparison of heuristic informativeness is summarized in the right half of Table 1, which shows the ratio of the median (per domain, over tasks solved by both planners) number of heuristic evaluations for the baseline planner to that of the planners using the Π^C_{ce}-based heuristics. In just under half the domains, the difference in informativeness of the Π^C_{ce}-based heuristics compared to the baseline does not exceed the threshold for significance set by the sensitivity study (shown in the "MaD" column). Among the domains where there is a significant difference, in the majority using the Π^C_{ce}-based heuristics reduces the number of node evaluations, indicating that the augmented heuristics are more informative. In most of these cases, the ratio grows as more π-fluents are added, i.e., as the growth bound x is increased. The most drastic example can be seen in the Floortile domain, where all Π^C_{ce}-based heuristics evaluate four or more orders of magnitude fewer nodes, compared to the standard delete relaxation heuristic. This allows them to easily solve all of the instances in the domain. By comparison, no planner in IPC 2011 was able to solve more than 9 of the 20 instances in this domain. In the Woodworking domain, Π^C_{ce} heuristics are more than two orders of magnitude more informative, but there is no associated increase in coverage as all tasks are solved by all configurations.

In roughly a third of the domains there is a consistent (or nearly consistent) loss of informativeness, though in most of them it is not significant. Note that this loss of informativeness does not always correlate with a loss in coverage. This can be attributed to different factors, including the small magnitudes of loss, as well as the fact that the ratio of node evaluations is taken only over tasks solved by both the planners compared. Another issue is that dramatic coverage losses are often due to the computational overhead incurred by the Π^C_{ce} compilation. In particular, in the Openstacks and Satellite domains, the decrease in the number of tasks solved with the Π^C_{ce}-based heuristics matches almost exactly the number of tasks for which the conflict selection and compilation process fails to complete within the 1800 seconds allocated per task. We get back to this in the next subsection.

It is worth noting that the quality of Π^C_{ce}-based heuristics can be highly sensitive to the precise choice of π-fluents used in the compilation.[6] Hence, there may exist better policies for making this choice than the relatively simple one we have used here.

The "HO" and "PO" colums in Table 1 (coverage only) examine the effect of the new heuristic function, respectively the new preferred operators returned by that function, in separation. PO corresponds to a configuration that uses the relaxed plan from the Π^C_{ce} task (built with $x = 1.5$ and a timeout of $t = 60s$, discussed in Section 5.3.3) only to identify preferred operators, together with the heuristic value from the baseline ($x = 1$) heuristic. HO, on the other hand, uses the heuristic values obtained with $x = 1.5$ and the preferred operators from $x = 1$. Interestingly, either heuristic values or preferred operators alone are sufficient to greatly improve coverage in Floortile, the domain in which our techniques have the greatest impact. Both the HO and PO configurations are able to solve every instance of this domain.[7] The effect in other domains is mixed, with both configurations solving sometimes more, sometimes fewer instances.

5.3.3 Computational Overhead

The computational overhead of the Π^C_{ce}-based heuristics, compared to the standard relaxed plan heuristic, stems from two sources: (1) the time spent on computing the set of π-fluents to add to the problem, and (2) the greater overhead of heuristic evaluation in the Π^C_{ce} task. Table 2 shows three measures of their impact.

The first four columns (under "Timeouts") show the number of instances in which the construction of the Π^C_{ce} task does not finish within 1800 seconds, while the second set of four columns (under "> 60 sec") shows the number of instances for which the construction time exceeds 60 seconds (inclusive of those instances in the first set of columns). Note that this behavior – spending a large amount of time on the Π^C_{ce} construction without reaching the growth bound – is partly due to our strategy for selecting π-fluents, since we purposely choose those π-fluents which will increase the size of the compiled task the least. While there are several domains in which construction time frequently exceeds 60 seconds, this does not happen in those domains where the Π^C_{ce}-based heuristic are most informative, such as Floortile and Woodworking. This suggests that imposing a time limit on the construction of the Π^C_{ce} task will incur only a small loss of informativeness. We present coverage results for such a strategy (using a 60 second time limit) in Table 4 below. It is significantly better than the baseline planner, and compares favourably with other state of the art heuristics.

As expected, in most domains evaluating heuristics on the Π^C_{ce} task is slower than on the standard delete relaxation, and tends to slow down more as the growth bound x increases, due to the larger number of fluents and actions in the compiled planning task. The median slowdown per domain is typically of the same order as x itself, and exceeds one order of

6. Indeed, the results reported in our earlier paper (Keyder, Hoffmann, & Haslum, 2012) show an increase in informativeness in the Barman and Parcprinter domains.
7. A plausible explanation for this is the behavior with respect to dead-end states (intuitively, where the robot has "painted itself into a corner") that are unrecognized under the standard delete relaxation heuristic, i. e., where a relaxed plan for Π exists. It appears that Π^C_{ce} is highly effective at fixing this issue: While under $x = 1$ search encounters millions of states with $h^{\mathrm{FF}}(\Pi) = \infty$, HO encounters only few states with $h^{\mathrm{FF}}(\Pi^C_{\mathsf{ce}}) = \infty$ (suggesting that $h^{\mathrm{FF}}(\Pi^C_{\mathsf{ce}})$ prunes dead-ends early on), and PO encounters no such states at all (suggesting that $h^{\mathrm{FF}}(\Pi^C_{\mathsf{ce}})$ preferred operators prevent the search from entering the dead-end regions in the first place).

Domain	**Coverage**								**Median Node Evaluations Ratio**				
	x=1	MaD	x =				PO	HO	MaD	x =			
			1.5	**2**	**2.5**	**3**	**1.5**	**1.5**		**1.5**	**2**	**2.5**	**3**
Airport (50)	36	±2	+1	+**3**	+1	+2	+1	±0	4.32	1.36:1	1.34:1	1.40:1	1.50:1
Barman (20)	13	±5	+**6**	+3	+1	−3	−3	+5	9.91	1:1.63	1:2.22	1:8.33	1:25
Blocksworld (35)	35	±0	±0	±0	±0	±0	±0	±0	5	2.84:1	2.90:1	2.90:1	3.02:1
Depots (22)	19	±1	+**2**	+**2**	+**2**	+**2**	±0	−1	12.13	2.84:1	3.26:1	3.65:1	8.55:1
Driverlog (20)	20	±0	±0	±0	±0	±0	±0	±0	1.16	**2.17**:1	**2.64**:1	**2.65**:1	**2.31**:1
Elevators (20)	19	±3	+1	+1	+1	+1	+1	+1	1.31	1.25:1	**1.47**:1	**1.34**:1	1.22:1
Floortile (20)	6	±0	+**14**	+**14**	+**14**	+**14**	+**14**	+**14**	6.42	**15013**:1	**15110**:1	**14757**:1	**19674**:1
FreeCell (80)	79	±2	+1	−**3**	−**4**	−2	±0	+1	1.28	1:1.11	1:1.28	1:1.17	1:**1.29**
Grid (5)	5	±0	±0	±0	±0	±0	±0	±0	1.21	**1.27**:1	**2.83**:1	1.27:1	1.27:1
Gripper (20)	20	±0	±0	±0	±0	±0	±0	±0	1	1:**1.05**	1:**1.51**	1:**1.49**	**1.23**:1
Logistics00 (28)	28	±0	±0	±0	±0	±0	±0	±0	1.64	1.47:1	1.47:1	1.53:1	1.60:1
Logistics98 (35)	34	±1	+1	±0	−1	±0	+1	+1	1.45	**2.53**:1	**2.57**:1	**2.83**:1	**2.70**:1
Miconic (150)	150	±0	±0	±0	±0	±0	±0	±0	1	**1.16**:1	**1.22**:1	**1.29**:1	**1.36**:1
Mprime (35)	35	±0	±0	±0	±0	±0	±0	±0	1.09	**2.33**:1	**2.33**:1	**2.33**:1	**2.33**:1
Mystery (30)	16	±0	+**3**	+**3**	+**3**	+**3**	+**3**	+**1**	1.07	**1.21**:1	**1.27**:1	**1.29**:1	**1.29**:1
Nomystery (20)	9	±2	−2	−**3**	−**3**	−**3**	−1	−2	13.93	1:1.44	2.58:1	9.95:1	10.13:1
Openstacks (20)	20	±0	−**9**	−**9**	−**9**	−**9**	−**1**	−**1**	1.08	**2.52**:1	**1.31**:1	1:**1.23**	1:**1.11**
Parcprinter (20)	16	±7	−**11**	−**8**	−7	−7	−4	−**10**	1.58	1:1.04	1:1.04	1:1.08	1:1.09
Parking (20)	20	±1	−**2**	−**8**	−**7**	−**5**	±0	−**6**	7.06	1:1.12	3.02:1	1.61:1	1.18:1
Pathways (30)	30	±2	−1	−2	−**5**	−1	−1	±0	1.21	1:1.03	1:1	1:1.05	1:1.11
Pegsol (20)	20	±0	±0	±0	±0	±0	±0	±0	3.42	1.44:1	1.09:1	1.66:1	1.41:1
Pipes-NoTk (50)	42	±1	±0	−1	±0	±0	−1	±0	16.47	1:1.07	1.08:1	1.14:1	1.29:1
Pipes-Tank (50)	38	±4	+3	+2	±0	+2	−1	+3	2.46	1:1	1:1	1:1	1:1.05
PSR (50)	50	±0	±0	±0	±0	±0	±0	±0	1	1:1	**1.02**:1	**1.12**:1	**1.12**:1
Rovers (40)	40	±0	−**1**	±0	±0	±0	−**1**	±0	1.33	1.12:1	1.15:1	1.12:1	1.20:1
Satellite (36)	35	±1	−**2**	−**5**	−**6**	−**8**	+1	±0	1.36	1.15:1	1.36:1	1.30:1	1.29:1
Scanalyzer (20)	18	±1	+**2**	+**2**	+**2**	+**2**	−1	+**2**	1.85	1.17:1	1.80:1	**3.43**:1	**3.25**:1
Sokoban (20)	19	±1	−**2**	−**2**	−**3**	−**3**	±0	−**2**	1.46	1.17:1	1:1.09	1.01:1	1.06:1
Tidybot (20)	16	±3	−2	±0	±0	−1	−1	±0	1.22	1.15:1	1:**1.25**	1:**1.38**	1:1.08
TPP (30)	30	±0	±0	±0	±0	±0	±0	±0	2.29	1:1.20	1:1.13	1:1.19	1:1.01
Transport (20)	11	±1	+**2**	−**2**	+1	+1	−**7**	−**3**	2.73	1.20:1	1:1.11	1:1.29	1:1.01
Trucks (30)	14	±0	+**1**	+**3**	+**4**	+**2**	±0	+**2**	1.66	1.16:1	**1.97**:1	**7.27**:1	**4.57**:1
Visitall (20)	19	±1	+1	+1	−**2**	−**2**	−**2**	−**2**	1.34	1.01:1	1:1.08	1:1.09	1:**1.49**
Woodwork (20)	20	±0	±0	±0	±0	±0	±0	±0	52.78	**245.72**:1	**263.3**:1	**245.72**:1	**245.72**:1
Zenotravel (20)	20	±1	±0	±0	±0	±0	±0	±0	1.28	1.22:1	1.24:1	**1.36**:1	**1.42**:1
Total (1126)	1002	±19	+6	−9	−17	−14	−3	+3					

Table 1: Planner coverage and heuristic informativeness using Π^C_{ce} with varying growth bounds, *without* conditional effect merging. Coverage shows the number of problems solved for the baseline configuration ($x = 1$), and the difference (increase/decrease) relative to the baseline for the other configurations; "PO" uses only the preferred operators obtained from the Π^C_{ce} compilation with $x = 1.5$ and $t = 60s$, returning the $x = 1$ heuristic value, while "HO" uses the heuristic values obtained with $x = 1.5$ and $t = 60s$, and the preferred operators from $x = 1$. Heuristic informativeness is measured by the ratio of the per-domain median number of node evaluations, comparing the baseline to our configurations (across those instances solved by both configurations), normalized so that the smaller value is 1. That is, an entry $m : 1$ means that the baseline planner requires m times as many heuristic evaluations as the other planner. Columns labeled "MaD" show the magnitude of the maximum deviation (in coverage and ratio) from the baseline in our sensitivity study: values in bold are those that exceed this threshold, and which we therefore consider to be significant.

magnitude only in the Floortile domain when $x = 1.5$. Somewhat surprisingly, there are domains in which heuristic evaluations become faster as more π-fluents are added. A possible explanation for this is that because we eliminate dominated preconditions (cf. Section 3), the number of action preconditions decreases and the delete-relaxation hypergraph of the Π^C_{ce} becomes more graph-like as a result.

	Timeouts x =				> 60 **sec** x =				**Ratio of Median Evaluations/sec** x =			
Domain	**1.5**	**2**	**2.5**	**3**	**1.5**	**2**	**2.5**	**3**	**1.5**	**2**	**2.5**	**3**
Airport (50)	1	1	1	1	15	20	23	26	1.08:1	1.19:1	1.30:1	1.26:1
Barman (20)									1.35:1	1.88:1	2.42:1	3.09:1
Blocksworld (35)									1.50:1	1.60:1	1.56:1	1.42:1
Depots (22)								1	1.65:1	1.81:1	2.03:1	2.31:1
Driverlog (20)									1.56:1	2.38:1	3.17:1	4.07:1
Elevators (20)					10	10	10	12	1.87:1	2.92:1	3.99:1	5.45:1
Floortile (20)									17.67:1	8.28:1	5.97:1	5.49:1
FreeCell (80)									1.20:1	1.31:1	1.39:1	1.66:1
Grid (5)									1.27:1	2.55:1	3.04:1	3.04:1
Gripper (20)									1:2.13	1:2.04	1:2.13	1:1.69
Logistics00 (28)									1.09:1	1:1.76	1:1.35	1:1.22
Logistics98 (35)			1	1	3	6	9	10	1.20:1	1.64:1	2.55:1	3.72:1
Miconic (150)									1.06:1	1.10:1	1.18:1	1.27:1
Mprime (35)									1.60:1	1.80:1	1.60:1	1.80:1
Mystery (30)									1.28:1	1.35:1	1.42:1	1.42:1
Nomystery (20)									1.43:1	2.20:1	2.53:1	3.26:1
Openstacks (20)	9	9	9	9	16	16	16	16	1.16:1	1.73:1	2.49:1	3.17:1
Parcprinter (20)									1:8.33	1:8.33	1:6.25	1:1.81
Parking (20)					14	16	16	16	2.32:1	3.61:1	4.84:1	6.20:1
Pathways (30)						3	6	7	1.36:1	2.04:1	1.01:1	1.64:1
Pegsol (20)									1.25:1	1.08:1	1.48:1	1.11:1
Pipes-NoTank (50)									1.41:1	1.94:1	2.31:1	2.78:1
Pipes-Tank (50)									1.61:1	2.37:1	2.42:1	3.39:1
PSR (50)									1:1	1:1	1.02:1	1:1.05
Rovers (40)					5	5	5	5	1.11:1	1.43:1	1.48:1	1.83:1
Satellite (36)	3	6	7	9	9	11	13	13	1.14:1	1.07:1	1:1.01	1.30:1
Scanalyzer (20)					3	3	3	4	1.09:1	2.22:1	2.42:1	2.73:1
Sokoban (20)									1.23:1	1.38:1	1.62:1	1.82:1
Tidybot (20)						1	1	1	1.48:1	2.03:1	3.36:1	2.22:1
TPP (30)					4	5	5	6	1.59:1	1.70:1	2.38:1	2.84:1
Transport (20)			2	3	11	12	16	17	2.38:1	4.04:1	6.50:1	8.80:1
Trucks (30)						2	2	4	1.62:1	2.29:1	2.52:1	3.30:1
Visitall (20)					6	8	9	10	1:1.49	1:1.33	1:1.15	1:1.07
Woodwork (20)									1:2.63	1:2.08	3.29:1	6.32:1
Zenotravel (20)									1.09:1	1:1	1.06:1	1.10:1
Total (1126)	13	16	20	23	96	118	134	148				

Table 2: Computational overhead of Π^C_{ce}. The first set of columns ("Timeouts") shows the number of tasks for which the Π^C_{ce} construction does not finish within the 1800 second time limit, and the second set ("> 60 sec") shows the number of tasks for which construction time exceeds 60 seconds (inclusive of those in the first set of columns). To improve readability, only non-zero entries are shown (i.e., the blank cells in these columns are zeroes). The last set of columns shows the median (per domain, over commonly solved tasks) ratio of heuristic evaluations per second for the baseline planner ($x = 1$) to that of the other planner ($x > 1$). An entry $m : 1$ means that the baseline planner performs m times as many heuristic evaluations per second as the other planner.

5.3.4 Conditional Effect Merging

In a majority of domains, conditional effect merging slightly increases or does not change the informativeness of the Π^C_{ce}-based heuristics. Some exceptions to this are the Logistics00 and Gripper domains, where merging results in a heuristic that is twice as informative (using the same ratio of median number of evaluations metric as presented in Table 1), the Nomystery domain where it is an order of magnitude more informative (for the problems that are solved by both heuristics), and the Barman domain where it is four times *less* informative. In general, higher informativeness occurs in domains in which all tasks are solved by all planners, so it does not result in increased coverage. Indeed, as shown in Table 4 below, conditional effect merging proves to be detrimental to the overall coverage of the planner using the Π^C_{ce}-based heuristic: the best Π^C_{ce} configuration with conditional effect merging solves, in total, only 4 more tasks than the standard relaxed plan heuristic, while the same configuration without conditional effect merging solves 20 more tasks.

The runtime overhead of the merging 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. For $x = 1.5$, comparing the Π^C_{ce}-based heuristic with conditional effect merging to that without, the ratio of the median number of heuristic evaluations per second (the same metric as used on the right-hand side of Table 2) shows a maximal per-domain slow-down of 2.04, and an across-domain average of 1.11. Coverage decreases in domains other than Barman therefore appear to be due to the sensitivity of search to small changes in the heuristic function (rather than due to the time taken to compute that function).

5.3.5 Plan Length with Π^C_{ce}

To determine the effect of using Π^C_{ce} heuristics on plan quality, we compare the length of the plans found with Π^C_{ce} heuristics to those found with $x = 1$, the standard delete relaxation heuristic. The plan length measure is equivalent to plan quality in the unit-cost setting. We consider the median ratio of plan length found with the standard delete relaxation heuristic to that found with the Π^C_{ce} heuristic, over the set of instances that are solved by both configurations (Table 3). In general, we do not observe large differences, with the median ratio staying close to 1. One notable exception is the blocksworld domain, in which heuristics based on the Π^C_{ce} compilation consistently find significantly shorter plans. This results from the Π^C_{ce} heuristics' ability to deduce implicit ordering constraints in the domain, avoiding actions that lead to temporary improvements during greedy search but that later need to be reversed, adding to plan length. Π^C_{ce} also leads to shorter plans in the Gripper, Mprime, and Woodworking domains, but tends to result in longer plans in the Barman and Grid domains.

5.3.6 Comparison to the State of the Art

Table 4 shows coverage for a variety of heuristics and planners. The best configurations for each of the two compilations with $x > 1$ achieve better overall coverage results than the standard relaxed plan heuristic. The best-performing heuristics obtained with the Π^C_{ce} compilation *without* conditional effect merging, and the Π^C compilation, give coverages of 1022 and 1023 respectively. The difference between these and the coverage of the baseline planner is greater than the significance threshold. These numbers also far exceed the coverage obtained with the h^{cea} heuristic, but fall short of the 1039 instances solved with the dual heuristic approach used by LAMA. However, combining LAMA and the best Π^C_{ce}-nm configuration in a portfolio planner that runs LAMA for 1500 seconds and search with the Π^C_{ce}-nm heuristic for 300 seconds results in a coverage of 1063 out of 1115 solvable problems. Almost all of this difference results from the Π^C_{ce}-based heuristic's superior performance in the Floortile, and to a lesser extent, Airport domains.

Domain	x = 1.5	x = 2	x = 2.5	x = 3
Airport	1 : 1.00	1 : 1.00	1 : 1.00	1 : 1.00
Barman	1 : 1.11	1 : 1.12	1 : 1.23	1 : 1.32
Blocksworld	1.65 : 1	1.68 : 1	1.69 : 1	1.64 : 1
Depots	1.08 : 1	1.03 : 1	1.04 : 1	1.06 : 1
Driverlog	1.04 : 1	1 : 1.00	1 : 1.00	1 : 1.00
Elevators	1 : 1.06	1 : 1.05	1 : 1.05	1 : 1.12
Floortile	1 : 1.10	1 : 1.15	1 : 1.05	1 : 1.03
FreeCell	1 : 1.02	1 : 1.03	1 : 1.04	1 : 1.06
Grid	1 : 1.24	1 : 1.17	1 : 1.17	1 : 1.17
Gripper	1.04 : 1	1.16 : 1	1.31 : 1	1.31 : 1
Logistics00	1 : 1.14	1 : 1.13	1 : 1.13	1 : 1.12
Logistics98	1.07 : 1	1.05 : 1	1.05 : 1	1.06 : 1
Miconic	1 : 1.00	1 : 1.00	1 : 1.00	1.01 : 1
Mprime	1.12 : 1	1.17 : 1	1.12 : 1	1.17 : 1
Mystery	1 : 1.00	1 : 1.00	1 : 1.00	1 : 1.00
Nomystery	1 : 1.00	1 : 1.02	1 : 1.02	1 : 1.02
Openstacks	1 : 1.02	1 : 1.02	1 : 1.02	1 : 1.01
Parcprinter	1 : 1.00	1 : 1.00	1 : 1.00	1.04 : 1
Parking	1.26 : 1	1 : 1.01	1.04 : 1	1 : 1.00
Pathways	1 : 1.01	1 : 1.00	1 : 1.01	1 : 1.02
Pegsol	1 : 1.04	1 : 1.00	1 : 1.03	1 : 1.02
Pipesworld	1 : 1.09	1 : 1.00	1 : 1.17	1 : 1.14
Pipesworld	1 : 1.12	1 : 1.00	1 : 1.04	1 : 1.06
PSR	1 : 1.00	1 : 1.00	1 : 1.00	1 : 1.00
Rovers	1 : 1.01	1 : 1.00	1.01 : 1	1 : 1.00
Satellite	1 : 1.00	1 : 1.00	1 : 1.00	1 : 1.01
Scanalyzer	1 : 1.00	1 : 1.00	1 : 1.00	1 : 1.00
Sokoban	1 : 1.00	1 : 1.03	1.01 : 1	1 : 1.03
Tidybot	1 : 1.06	1 : 1.19	1 : 1.16	1 : 1.13
TPP	1 : 1.04	1 : 1.02	1 : 1.00	1 : 1.00
Transport	1 : 1.10	1 : 1.16	1 : 1.06	1 : 1.13
Trucks	1 : 1.00	1 : 1.00	1 : 1.00	1 : 1.00
Visitall	1.01 : 1	1 : 1.00	1 : 1.04	1 : 1.08
Woodwork	1.21 : 1	1.21 : 1	1.21 : 1	1.21 : 1
Zenotravel	1 : 1.00	1 : 1.00	1 : 1.00	1 : 1.00

Table 3: Median ratio of length of plans found with $x = 1$, to the length of plans found with Π^C_{ce} with different values of x, over instances solved by both planners. An entry $m : 1$ means that the baseline's plans are m times longer than those of the other planner. No conditional effect merging was used.

5.3.7 Π^C vs. Π^C_{ce}

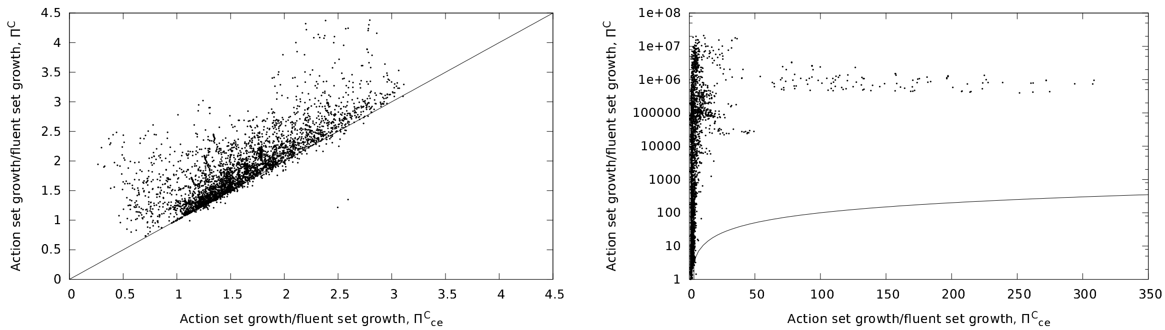
Given a fixed number of π-fluents, the difference in size between the Π^C and the Π^C_{ce} compilations is exponential in the worst case. Mutex pruning, however, can mitigate much of the growth of Π^C. Consider, for example, an action in the Π^C_{ce} compilation that has n different conditional effects. If mutexes are not considered, one would expect the number of action representatives generated for the same set of π-fluents in Π^C to be 2^n. If, however, each of the n π-fluents generating the conditional effects can be shown to be mutex with one another, the number of action representatives generated in Π^C is also only n.

We have found in our experiments that this effect leads to much slower growth in Π^C than what might be expected. Consider Figure 2. Each point on the graph represents a

	Coverage								
Domain	**x=1**	MaD	Π^C_{ce}	Π^C_{ce}-nm	Π^C	h^{FF}	h^{cea}	LAMA	PF
Airport (50)	36	±2	37	36	38	34	42	31	36
Barman (20)	13	±5	14	**19**	18	20	0	20	20
Blocksworld (35)	35	±0	35	35	35	35	35	35	35
Depots (22)	19	±1	**22**	**21**	**21**	18	18	21	22
Driverlog (20)	20	±0	20	20	20	20	20	20	20
Elevators (20)	19	±3	19	20	19	19	20	20	20
Floortile (20)	6	±0	**20**	**20**	**20**	6	6	5	20
FreeCell (80)	79	±2	80	80	79	79	79	79	79
Grid (5)	5	±0	**4**	5	5	5	5	5	5
Gripper (20)	20	±0	20	20	20	20	20	20	20
Logistics00 (28)	28	±0	28	28	28	28	28	28	28
Logistics98 (35)	34	±1	35	35	35	33	35	35	35
Miconic (150)	150	±0	150	150	150	150	150	150	150
Mprime (35)	35	±0	35	35	35	35	35	35	35
Mystery (30)	16	±0	**18**	**19**	**19**	16	19	19	19
Nomystery (20)	9	±2	9	7	11	10	7	13	13
Openstacks (20)	20	±0	20	20	20	20	19	20	20
Parcprinter (20)	16	±7	9	**5**	10	20	12	20	20
Parking (20)	20	±1	**12**	**18**	**15**	20	20	20	20
Pathways (30)	30	±2	**24**	29	29	30	28	30	30
Pegsol (20)	20	±0	20	20	20	20	20	20	20
Pipes-NoTank (50)	42	±1	42	42	42	43	40	44	44
Pipes-Tank (50)	38	±4	41	41	40	39	32	43	44
PSR (50)	50	±0	50	50	50	50	50	50	50
Rovers (40)	40	±0	40	40	40	40	40	40	40
Satellite (36)	35	±1	36	36	36	36	36	36	36
Scanalyzer (20)	18	±1	19	**20**	**20**	18	20	20	20
Sokoban (20)	19	±1	18	**17**	**17**	19	3	19	19
Tidybot (20)	16	±3	14	14	16	14	16	17	17
TPP (30)	30	±0	30	30	30	30	29	30	30
Transport (20)	11	±1	11	**15**	11	11	17	19	19
Trucks (30)	14	±0	**16**	**15**	**16**	19	15	15	16
Visitall (20)	19	±1	18	20	18	3	3	20	20
Woodwork (20)	20	±0	20	20	20	20	8	20	20
Zenotravel (20)	20	±1	20	20	20	20	20	20	20
Total (1126)	1002	±19	1006	**1022**	**1023**	1000	947	1039	1062

Table 4: Comparison of state-of-the-art-heuristics for satisficing planning. Columns "$x = 1$" and "MaD" are as in Table 1. Column Π^C_{ce} shows coverage for the best configuration (in terms of overall coverage) with that compilation when using conditional effect merging (namely $x = 1.5, t = 60$); Π^C_{ce}-nm is the best Π^C_{ce} configuration without conditional effect merging (which happens to use the same x and t); Π^C is the best Π^C configuration where $x > 1$ (which again happens to use the same x and t). Entries in bold in these columns are those where the difference from the baseline planner exceeds our threshold for significance (given by the "MaD" column). Column PF shows coverage for a portfolio planner that runs LAMA for 1500 seconds and Π^C_{ce} for 300 seconds.

single problem instance (from the same instance set as before), paired with a value of x. For each of Π^C_{ce} (x-axis) and Π^C (y-axis), we measure the ratio of growth in $|A|$ over growth in $|F|$, i. e., the factor by which the compilation increased the size of the action set encoding (measured by the number of actions in Π^C and by the number of conditional effects in Π^C_{ce}), divided by the factor by which the compilation increased the number of fluents. In other



(a) (b)

Figure 2: Growth in problem size as ratio of growth in $|A|$ to growth in $|F|$, with (a) and without (b) mutex pruning. Each point corresponds to a single instance and value of x. $f(x) = x$ is also shown for reference.

words, we assess the growth of the encoding over the number of conjunctions $|C|$, which in theory is worst-case exponential for Π^C but linear for Π^C_{ce}.

When mutex pruning is not used (Figure 2 (b)), the growth of A in Π^C is rapid and the ratio quickly increases to millions; with mutex pruning (Figure 2 (a)), the growth in Π^C is still faster than in Π^C_{ce}, but the difference is much smaller.

5.3.8 Finding Plans With No Search

When no growth or time limit is imposed on the construction of the Π^C or Π^C_{ce} tasks, Algorithm 1 can be used as a complete planning algorithm. While it is not competitive with heuristic search methods, it is nevertheless interesting to observe some performance details of this algorithm in various domains. The coverage obtained with this algorithm using the Π^C and Π^C_{ce} compilations, as well as some statistics about the growth of the compiled tasks, are shown in Table 5. The difference between Π^C_{ce} and Π^C is much more visible here, since the number of π-fluents added is, in general, much larger than in the growth-bounded constructions we have used for heuristic computation. Π^C_{ce} is able to rapidly add a much larger number of π-fluents, and can therefore find relaxed plans that are solutions to the original planning task as well. Overall, Π^C_{ce} solves 568 tasks compared to 404 for Π^C, and solves an equal or greater number of tasks in all except 4 domains.

Considering individual domains, it can be seen that Π^C and Π^C_{ce} are able to solve all or almost all of the tasks in certain domains such as Logistics00, Mprime, Mystery, Parcprinter, PSR, and Woodworking. In some of these domains, even the addition of a small amount of information is sufficient to obtain relaxed plans that are plans for the original task, and the maximum x values required to solve all tasks are quite low. This is the case in the Mprime, Mystery, and Woodworking domains, where the maximum required x values are 4.07, 4.62, and 1.35, respectively. In others such as Elevators, Openstacks, Transport, and Visitall, where even the smallest tasks are quite large and many different plans are possible, it is not possible to introduce enough π-fluents to disqualify all of the possible relaxed plans that do not constitute real plans, and no tasks can be solved.

	Π^C_{ce}				Π^C			
Domain	**Cov.**	**Min**	**Max**	**Med**	**Cov.**	**Min**	**Max**	**Med**
Airport (50)	34	1.00	44.50	2.31	31	1.00	168.45	3.56
Barman-sat (20)	0	-	-	-	0	-	-	-
Blocksworld (35)	33	1.21	141.79	11.44	30	1.21	400.94	15.49
Depots (22)	16	1.82	90.93	8.48	14	1.82	221.41	12.45
Driverlog (20)	15	1.00	37.11	8.92	14	1.00	74.97	9.96
Elevators-sat11 (20)	0	-	-	-	0	-	-	-
Floortile-sat11 (20)	4	31.47	216.18	112.50	9	48.92	633.11	107.21
FreeCell (80)	1	91.41	91.41	91.41	2	212.24	272.12	242.18
Grid (5)	4	1.29	22.53	1.41	4	1.31	41.22	1.54
Gripper (20)	16	5.37	208.75	67.69	6	6.77	501.28	76.12
Logistics00 (28)	27	1.27	91.10	6.79	27	1.27	585.53	14.11
Logistics98 (35)	24	1.22	41.16	31.14	21	1.22	149.91	5.80
Miconic (150)	106	1.20	175.11	55.01	36	1.20	2706.86	54.47
Mprime (35)	35	1.02	4.07	1.15	35	1.02	13.22	1.14
Mystery (30)	19	1.02	4.62	1.25	19	1.02	8.34	1.20
Nomystery-sat11 (20)	5	16.72	59.80	41.14	3	21.91	43.50	33.27
Openstacks-sat11 (20)	0	-	-	-	0	-	-	-
Parcprinter-sat11 (20)	20	3.66	76.04	9.95	9	1.60	276.81	88.23
Parking-sat11 (20)	0	-	-	-	0	-	-	-
Pathways-noneg (30)	5	1.77	51.72	7.26	3	2.32	203.41	89.79
Pegsol-sat11 (20)	19	69.36	707.06	137.98	0	-	-	-
Pipes-NoTank (50)	10	3.51	187.41	11.60	5	12.61	738.88	242.54
Pipes-Tank (50)	9	2.69	94.33	13.46	8	2.56	110.51	11.71
PSR (50)	50	1.03	77.38	7.29	49	1.05	1052.26	16.04
Rovers (40)	21	1.35	459.66	7.40	14	1.70	645.00	26.04
Satellite (36)	15	4.22	43.61	19.16	7	5.22	399.10	22.09
Scanalyzer-sat11 (20)	9	1.90	47.75	13.93	10	2.38	161.42	44.21
Sokoban-sat11 (20)	0	-	-	-	1	2.89	2.89	2.89
Tidybot-sat11 (20)	1	22.42	22.42	22.42	0	-	-	-
TPP (30)	20	1.17	43.12	13.75	7	1.17	560.95	11.27
Transport-sat11 (20)	0	-	-	-	0	-	-	-
Trucks (30)	15	5.54	21.36	13.56	8	6.32	100.15	41.05
Visitall-sat11 (20)	0	-	-	-	0	-	-	-
Woodwork-sat11 (20)	20	1.00	1.35	1.24	20	1.00	1.40	1.25
Zenotravel (20)	15	1.00	92.26	7.25	12	1.00	191.54	11.14
Total (1126)	568				404			

Table 5: Solving planning tasks with no search. Table shows **Cov**erage for the Π^C and Π^C_{ce} compilations, and the **Min**imum, **Max**imum, and **Med**ian values of x for solved tasks.

6. Heuristics for Optimal Planning

We now consider *admissible* heuristics, for optimal planning. Section 6.1 considers the LM-cut heuristic, showing that there are certain complications that make it difficult to obtain improved heuristic estimates from both Π^C and Π^C_{ce}. In Section 6.2, we consider an alternative method to lower-bound h^+, namely admissible cost-partitioning heuristics based on the conjunctive landmarks that can be obtained from Π^C_{ce}. We detail how to choose C in that setting, and present our experimental results in Section 6.3.

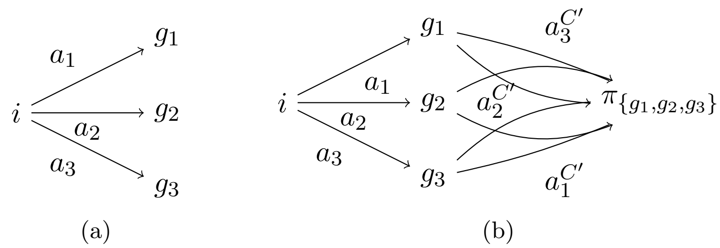


Figure 3: LM-cut in Π, and in the Π^C compilation, for Example 3.

6.1 LM-cut

The state-of-the-art admissible approximation of h^+ is computed by the LM-cut algorithm (Helmert & Domshlak, 2009). A logical approach to obtaining admissible heuristics from Π^C and Π^C_{ce} is therefore to apply LM-cut to these compilations. Unfortunately, it turns out that there are several serious obstacles to this. Before discussing these issues, we first give a brief description of the LM-cut algorithm, and then present the simpler case of the Π^C compilation, where the additional complication of conditional effects is not present.

Before LM-cut can be computed on a planning task with no deletes, a simple transformation is first applied that replaces the goal set G with a single goal that can only be achieved with a goal-achievement action whose precondition set is G, and adds a *dummy precondition* to all actions whose precondition set is empty. LM-cut then initializes $h^{\text{LM-cut}} := 0$ and, repeats the following steps until $h^{\max}(G)$ becomes 0: (1) Compute $h^{\max}$; (2) apply a *precondition choice function (PCF)* to each action precondition $\mathsf{pre}(a)$ that removes from $\mathsf{pre}(a)$ all but one of the fluents $p \in \mathsf{pre}(a)$ for which $h^{\max}(p)$ is maximal; (3) construct the *justification graph* whose vertices are the fluents and whose arcs are the precondition/effect pairs according to the PCF; (4) find a cut L between the initial state and the goal in the justification graph, given by the set of actions that enters the "goal zone", i. e., the set of fluents from which the goal can be reached at 0 cost; and (5) add $\mathsf{cost}_{\min} := \min_{a \in L} \mathsf{cost}(a)$ to the heuristic value $h^{\text{LM-cut}}$, and reduce the cost of each $a \in L$ by $\mathsf{cost}_{\min}$. As proved by Helmert and Domshlak (2009), this algorithm has two fundamental properties, namely (i) admissibility, $h^{\text{LM-cut}} \leq h^+$, and (ii) domination of $h^{\max}$, $h^{\max} \leq h^{\text{LM-cut}}$.

While it would be expected that the heuristic obtained in this manner from Π^C would be strictly more informative than that obtained from the original planning task Π, this turns out not to be the case. Indeed, the heuristic can become strictly *less* informative:

Example 3 *Let $\pi = \langle F, A, I, G \rangle$ be given by $F = \{g_1, g_2, g_3\}$, $A = \{a_1, a_2, a_3\}$, where $a_i = \langle \emptyset, \{g_i\}, \emptyset, \emptyset \rangle$, $I = \emptyset$, and $G = \{g_1, g_2, g_3\}$ (Figure 3). In words, π has three goals, each of which is achievable by a single action. Valid plans for π apply each action once in any order, to make all of the goals true. The cuts found by the LM-cut algorithm for this task are $\{a_1\}$, $\{a_2\}$, and $\{a_3\}$, regardless of the PCFs chosen, and the LM-cut algorithm therefore always computes the optimal cost 3. Consider now the Π^C compilation that results from the set $C = \{\{g_1, g_2, g_3\}\}$. F^C then contains the single π-fluent $\pi_{\{g_1,g_2,g_3\}}$, and a representative of each action $a_i^{C'}$ constructed with the sole non-empty subset $C' = \{\{g_1, g_2, g_3\}\}$ of C. The first cut found by LM-cut then contains these three representatives, as each adds the most expensive goal $\pi_{\{g_1,g_2,g_3\}}$. For* any *of the possible PCFs, the next cut is then the last, and the final heuristic estimate is only 2 as only two cuts have been found. If, for example, the* *precondition choice function chooses* g_1 *as the* $h^{\max}$ *justifier for* $a_2^{C'}$ *and* $a_3^{C'}$*, and* g_2 *as the* $h^{\max}$ *justifier for* $a_1^{C'}$*, the cut is* $\{a_1, a_2\}$*. After that cut, the goal can be reached at* 0 *cost via* $a_1, a_2, a_3^{\{\{g_1,g_2,g_3\}\}}$*, so* $h^{\max}$ *is* 0 *and LM-cut stops.*

Note that (similarly to $h^+(\Pi^C_{\mathsf{ce}})$, cf. Theorem 2) it is not possible for either $h^+(\Pi^C)$ or $h^{\max}(\Pi^C)$ to *decrease* with the addition of π-fluents, and that in this example the $h^{\max}$ cost of the task actually *increases* (from 1 to 2) with the addition of the π-fluent $\pi_{\{g_1,g_2,g_3\}}$. However, the type of interactions that are introduced are difficult for the LM-cut algorithm to reason about, resulting in worse admissible bounds in practice. LM-cut of course continues to dominate $h^{\max}$, proving that if a sufficient number of π-fluents are added, LM-cut will eventually tend towards the optimal cost of the task.

The weakness pointed out by Example 3 is inherited in the application of the LM-cut algorithm to the Π^C_{ce} compilation. Furthermore, that application involves an additional complication that proves formidable: LM-cut is not defined for conditional effects, and therefore cannot be directly applied to the Π^C_{ce} task. It turns out that of the two straightforward adaptations of the algorithm to problems with conditional effects, neither preserves both properties (i) admissibility and (ii) domination of $h^{\max}$.

To see why (ii) is at stake, 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 the cost estimate $h^{\text{LM-cut}} = 1$.

The issue 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 (i). Indeed, as we detail in Example 4 (Appendix A), there exist STRIPS tasks Π whose Π^C_{ce} compilations have the following property: there exists an action a such that reducing its cost 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 each of its effects separately leads to an estimate greater than $h^+(\Pi^C_{\mathsf{ce}}) = h^*(\Pi)$.

There is therefore no simple strategy for dealing with conditional effects that preserves both (i) and (ii) on all planning tasks. Since admissibility cannot be sacrificed, we must reduce costs globally and give up on dominating $h^{\max}$. As a particular implication of doing so, despite Theorem 5 which shows that $h^{\max}(\Pi^C_{\mathsf{ce}}) = h^1(\Pi^C_{\mathsf{ce}})$ converges to $h^*(\Pi)$, such convergence is *not* guaranteed for $h^{\text{LM-cut}}(\Pi^C_{\mathsf{ce}})$. 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.

As we detail in Section 6.3 below, on the IPC benchmarks, using A* with LM-cut computed on either Π^C or Π^C_{ce} often results in larger search spaces as more π-fluents are introduced. In all but a few cases, overall performance is worse with $h^{\text{LM-cut}}(\Pi^C)$ or $h^{\text{LM-cut}}(\Pi^C_{\mathsf{ce}})$ than with $h^{\text{LM-cut}}(\Pi)$. It remains an open question whether this can be improved.

6.2 Π^C_{ce} Landmarks

Landmarks in planning tasks are formulas ϕ over the set of fluents F that have the property that they are made true in some state during the execution of any valid plan. As the problem of checking whether even a single fluent is a landmark for a planning task is **PSPACE**-complete, approaches to finding landmarks have in the past focused on the delete relaxation, in which setting whether a fluent is a landmark or not can be checked in polynomial time. It has recently been shown that the maximum fixpoint solution to a set of simple recursive equations defines the complete set of single fact delete-relaxation landmarks, in other words those landmark formulas that consist of only a single literal $\phi = p$ (Keyder et al., 2010). This solution can be computed by an algorithm that repeatedly updates the set of such landmarks for each fluent and action in the planning task, until convergence. This method can naturally handle conditional effects by treating them as independent actions, as described in Section 5.1.

It has also been shown that these equations can be applied to any AND/OR graph structure, not necessarily corresponding to the delete relaxation of a planning task. This insight has been used to obtain landmarks from the Π^m task. Single π-fluent landmarks in Π^m correspond to conjunctive landmarks in Π that are not necessarily landmarks of the delete relaxation Π^+. This approach suffers, however, from the large number of π-fluents that are considered in Π^m, rendering landmark generation impractical for the Π^m compilations of larger tasks. Here we aim to take advantage of the flexibility of the Π^C_{ce} compilation to obtain non-delete-relaxation landmarks for the original task, while considering only a focused set of π-fluents and not all those of a given size m. As before, this allow us to consider larger conjunctions while keeping the size of the delete relaxation task low.

When using Π^C_{ce} for landmark finding, to focus the technique and keep its overhead at bay, we choose the set of conjunctions C so as to guarantee that every π-fluent is a landmark in Π^C_{ce} (and therefore in the original planning task). This is accomplished by extracting from the landmark graph sets of landmarks that are *simultaneously achieved*:

Definition 7 (Simultaneously achieved landmarks) *A set of landmarks $L_s = \{\phi_1, \ldots, \phi_n\}$ is simultaneously achieved if $L_c = \phi_1 \wedge ... \wedge \phi_n$ is a landmark in Π.*

Maximal sets of simultanously achieved landmarks can easily be extracted from a set of landmarks and orderings. Given an initial set of landmarks L and a set of orderings, the following are sets of sets of simultaneously achieved landmarks:

- $L_G = \{\{\phi \mid G \models \phi\}\}$
- $L_{nec} = \{\{\phi \mid \phi \prec_{\text{nec}} \psi\} \mid \psi \in L\}$
- $L_{gn} = \{\{\phi \mid \phi \prec_{\text{gn}} \psi\} \mid \psi \in L\}$

L_G contains a single set that is made up of the landmarks in L that are entailed by G. Since any valid plan must make all goals true in its final state, they are necessarily simultaneously achieved. Given a landmark ψ, L_{nec} contains a set which has as its elements all those landmarks ϕ that are ordered necessarily before ψ. Due to the definition of necessary orderings, all of these ϕ must be simultaneously true in every state that immediately precedes a state in which ψ becomes true. L_{gn} is a similar set, yet since greedy necessary orderings are weaker than necessary orderings, can sometimes contain sets that do not appear in L_{nec}, and therefore result in a larger overall set of conjunctive landmarks. Note that all necessary orderings are also greedy necessary orderings, and each conjunctive landmark that results from a set of necessary orderings is therefore a subset of some conjunctive landmark that results from greedy necessary orderings. We include conjunctive landmarks that result from necessary orderings as they result in stronger necessary orderings between the added conjunctive landmark and ψ. Landmark heuristics can then sometimes infer that these conjunctive landmarks must be reachieved if a landmark that they are ordered necessarily before has to be reachieved. This is not the case for the conjunctive landmarks derived from greedy-necessary orderings, as they only need to be achieved to make the landmarks they are ordered before true for the *first* time.

Algorithm 2: Choosing C for landmark generation.

$C = \emptyset$
$L = \mathsf{FindLandmarks}(\Pi^C_{\mathsf{ce}})$
repeat
    $C = C \cup \mathsf{SimultaneouslyAchieved}(L)$
    $L = \mathsf{FindLandmarks}(\Pi^C_{\mathsf{ce}})$
until $\mathsf{SimultaneouslyAchieved}(L) \subseteq C$

Our strategy for choosing C for landmark generation is shown in Algorithm 2. While new conjunctive landmarks $L = p_1 \wedge \cdots \wedge p_n$ can be discovered, the corresponding fluents $\pi_{\{p_1,\ldots,p_n\}}$ are added to Π^C_{ce} and the landmark computation step is repeated. Note that the process may go through several iterations, and is run until a fixpoint is reached, as the addition of the new π-fluents to the Π^C_{ce} task can result in the discovery of new landmarks. The process terminates when all the new conjunctive landmarks that are discovered already exist as π-fluents in Π^C_{ce}. We note that this method of choosing C does have the desired property mentioned above: all π-fluents introduced in Π^C_{ce} represent fact landmarks in Π^C_{ce} and conjunctive landmarks in the original task Π.

The above strategy works especially well in domains in which many landmarks have several landmarks that are necessarily or greedy necessarily ordered before them. One domain where this occurs is in Blocksworld (see an illustration in Figure 4), where the method is able to find extremely informative conjunctive landmarks that allow it to optimally solve more tasks than any other heuristic we tested.

6.3 Experiments

We consider the performance of the LM-cut heuristic $h^{\text{LM-cut}}$ on the Π^C and Π^C_{ce} compilations, and that of the admissible landmark cost-partitioning heuristic h^{LM} introduced by Karpas and Domshlak (2009) with different landmark generation schemes, including the landmarks obtained from Π^C_{ce}. $h^{\text{LM-cut}}$ is used with the A* search algorithm, while for h^{LM} we use LM-A*, a variant which is more effective when there are known fluent landmarks (Karpas & Domshlak, 2009). The benchmarks, computers, and time/memory limits are the same as those used in in Section 5.3.

	Informativeness		Coverage			
Domain	Π^C, **1.5**	Π^C_{ce}, **1.5**	Orig.	$x = 1$	Π^C, **1.5**	Π^C_{ce}, **1.5**
Airport	1 : 49.02	1 : 52.91	**28**	**28**	19	18
Barman-opt	1 : 1.06	1 : 1.12	4	4	4	4
Blocksworld	4.22 : 1	1 : 1.71	**28**	**28**	**28**	27
Depots	1 : 1.96	1 : 6.49	**7**	5	4	4
Driverlog	1 : 23.7	1 : 48.31	**13**	**13**	10	10
Elevators-opt11	1.65 : 1	1 : 1.03	**18**	**18**	16	15
Floortile-opt11	19.23 : 1	13.93 : 1	7	6	**12**	**12**
FreeCell	1.07 : 1	1 : 2.12	**15**	**15**	13	9
Grid	3.47 : 1	1 : 1.35	**2**	**2**	**2**	1
Gripper	1 : 1	1 : 1	**7**	**7**	6	6
Logistics00	1 : 9.38	1 : 10.47	**20**	**20**	16	15
Logistics98	1 : 7.69	1 : 18.87	**6**	**6**	2	3
Miconic	1 : 232.55	1 : 769.23	**141**	**141**	50	45
Mprime	13.08 : 1	1 : 1.13	22	22	**28**	22
Mystery	1.03 : 1	1.07 : 1	16	16	**17**	**17**
Nomystery-opt11	1 : 126.58	1 : 588.24	**14**	**14**	8	8
Openstacks-opt11	1 : 1	1 : 1	14	14	14	14
Parcprinter-opt11	1 : 1.14	1 : 4.23	**13**	**13**	**13**	12
Parking-opt11	-	-	**2**	1	1	0
Pathways-noneg	1 : 15.72	1 : 51.81	**5**	**5**	4	4
Pegsol-opt11	1.08 : 1	1.08 : 1	17	17	17	17
Pipes-NoTank	1 : 1.48	1 : 2.09	**17**	**17**	15	14
Pipes-Tank	1 : 1.30	1 : 2.25	**11**	10	8	7
PSR	1.04 : 1	1 : 1.03	49	49	49	49
Rovers	1 : 1.72	1 : 3.56	**7**	**7**	**7**	6
Satellite	1 : 3.77	1 : 33.33	**7**	**7**	6	6
Scanalyzer-opt11	1 : 1.36	1 : 1.06	**11**	**11**	4	5
Sokoban-opt11	1 : 1.28	1 : 1.31	20	20	20	20
Tidybot-opt11	1 : 4.79	1 : 13.19	**13**	**13**	7	6
TPP	1 : 5.14	1 : 1.70	6	6	6	6
Transport-opt11	1 : 1.87	1.35 : 1	6	6	6	**7**
Trucks	1 : 5.94	1 : 10.26	**10**	**10**	7	6
Visitall-opt11	1 : 3.86	1 : 3.32	10	10	10	10
Woodwork-opt11	4.07 : 1	1 : 2.36	**11**	**11**	7	5
Zenotravel	1 : 39.37	1 : 153.85	**12**	**12**	8	8
Total			**589**	584	444	418

Table 6: LM-cut with Π^C_{ce} and Π^C. The two columns on the left show the ratio of the summed number of heuristic evaluations for tasks solved by both configurations, comparing the standard LM-cut that results from $x = 1$ to LM-cut computed on Π^C and Π^C_{ce} with $x = 1.5$. For example, the first entry in the table, 1 : 49.02, shows that LM-cut computed on Π^C with a growth bound of $x = 1.5$ evaluates, in sum over the commonly solved tasks, nearly 50 times as many states as LM-cut computed on the standard delete relaxation. The last 4 columns show coverage. Column "Original" shows results obtained with Fast Downward's implementation of LM-cut (which applies only to the standard delete relaxation), while column "$x = 1$" shows the results of our implementation of LM-cut on the unmodified delete relaxation (with any differences between the two being purely due to implementation details). Entries in bold indicate the highest coverage in each domain, and in total.

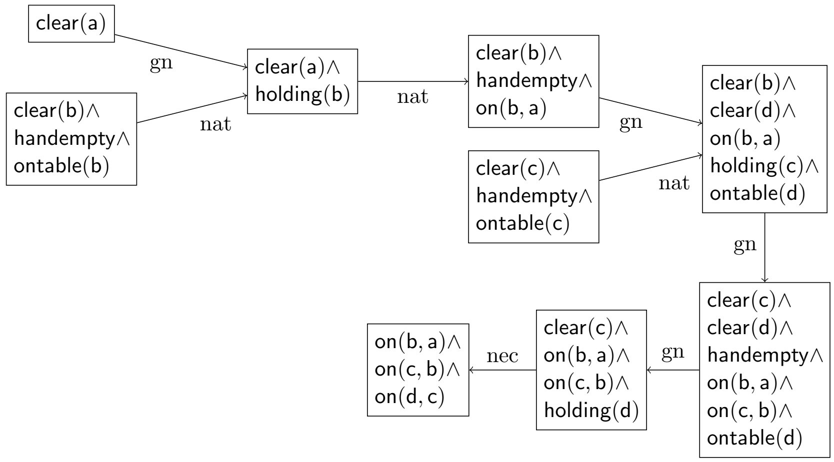


Figure 4: Landmarks graph found with the Π^C_{ce} compilation for a small Blocksworld task, in which all blocks are initially on the table and $G = \{\mathsf{on}(\mathsf{b},\mathsf{a}), \mathsf{on}(\mathsf{c},\mathsf{b}), \mathsf{on}(\mathsf{d},\mathsf{c})\}$. Some smaller conjunctive landmarks and single fluent landmarks are omitted.

6.3.1 LM-cut with Π^C and Π^C_{ce}

To evaluate the impact of using the Π^C and Π^C_{ce} compilations on LM-cut, we constructed the Π^C and Π^C_{ce} tasks following the same procedure as described in Section 5.3, repeatedly selecting conflicts until the increase in the size of the compiled task reached a fixed growth bound x. Conflict selection was based on $h^{\max}$ supporters rather than h^{add} supporters, as $h^{\max}$ plays a key role in the computation of LM-cut, and also resulted in better performance. Other than that, the procedure used to generate the Π^C and Π^C_{ce} tasks was the same. We tested each value of x from the set $\{1.5, 2, 2.5, 3\}$ for both Π^C and Π^C_{ce}. We observed that $x = 1.5$ dominated the larger values of x on a domain-by-domain and overall basis, and therefore report results only for these two configurations. The only exception to this is the Mystery domain, in which Π^C with $x = 2.5$ and $x = 3$ solved 18 tasks compared to 17 for $x = 1.5$.

Overall, the heuristic computed by the LM-cut algorithm on the standard delete relaxation Π^+ dominates that computed on Π^C and Π^C_{ce}, both in terms of informativeness and in terms of coverage. The first two columns of Table 6 show that for the large majority of domains, search using LM-cut computed on Π^C and Π^C_{ce} performs many more heuristic evaluations in tasks that are solved by both configurations. In the Airport domain, for instance, LM-cut on the standard delete relaxation requires approximately 50 times fewer heuristic evaluations to solve the same set of tasks as either Π^C or Π^C_{ce}. In most other domains, the situation is less extreme, but the standard delete relaxation continues to give the better heuristic estimates. The exceptions to this are the Blocksworld, Elevators, Floortile, FreeCell, Grid, Mprime, Mystery, Pegsol, PSR, Transport and Woodworking domains, in which at least one of Π^C or Π^C_{ce} yields more informative heuristic estimates than Π^+. Most impressively, Π^C and Π^C_{ce} give estimates that are respectively 19 and 14 times more informative than the estimates obtained from Π^+ in the Floortile domain, and estimates using Π^C are 13 times more informative than those of Π^+ on the Mprime domain. In terms of coverage, this translates into 12 tasks solved with both Π^C and Π^C_{ce} in the Floortile domain, compared to 7 for the standard version of LM-cut, and 28 tasks solved with Π^C in the

Mprime domain, compared to 22. In the Mystery domain, coverage is increased by 1. In all other domains, the coverage achieved with Π^C and Π^C_{ce}-based LM-cut is less than or equal to the coverage achieved with standard LM-cut. Overall, the standard version of LM-cut solves 589 planning tasks as compared to 445 for Π^C and 418 for Π^C_{ce}. Though a large part of this difference (90 tasks) comes from the Miconic domain, the difference in the remaining domains is still significant.

Comparing Π^C and Π^C_{ce}, it can be seen that the additional loss of information resulting from the treatment of conditional effects in LM-cut leads to worse heuristic estimates when using Π^C_{ce}. As expected from the theoretical result that Π^C_{ce} grows only linearly with the number of π-fluents, the number of π-fluents that are added to the task when using the Π^C_{ce} compilation is almost always higher than when using Π^C. However the treatment of conditional effects in LM-cut (described above) turns out to greatly degrade performance, and LM-cut using Π^C_{ce} is more informative than LM-cut using Π^C in only 4 domains.

6.3.2 Admissible Landmark Heuristics with Π^C_{ce} Landmarks

The admissible landmark heuristic, h^{LM}, uses action cost partitioning to derive heuristic values from a collection of (ordered) landmarks, distributing the cost of each action over the set of landmarks it achieves (Karpas & Domshlak, 2009). Cost partitioning can be done in different ways: optimal cost partitioning is tractable, and yields the best possible heuristic value for a given set of landmarks, but in practice is so slow that coverage suffers; uniform partitioning generally achieves a better time/informativeness trade-off, and therefore better coverage.

To evaluate the potential informativeness of the landmarks obtained with Π^C_{ce} using the iterative technique described in Section 6.2, we used these landmarks in the optimal cost partitioning setting, since this setting makes the best possible use of information present in the given landmarks. We compared their informativeness to that of the heuristic using landmarks obtained from the Π^m compilation with $m = 1$ and $m = 2$ and the sound and complete landmark generation algorithm (Keyder et al., 2010). The results are shown in the first two columns of Table 7. They show the ratio of total number of heuristic evaluations, per domain, over all tasks solved by all configurations, for h^{LM} using landmarks from Π^1 only to the heuristic using landmarks from Π^2 and Π^C_{ce}, respectively. Note that the landmarks generated from both the Π^C_{ce} and Π^2 compilations contain the landmarks obtained from Π^1 as a subset, and that h^{LM} with optimal partitioning over the Π^2 or Π^C_{ce} landmarks therefore dominates h^{LM} with optimal partitioning over Π^1 landmarks. Hence the ratio is always greater than 1.

In 9 of the 35 domains considered, neither the addition of Π^2 landmarks nor Π^C_{ce} landmarks leads to a more informative heuristic (cases in which both columns show the value 1). Of the remaining 26 domains, both schemes improve over Π^1 landmarks to an equal degree in 7, and Π^2 improves over Π^1 to a greater degree than Π^C_{ce} in 17. In one case, Blocksworld, Π^C_{ce} landmarks are much more informative than landmarks found by both the other methods, and improve informativeness over the baseline heuristic using only Π^1 landmarks by a factor of 122.

Uniform cost partitioning divides the cost of each action evenly over the set of landmarks that it achieves, rather than searching for a partitioning that maximizes the heuristic value

Domain	**Informativeness** (optimal partitioning)		**Coverage** (uniform partitioning)		
	Π^2	Π^C_{ce}	Π^1	Π^2	Π^C_{ce}
Airport	**1.03**	**1.03**	27	11	27
Barman-opt	-	-	4	4	4
Blocksworld	25.65	**122.31**	26	28	**32**
Depots	**4.28**	1.11	7	7	7
Driverlog	**1.02**	1.01	**10**	9	9
Elevators-opt11	**1.54**	**1.54**	12	12	12
Floortile-opt11	**3.28**	1.02	2	2	2
FreeCell	**1.32**	1.01	**60**	38	39
Grid	**1.12**	**1.12**	2	2	2
Gripper	1	1	**7**	6	**7**
Logistics00	**13.65**	**13.65**	20	**22**	**22**
Logistics98	**1.25**	**1.25**	3	3	3
Miconic	**3.91**	**3.91**	142	142	**143**
Mprime	**1.39**	1	20	20	20
Mystery	**2.17**	1	15	15	15
Nomystery-opt11	**3.08**	1.69	**20**	**20**	18
Openstacks-opt11	1	1	**12**	7	11
Parcprinter-opt11	**4.09**	1	**10**	8	**10**
Parking-opt11	**9.84**	1	**3**	0	0
Pathways-noneg	1	1	4	4	4
Pegsol-opt11	**1.25**	1	**17**	15	**17**
Pipes-NoTank	**1.37**	1.27	16	16	16
Pipes-Tank	**1.64**	1.13	**13**	10	11
PSR	**2.68**	1.23	49	49	49
Rovers	1	1	**6**	**6**	5
Satellite	1.06	1.06	6	6	6
Scanalyzer-opt11	**1.54**	1.01	**6**	3	**6**
Sokoban-opt11	1.02	1	**20**	14	18
Tidybot-opt11	**1.18**	1.05	**14**	9	**14**
TPP	1	1	6	6	6
Transport-opt11	1	1	6	6	6
Trucks	1	1	**8**	6	7
Visitall-opt11	1	1	**16**	9	9
Woodwork-opt11	**3.04**	1.16	**7**	4	5
Zenotravel	1	1	8	8	8
Total			**604**	527	570

Table 7: h^{LM} with landmarks generated from the delete relaxation (Π^1), Π^2 (Keyder et al., 2010) and Π^C_{ce}. The two columns on the left show the ratio of the summed number of heuristic evaluations for tasks solved by both configurations, comparing Π^2 and Π^C_{ce} to the baseline of using only landmarks from Π^1. Using optimal cost partitioning in h^{LM}, more landmarks can only yield better lower bounds, and indeed all the ratios are ≥ 1 (which is why we do not use a $m:1$ presentation, differently from the previous tables). The right-most three columns show coverage, using uniform cost partitioning. The heuristic with uniform partitioning solves more tasks than with optimal cost partitioning under all the landmark generation schemes considered. Entries in bold indicate best results, per domain and in total.

in each state. This can make the h^{LM} heuristic weaker, though typically not by much, but also makes it much faster to compute, leading to better coverage in general. We confirmed that uniform cost partitioning results in higher coverage than optimal cost partitioning in all domains for all three of the landmark generation schemes we considered.

The three right-most columns in Table 7 show the coverage achieved with h^{LM} and the three landmark generation schemes in this setting. It can be seen that using only Π^1 landmarks results in greater coverage than combining them with either Π^2 or Π^C_{ce} landmarks. Compared to the heuristic using Π^2 landmarks, using Π^C_{ce} landmarks solve as many or more tasks in every domain except the Nomystery and Rovers domains. Π^C_{ce} landmarks outperform Π^1 landmarks in two domains: Blocksworld, where using Π^C_{ce} landmarks the planner finds optimal solutions to 32 out of the 35 tasks, more than any other tested heuristic, and in Miconic, by 1 instance. In the other domains, the use of Π^C_{ce} landmarks either has no effect or worsens coverage compared to Π^1. Interestingly, while the informativeness of the LM-cut heuristic increases greatly with the Π^C and Π^C_{ce} compilations in the Floortile domain, there is no corresponding increase when the compilations are used to find landmarks. This is because few conjunctive landmarks, besides the goal, are found.

7. Conclusions and Open Questions

There is a long tradition of works attempting to devise heuristics taking into account *some* delete effects. However, techniques rendering h^+ perfect in the limit – and thus allowing to smoothly interpolate between h^+ and h^* – have been proposed only quite recently, by Haslum (2012) and Katz et al. (2013) respectively. We have extended Haslum's approach by introducing a new compilation method with linear (vs. worst-case exponential) growth, and demonstrated the machinery needed for using the approach to generate heuristics. Our evaluation shows that, in some domains, informedness can be dramatically improved at a small cost in terms of computational overhead.

The main open issue lies in our use of the words "in some domains" here. In most domains, the gain in informativeness is small, and in some domains overall performance suffers dramatically. While no domain-independent planning technique can work well in every domain, and while a simple portfolio approach (cf. column "PF" in Table 4) suffices to improve the state of the art in satisficing planning, the extent of the per-domain performance variation of our technique is dramatic. Can we obtain an understanding of what causes these phenomena, and ultimately exploit that understanding to devise more reliable/effective practical methods? Is the unchanged or worse performance in many domains due to fundamental limitations of the technique, or only due to its particular instantiation (especially the selection of π-fluents) as run in our experiments?

From a practical perspective, answering these questions comes down to the exploration of techniques for predicting the impact of adding π-fluents, and for making more informed decisions about *which* π-fluents to add. We have observed that changes in domain formulation, random reorderings, and small changes to the heuristic criteria used in π-fluent selection can have a large impact on both heuristic informativeness and coverage. Further research to formulate new heuristic criteria and improve existing ones therefore could, potentially, provide better performance across a wide range of domains. It might also be interesting to systematically explore the impact of random/arbitrary changes, and to attempt building complementary-strength compilations to be combined into effective portfolios.

From a theoretical perspective, we are currently approaching the above questions in terms of analyzing the conditions under which a small (polynomial-size) set of π-fluents suffices to render h^+ perfect. Applied to individual domains, this analysis offers a way of answering the question of whether a lack of performance improvement is due to an essential limitation or only due to choosing the "wrong" set of π-fluents. Our hope is to eventually obtain syntactic criteria (e. g., based on causal graph structure) that can be automatically applied to arbitrary planning task descriptions, serving to select the π-fluents (or to exclude subsets of π-fluents from consideration) in a targeted manner. Our first results in that direction have already been published in HSDIP'14 (Hoffmann, Steinmetz, & Haslum, 2014).

Our observations in optimal planning pose many questions for future work. A simple one is whether more effective Π^C_{ce} landmarks could be extracted by not restricting our techniques to adding only π-fluents guaranteed to be landmarks. The more daunting challenges regard LM-cut. Our observations suggest that the methods we use suffer greatly from suboptimal choices of precondition choice functions (PCFs). It would therefore be worthwhile to investigate new methods for obtaining better PCFs. Another important direction is to develop extensions of LM-cut to conditional effects that guarantee both admissibility and domination of $h^{\max}$. A simple yet impractical method is to multiply out the conditional effects (enumerating all subsets thereof). A more sophisticated method based on "context splitting", where distinctions between different occurences of the same action are introduced in a targeted manner only where necessary, has recently been proposed (Röger, Pommerening, & Helmert, 2014).

In summary, explicitly represented conjunctions clearly exhibit the potential to dramatically improve delete relaxation heuristics. But much remains to be done in order to understand and use them effectively.

Acknowledgments

Part of the work leading to this publication was carried out while Emil Keyder and Jörg Hoffmann were working at INRIA Grand Est, Nancy, France. NICTA is funded by the Australian Government through the Department of Communications and the Australian Research Council through the ICT Centre of Excellence Program. We thank the University of Freiburg for allowing us to use their computional resources.

Appendix A. Proofs

Theorem 3 ($h^+(\Pi^C)$ dominates $h^+(\Pi^C_{\mathsf{ce}})$) *Given a planning task Π and a set of conjunctions C, $h^+(\Pi^C) \geq h^+(\Pi^C_{\mathsf{ce}})$. There are cases in which the inequality is strict.*

Proof: This follows from the fact that any plan for Π^C is also a plan for Π^C_{ce}, yet the inverse is not the case. To show the first part, let $\sigma = \langle a_1^{C_1}, \dots, a_n^{C_n} \rangle$ be a plan for Π^C. We show that the same sequence of actions constitutes a plan for Π^C_{ce}, by showing by induction that $I^C[a_1^{C_1}, \dots, a_n^{C_n}] \subseteq I^C[a_1^C, \dots, a_n^C]_{\mathsf{ce}}$, where $I[\dots]_{\mathsf{ce}}$ denotes the result of applying a sequence of actions to the initial state I^C in Π^C_{ce}. Since the goal in both tasks is defined to be G^C, this shows the desired result. For the base case, the initial state in both Π^C and Π^C_{ce} is I^C, and the subset relation holds. For the inductive case, assume $I^C[a_1^{C_1}, \dots, a_{i-1}^{C_{i-1}}] \subseteq I^C[a_1^C, \dots, a_{i-1}^C]_{\mathsf{ce}}$. Since the precondition of a_i^C in Π^C_{ce} is a subset of the precondition of $a_i^{C_i}$ in Π^C for all a and all C_i, a_i^C can be applied in $I^C[a_1^C, \dots, a_{i-1}^C]_{\mathsf{ce}}$ by the induction hypothesis. We then need to show that all fluents added by $a_i^{C_i}$ in Π^C are also added by a_i^C in Π^C_{ce} when applied in $I^C[a_1^C, \dots, a_{i-1}^C]_{\mathsf{ce}}$. The add effect of $a_i^{C_i}$ in Π^C consists of the union of two sets, $(\mathsf{add}(a) \cup (\mathsf{pre}(a) \setminus \mathsf{del}(a)))^C$, which is also the add effect of a_i^C in Π^C_{ce} and therefore added, and $\{\pi_{c'} \mid c' \in C_i\}$. Since $a_i^{C_i}$ was applicable in Π^C, each of its preconditions $(\mathsf{pre}(a) \cup \bigcup_{c' \in C_i}(c' \setminus \mathsf{add}(a)))^C$ must be true in $I^C[a_1^{C_1}, \dots, a_{i-1}^{C_{i-1}}]$, and therefore in $I^C[a_1^C, \dots, a_{i-1}^C]_{\mathsf{ce}}$, by the induction hypothesis. For $c \in C$ (and therefore $c' \in C_i$, as $C_i \subseteq C$), a_i^C in Π^C_{ce} has a conditional effect with effect π_c and condition $(\mathsf{pre}(a) \cup (c \setminus \mathsf{add}(a)))^C$, which applies because its condition is a subset of the precondition of $a_i^{C_i}$ in Π^C. This shows the desired property.

For strictness, 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

$$
\begin{aligned}
a_{p_2} &: \langle \{p_1\}, \{p_2\}, \{r, p_1\}, \emptyset \rangle & a_r &: \langle \emptyset, \{r\}, \emptyset, \emptyset \rangle \\
a_{g_1} &: \langle \{p_1, r\}, \{g_1\}, \emptyset, \emptyset \rangle & a_{g_2} &: \langle \{p_2, r\}, \{g_2\}, \emptyset, \emptyset \rangle
\end{aligned}
$$

Let $C = \{c \subseteq F \mid |c| = 2\}$. The only optimal plan for both Π and Π^C is the sequence $\langle a_r, a_{g_1}, a_{p_2}, a_r, a_{g_2} \rangle$. In the case of Π^C this follows from the fact that the plan must include a_{g_1} and a_{g_2} as they are the only actions achieving the two goals, and therefore must achieve their precondition π-fluents $\pi_{\{p_1,r\}}$ and $\pi_{\{p_2,r\}}$, respectively. Each of these π-fluents can be achieved only with a_r, as no action achieves either of the p fluents without deleting r. There is no single representative of a_r that achieves both $\pi_{\{p_1,r\}}$ and $\pi_{\{p_2,r\}}$, as such a representative would have the precondition $\pi_{\{p_1,p_2\}}$, which is unreachable, since the only action achieving p_2 deletes p_1. A plan in Π^C therefore must contain a_{g_1}, a_{g_2}, at least two instances of a_r, and a_{p_2}.

This no longer holds, however, when considering Π^C_{ce}, for which the action sequence $\langle a_{p_2}, a_r, a_{g_1}, a_{g_2} \rangle$ is a plan that contains only 4 actions. In Π^C_{ce}, the two possible π-fluents added by a_r, $\pi_{\{p_1,r\}}$ and $\pi_{\{p_2,r\}}$, are treated independently, and a separate conditional effect is created for each, with the conditions p_1 and p_2 respectively. Once p_1 and p_2 have been achieved separately, a single application of the action a_r is then sufficient to achieve the two π-fluents, without making true the (unreachable) cross-context π-fluent $\pi_{\{p_1,p_2\}}$. In this and similar cases, there exist plans for Π^C_{ce} that are shorter than the minimum length plans for Π^C. ■

Given a STRIPS task $\Pi = \langle F, A, I, G, \mathsf{cost} \rangle$, the h^1 heuristic for a set P of fluents, is defined as follows (Bonet & Geffner, 2001):

$$
\begin{aligned}
h^1(p) &= \begin{cases} 0 & \text{if } p \in s \\ \min_{\{a \mid p \in \mathsf{add}(a)\}} h^1(\mathsf{pre}(a)) + \mathsf{cost}(a) & \text{otherwise} \end{cases} \\
h^1(P) &= \max_{p \in P} h^1(p)
\end{aligned}
$$

The value of the heuristic for a given planning task is taken to be the h^1 cost of the goal G, $h^1(\Pi) = h^1(G)$.

Lemma 1 *Given a planning task* Π *and* $C = \{c \in \mathcal{P}(F) \mid 1 < |c| \leq m\}$, $h^1(\Pi^C) = h^1(\Pi^m)$.

Proof: Let $\Pi = \langle F, A, I, G\rangle$. Π^m and Π^C are identical except for the action sets. We denote the action set of Π^m by $A^C(\Pi^m)$ and that of Π^C by $A^C(\Pi^C)$. There are no deletes and conditional effects in either of $A^C(\Pi^m)$ and $A^C(\Pi^C)$, so we will ignore these in what follows.

We first show that $h^1(\Pi^C) \leq h^1(\Pi^m)$, then that $h^1(\Pi^m) \leq h^1(\Pi^C)$. Each direction is based on the following two observations. First, for any STRIPS planning task, we can *split up* the actions over their singleton add effects, without affecting h^1. Precisely, given an action a and $p \in \mathsf{add}(a) \setminus \mathsf{pre}(a)$, we denote by $a[p]$ the action where $\mathsf{pre}(a[p]) = \mathsf{pre}(a)$ and $\mathsf{add}(a[p]) = \{p\}$. Replacing each a with all its split-up actions $a[p]$ (i. e. generating a split-up action $a[p]$ for every non-redundant add effect of a), h^1 remains the same. Second, say that every split-up action $a[p]$ in action set A is *dominated* by an action a' in action set A', i. e., $\mathsf{pre}(a') \subseteq \mathsf{pre}(a[p])$ and $\mathsf{add}(a') \supseteq \mathsf{add}(a[p])$. Then h^1 using A' is a lower bound on h^1 using A.

We now prove that $h^1(\Pi^C) \leq h^1(\Pi^m)$. For every $a \in A$ and $c \subseteq F$ so that $1 < |c| \leq m$, $\mathsf{del}(a) \cap c = \emptyset$, and $\mathsf{add}(a) \cap c \neq \emptyset$, $A^C(\Pi^m)$ contains the action a^c given by $\mathsf{pre}(a^c) = (\mathsf{pre}(a) \cup (c \setminus \mathsf{add}(a)))^C$, and $\mathsf{add}(a^c) = \mathsf{add}(a) \cup \{\pi_{c'} \mid c' \in C \wedge c' \subseteq (\mathsf{add}(a) \cup c)\}$. Let $p \in \mathsf{add}(a^c)$. If $p \in \mathsf{add}(a)$, then $a^{C'}$ for $C' = \emptyset$ dominates $a^c[p]$. Say $p = \pi_{c'}$ where $\pi_{c'} \notin \mathsf{pre}(a^c)$. To obtain a dominating action in $A^C(\Pi^C)$, we define:

$$C' := \{c'' \in C \mid \mathsf{del}(a) \cap c'' = \emptyset, \mathsf{add}(a) \cap c'' \neq \emptyset, c'' \subseteq c'\}$$

We have $C' \subseteq C$, and for all $c'' \in C'$ the conditions (1) $\mathsf{del}(a) \cap c'' = \emptyset \wedge \mathsf{add}(a) \cap c'' \neq \emptyset$ and (2) $\forall c \in C : ((c \subseteq c'' \wedge \mathsf{add}(a) \cap c \neq \emptyset) \implies c \in C')$ of Definition 2 are obviously satisfied. Thus $A^C(\Pi^C)$ contains the action $a^{C'}$ given by $\mathsf{pre}(a^{C'}) = (\mathsf{pre}(a) \cup \bigcup_{c'' \in C'}(c'' \setminus \mathsf{add}(a)))^C$ and $\mathsf{add}(a^{C'}) = (\mathsf{add}(a) \cup (\mathsf{pre}(a) \setminus \mathsf{del}(a)))^C \cup \{\pi_{c''} \mid c'' \in C'\}$. We now prove that (a) $\mathsf{pre}(a^{C'}) \subseteq \mathsf{pre}(a^c)$ and (b) $p = \pi_{c'} \in \mathsf{add}(a^{C'})$. Regarding (a), for every $c'' \in C'$ we have $c'' \setminus \mathsf{add}(a) \subseteq c' \setminus \mathsf{add}(a) \subseteq c \setminus \mathsf{add}(a)$, and thus $\bigcup_{c'' \in C'}(c'' \setminus \mathsf{add}(a)) \subseteq c' \setminus \mathsf{add}(a) \subseteq c \setminus \mathsf{add}(a)$. Regarding (b), we need to prove that $c' \in C$, $\mathsf{del}(a) \cap c' = \emptyset$, $\mathsf{add}(a) \cap c' \neq \emptyset$, and $c' \subseteq c'$. The first and the last of these properties are obvious, the second one is direct from construction. As for the third one, $\mathsf{add}(a) \cap c' \neq \emptyset$, this is true because otherwise we would have $c' \subseteq c \setminus \mathsf{add}(a)$ implying in contradiction to construction that $\pi_{c'} \in \mathsf{pre}(a^c)$.

It remains to prove that $h^1(\Pi^m) \leq h^1(\Pi^C)$. For every $a \in A$ and $C' \subseteq C$ with conditions (1) and (2) as stated above, $A^C(\Pi^C)$ contains the action $a^{C'}$. Let $p \in \mathsf{add}(a^{C'})$. If p is not a π-fluent, then either $p \in \mathsf{add}(a)$ or $p \in \mathsf{pre}(a) \setminus \mathsf{del}(a)$. The latter case is irrelevant (and no split-up action is generated); in the former case, setting $c := \mathsf{add}(a)$ we get that $a^{C'}[p]$ is dominated by a^c in $A^C(\Pi^m)$. Say $p = \pi_c$. Then at least one of the following cases must hold: (a) $c \in C'$ or (b) $c \subseteq (\mathsf{pre}(a) \setminus \mathsf{del}(a))$ or (c) $c \subseteq (\mathsf{add}(a) \cup (\mathsf{pre}(a) \setminus \mathsf{del}(a)))$ and $c \cap \mathsf{add}(a) \neq \emptyset$. In case (a), it follows directly by definition that $a^c \in A^C(\Pi^m)$ which dominates $a^{C'}[p]$. In case (b), $p = \pi_c \in \mathsf{pre}(a^{C'})$ so that case is irrelevant. In case (c), $c \cap \mathsf{add}(a) \neq \emptyset$ and $c \cap \mathsf{del}(a) = \emptyset$ so again $a^c \in A^C(\Pi^m)$ which dominates $a^{C'}[p]$. This concludes the proof. ■

Lemma 2 *Given a planning task* Π *and a set of non-unit conjunctions* C*,* $h^1(\Pi^C) \leq h^+(\Pi^C_{\mathsf{ce}})$*.*

Proof: Consider the planning task $\Pi^C_{\text{no-cc}}$ that is identical to Π^C except that it does not include cross-context preconditions. That is, the precondition of each action representative $a^{C'}$ is modified to be the following:

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

We show that (A) $h^1(\Pi^C) \leq h^1(\Pi^C_{\text{no-cc}})$, and (B) $h^1(\Pi^C_{\text{no-cc}}) \leq h^+(\Pi^C_{\mathsf{ce}})$.

We first prove (A). As in the proof of Lemma 1, it suffices to prove that, for every split-up action $a^{C'}[p]$ of $\Pi^C_{\text{no-cc}}$, there exists a dominating action in Π^C. If p is not a π-fluent, then $a^{C''}$ in Π^C for $C'' = \emptyset$ dominates $a^{C'}[p]$. Otherwise, say $p = \pi_{c'}$. Then at least one of the following cases must hold: (a) $c' \in C'$ or (b) $c' \subseteq (\mathsf{add}(a) \cup (\mathsf{pre}(a) \setminus \mathsf{del}(a)))$. In case (b), again $a^{C''}$ in Π^C for $C'' = \emptyset$ dominates $a^{C'}[p]$. In case (a), to obtain a dominating action $a^{C''}$ in Π^C, we define

$$C'' := \{c'' \in C \mid \mathsf{del}(a) \cap c'' = \emptyset, \mathsf{add}(a) \cap c'' \neq \emptyset, c'' \subseteq c'\}$$

All $c'' \in C''$ satisfy the conditions (1) $\mathsf{del}(a) \cap c'' = \emptyset \wedge \mathsf{add}(a) \cap c'' \neq \emptyset$ and (2) $\forall c \in C : ((c \subseteq c'' \wedge \mathsf{add}(a) \cap c \neq \emptyset) \implies c \in C')$ of Definition 2, so indeed $a^{C''}$ is an action in Π^C. We obviously have $c' \in C''$ and thus $p \in \mathsf{add}(a^{C''})$. It remains to prove that $\mathsf{pre}(a^{C''}) \subseteq \mathsf{pre}(a^{C'}[p])$. This is so, intuitively, because C'' corresponds to the single conjunction c' (plus subsumed conjunctions) and hence no cross-context fluents arise. Specifically, for every $c'' \in C''$ we have $c'' \setminus \mathsf{add}(a) \subseteq c' \setminus \mathsf{add}(a)$. Thus $\mathsf{pre}(a^{C''}) = (\mathsf{pre}(a) \cup \bigcup_{c'' \in C''} (c'' \setminus \mathsf{add}(a)))^C = (\mathsf{pre}(a) \cup (c' \setminus \mathsf{add}(a)))^C$. The latter is obviously contained in $\mathsf{pre}(a^{C'}[p])$, concluding the proof of (A).

It remains to prove (B). Since $h^1(\Pi^C_{\text{no-cc}}) \leq h^+(\Pi^C_{\text{no-cc}})$, it suffices to prove that $h^+(\Pi^C_{\text{no-cc}}) \leq h^+(\Pi^C_{\mathsf{ce}})$. Consider a state s, and a relaxed plan σ^+_{ce} for s in Π^C_{ce}. For each action a_{ce} in σ^+_{ce}, representing action a of the original task Π, let C' be the set of conjunctions c whose π-fluents are added by a_{ce} during the execution of σ^+_{ce} in Π^C_{ce}. C' obviously qualifies for constraint (1) in Definition 2; it qualifies for constraint (2) because any conditional effect for c with that property will be triggered by a_{ce} if the conditional effect for a suitable c' is triggered. Thus $\Pi^C_{\text{no-cc}}$ includes the representative $a^{C'}$ of a. Define the action sequence σ^+ in $\Pi^C_{\text{no-cc}}$ to be the sequence of these $a^{C'}$. Obviously, $a^{C'}$ in σ^+ adds the same fluents as a_{ce}, and its precondition is the union of that of a_{ce} and of its conditional effects that fire. Thus σ^+ is a relaxed plan for $\Pi^C_{\text{no-cc}}$. It follows that $h^+(\Pi^C_{\text{no-cc}}) \leq h^+(\Pi^C_{\mathsf{ce}})$ as desired. ∎

Example 4 *Consider the STRIPS planning task* Π *with variables* $\{i, p, q, r, z, g_1, g_2, g_3\}$*, initial state* $I = \{i\}$*, goal* $G = \{g_1, g_2, g_3\}$*, and actions* A *as follows:*

Name	pre	add	del	ce	cost
a_i^{qz}	$\{i\}$	$\{q, z\}$	$\emptyset$	$\emptyset$	4
a_i^{r}	$\{i\}$	$\{r\}$	$\emptyset$	$\emptyset$	1
$a_{iq}^{p\neg z}$	$\{i, q\}$	$\{p\}$	$\{z\}$	$\emptyset$	1
$a_r^{p\neg z}$	$\{r\}$	$\{p\}$	$\{z\}$	$\emptyset$	4
$a_{pz}^{g_1}$	$\{p, z\}$	$\{g_1\}$	$\emptyset$	$\emptyset$	1
$a_{iq}^{g_2}$	$\{i, q\}$	$\{g_2\}$	$\emptyset$	$\emptyset$	1
$a_r^{g_3}$	$\{r\}$	$\{g_3\}$	$\emptyset$	$\emptyset$	1

We set $C = \{\{i, q\}, \{p, z\}\}$. The only operator adding part of $\{i, q\}$ is a_i^{qz} which adds q. The only operators adding part of $\{p, z\}$ are a_i^{qz} which adds z, and $a_{iq}^{p\neg z}, a_r^{p\neg z}$ which add p; since $a_{iq}^{p\neg z}$ and $a_r^{p\neg z}$ both delete z, they cannot be used to establish the conjunction $\{p, z\}$. Thus the actions of Π^C_{ce} are:

Name	pre	add	del	ce	cost
a_i^{qz}	$\{i\}$	$\{q, z\}$	$\emptyset$	$\mathsf{ce}(a_i^{qz})$	4
a_i^{r}	$\{i\}$	$\{r\}$	$\emptyset$	$\emptyset$	1
$a_{iq}^{p\neg z}$	$\{i, q, \pi_{i,q}\}$	$\{p\}$	$\emptyset$	$\emptyset$	1
$a_r^{p\neg z}$	$\{r\}$	$\{p\}$	$\emptyset$	$\emptyset$	4
$a_{pz}^{g_1}$	$\{p, z, \pi_{p,z}\}$	$\{g_1\}$	$\emptyset$	$\emptyset$	1
$a_{iq}^{g_2}$	$\{i, q, \pi_{i,q}\}$	$\{g_2\}$	$\emptyset$	$\emptyset$	1
$a_r^{g_3}$	$\{r\}$	$\{g_3\}$	$\emptyset$	$\emptyset$	1

where $\mathsf{ce}(a_i^{qz})$ contains two conditional effects:

Name	c	add	del
$e_i^{\pi_{i,q}}$	$\{i\}$	$\{\pi_{i,q}\}$	$\emptyset$
$e_p^{\pi_{p,z}}$	$\{p\}$	$\{\pi_{p,z}\}$	$\emptyset$

Clearly, with respect to $h^{\max}$, the π-fluents in the preconditions of $a_{iq}^{p\neg z}$, $a_{pz}^{g_1}$, and $a_{iq}^{g_2}$ dominate the respective other preconditions of these actions (as pointed out in Section 5.1, in our implementation we actually remove the other preconditions). Thus LM-cut's justification graph on Π^C_{ce} would have the structure shown in Figure 5.

We have $h^{\max}(\Pi^C_{\mathsf{ce}}) = 10$ due to the cost of achieving g_1. As for $h^+(\Pi^C_{\mathsf{ce}})$, to construct a plan for Π^C_{ce} the only choice we have is how to establish p. We can use $a_i^{qz}, a_{iq}^{p\neg z}$, or we can use $a_i^r, a_r^{p\neg z}$. In the latter case, we make do with a single application of the conditional-effects action a_i^{qz}, by the relaxed plan $\sigma_1 = \langle a_i^r, a_r^{p\neg z}, a_i^{qz}, a_{pz}^{g_1}, a_{iq}^{g_2}, a_r^{g_3}\rangle$, whose cost is 12. In the former case, we must use a_i^{qz} twice – first for $\pi_{i,q}$, then for $\pi_{p,z}$ – yielding the relaxed plan $\sigma_2 = \langle a_i^{qz}, a_{iq}^{p\neg z}, a_i^{qz}, a_{pz}^{g_1}, a_{iq}^{g_2}, a_i^r, a_r^{g_3}\rangle$, whose cost is 13. Thus $h^+(\Pi^C_{\mathsf{ce}}) = 12$. Since, in the execution of σ_1, the only delete is that of $a_r^{p\neg z}$, which is not true anyhow in the state of execution, σ_1 also solves the original task Π and we get $h^*(\Pi) = h^+(\Pi^C_{\mathsf{ce}}) = 12$.

Now consider LM-cut, and say that we produced the cut for the conditional-effects action a_i^{qz} that connects p to $\pi_{p,z}$ via the conditional effect $e_p^{\pi_{p,z}}$. The two options discussed in Section 6.1 are to (A) reduce the cost of a_i^{qz} globally, sticking to the original definition of LM-cut; or to (B) reduce the cost only of $e_p^{\pi_{p,z}}$, because the other conditional effect $e_i^{\pi_{i,q}}$ is part of an optimal-cost path to $e_p^{\pi_{p,z}}$ and thus serves to justify its $h^{\max}$ value. Each of these options violates one of the essential properties of LM-cut:

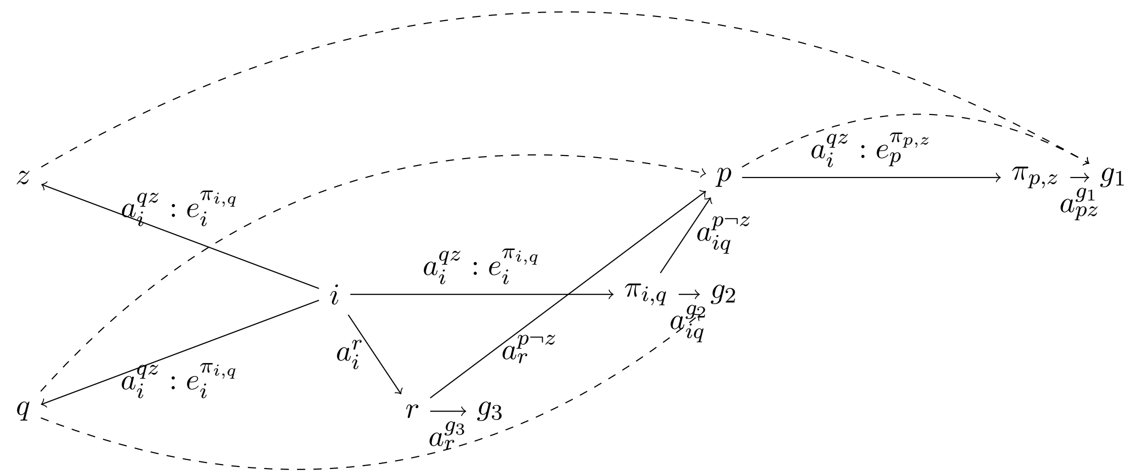


Figure 5: Illustration of LM-cut justification graphs for Π^C_{ce} in Example 4. The dashed edges correspond to preconditions that are not critical ($h^{\max}$-maximizing) at the start, but that become critical at some point during the execution of LM-cut.

(A) In this configuration, LM-cut produces the cuts $\{a_{pz}^{g_1}\}$ *[cost 1],* $\{a_i^{qz} : e_p^{\pi_{p,z}}\}$ *[cost 4],* $\{a_r^{g_3}\}$ *[cost 1],* $\{a_r^{p\neg z}, a_{iq}^{p\neg z}\}$ *[cost 1],* $\{a_{iq}^{g_2}\}$ *[cost 1],* $\{a_i^r\}$ *[cost 1]. Note here that, after the cut* $\{a_i^{qz} : e_p^{\pi_{p,z}}\}$*, the cost of* a_i^{qz} *is reduced to 0 globally; in particular, the cut* $\{a_i^{qz} : e_i^{\pi_{i,q}}\}$ *is not produced. We get the heuristic value* $h^{\text{LM-cut}} = 9 < h^{\max}(\Pi^C_{\mathsf{ce}}) = 10$*, so here LM-cut does not dominate* $h^{\max}$*.*

(B) In this configuration, LM-cut can produce the following cuts. At the start, for every possible precondition choice function (pcf), we get the cuts $\{a_{pz}^{g_1}\}$ *[cost 1] and* $\{a_i^{qz} : e_p^{\pi_{p,z}}\}$ *[cost 4]. Now* $h^{\max}$ *is 5 for each of* g_1, g_2*; say the pcf selects* g_2*, and we get the cut* $\{a_{iq}^{g_2}\}$ *[cost 1]. Now, the pcf has to select* g_1*, getting the cut* $\{a_r^{p\neg z}, a_{iq}^{p\neg z}\}$ *[cost 1]. Then,* $h^{\max}$ *is 4 for each of* g_1, g_2*; say the pcf selects* g_1*. Say further that the pcf selects* $\pi_{p,z}$ *for* $a_{pz}^{g_1}$ *(another choice would be* z*), and selects* $\pi_{i,q}$ *for* $a_{iq}^{p\neg z}$ *(another choice would be* q*), thus remaining in the non-dashed part of Figure 5. Then we get the cut* $\{a_i^{qz} : e_i^{\pi_{i,q}}, a_r^{p\neg z}\}$ *[cost 3] because* g_1 *can be reached at 0 cost from both* p *and* $\pi_{i,q}$ *(we would get the same cut for any pcf selecting* g_1*, at this point). Now,* $h^{\max}$ *is 2 for* g_3 *and 1 for each of* g_1, g_2*, so we get the cut* $\{a_r^{g_3}\}$ *[cost 1]. At this point,* $h^{\max}$ *is 1 for all goal facts; say LM-cut selects* g_3*, and thus we get the cut* $\{a_i^r\}$ *[cost 1] because that is the only way to achieve* r*. Then finally* $h^{\max}$ *is 1 for* g_2 *only, yielding the cut* $\{a_i^{qz} : e_i^{\pi_{i,q}}\}$ *[cost 1]. Overall, we get the heuristic value* $h^{\text{LM-cut}} = 13 > h^*(\Pi) = h^+(\Pi^C_{\mathsf{ce}}) = 12$*, so here LM-cut is not admissible.*

Bibliography

Baier, J. A., & Botea, A. (2009). Improving planning performance using low-conflict relaxed plans. In Gerevini, A., Howe, A., Cesta, A., & Refanidis, I. (Eds.), *Proceedings of the*

19th International Conference on Automated Planning and Scheduling (ICAPS'09), pp. 10–17, Thessaloniki, Greece. AAAI Press.

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

Bonet, B., & Helmert, M. (2010). Strengthening landmark heuristics via hitting sets. In Coelho, H., Studer, R., & Wooldridge, M. (Eds.), *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI'10)*, pp. 329–334, Lisbon, Portugal. IOS Press.

Bonet, B., Palacios, H., & Geffner, H. (2009). Automatic derivation of memoryless policies and finite-state controllers using classical planners. In Gerevini, A., Howe, A., Cesta, A., & Refanidis, I. (Eds.), *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS'09)*, pp. 34–41, Thessaloniki, Greece. AAAI Press.

Bylander, T. (1994). The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, *69*(1–2), 165–204.

Cai, D., Hoffmann, J., & Helmert, M. (2009). Enhancing the context-enhanced additive heuristic with precedence constraints. In Gerevini, A., Howe, A., Cesta, A., & Refanidis, I. (Eds.), *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS'09)*, pp. 50–57, Thessaloniki, Greece. AAAI Press.

Do, M. B., & Kambhampati, S. (2001). Sapa: A domain-independent heuristic metric temporal planner. In Cesta, A., & Borrajo, D. (Eds.), *Recent Advances in AI Planning. 6th European Conference on Planning (ECP'01)*, Lecture Notes in Artificial Intelligence, pp. 109–120, Toledo, Spain. Springer-Verlag.

Fox, M., & Long, D. (2001). STAN4: A hybrid planning strategy based on subproblem abstraction. *The AI Magazine*, *22*(3), 81–84.

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

Gazen, B. C., & Knoblock, C. (1997). Combining the expressiveness of UCPOP with the efficiency of Graphplan. In Steel, S., & Alami, R. (Eds.), *Recent Advances in AI Planning. 4th European Conference on Planning (ECP'97)*, Lecture Notes in Artificial Intelligence, pp. 221–233, Toulouse, France. Springer-Verlag.

Gerevini, A., Saetti, A., & Serina, I. (2003). Planning through stochastic local search and temporal action graphs. *Journal of Artificial Intelligence Research*, *20*, 239–290.

Haslum, P., & Geffner, H. (2000). Admissible heuristics for optimal planning. In Chien, S., Kambhampati, R., & Knoblock, C. (Eds.), *Proceedings of the 5th International Conference on Artificial Intelligence Planning Systems (AIPS'00)*, pp. 140–149, Breckenridge, CO. AAAI Press.

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., & Refanidis, I. (Eds.), *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS'09)*, pp. 354–357, Thessaloniki, Greece. AAAI Press.

Haslum, P. (2012). Incremental lower bounds for additive cost planning problems. In Bonet, B., McCluskey, L., Silva, J. R., & Williams, B. (Eds.), *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*, pp. 74–82, Sao Paulo, Brasil. AAAI Press.

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

Helmert, M., & Domshlak, C. (2009). Landmarks, critical paths and abstractions: What's the difference anyway?. In Gerevini, A., Howe, A., Cesta, A., & Refanidis, I. (Eds.), *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS'09)*, pp. 162–169, Thessaloniki, Greece. AAAI Press.

Helmert, M., & Geffner, H. (2008). Unifying the causal graph and additive heuristics. In Rintanen, J., Nebel, B., Beck, J. C., & Hansen, E. (Eds.), *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS'08)*, pp. 140–147, Sydney, Australia. AAAI Press.

Hoffmann, J. (2005). Where 'ignoring delete lists' works: Local search topology in planning benchmarks. *Journal of Artificial Intelligence Research*, *24*, 685–758.

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

Hoffmann, J., Porteous, J., & Sebastia, L. (2004). Ordered landmarks in planning. *Journal of Artificial Intelligence Research*, *22*, 215–278.

Hoffmann, J., Steinmetz, M., & Haslum, P. (2014). What does it take to render $h^+(\pi^c)$ perfect?. In *Proceedings of the 6th Workshop on Heuristics and Search for Domain Independent Planning, at ICAPS'14*.

Karpas, E., & Domshlak, C. (2009). Cost-optimal planning with landmarks. In Boutilier, C. (Ed.), *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI'09)*, pp. 1728–1733, Pasadena, California, USA. Morgan Kaufmann.

Katz, M., Hoffmann, J., & Domshlak, C. (2013). Who said we need to relax *all* variables?. In Borrajo, D., Fratini, S., Kambhampati, S., & Oddi, A. (Eds.), *Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS'13)*, pp. 126–134, Rome, Italy. AAAI Press.

Keyder, E., & Geffner, H. (2008). Heuristics for planning with action costs revisited. In Ghallab, M. (Ed.), *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI'08)*, pp. 588–592, Patras, Greece. Wiley.

Keyder, E., & Geffner, H. (2009). Trees of shortest paths vs. Steiner trees: Understanding and improving delete relaxation heuristics. In Boutilier, C. (Ed.), *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI'09)*, pp. 1734–1749, Pasadena, California, USA. Morgan Kaufmann.

Keyder, E., Hoffmann, J., & Haslum, P. (2012). Semi-relaxed plan heuristics. In Bonet, B., McCluskey, L., Silva, J. R., & Williams, B. (Eds.), *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*, pp. 128–136, Sao Paulo, Brasil. AAAI Press.

Keyder, E., Richter, S., & Helmert, M. (2010). Sound and complete landmarks for And/Or graphs. In Coelho, H., Studer, R., & Wooldridge, M. (Eds.), *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI'10)*, pp. 335–340, Lisbon, Portugal. IOS Press.

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

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

Röger, G., Pommerening, F., & Helmert, M. (2014). Optimal planning in the presence of conditional effects: Extending LM-Cut with context-splitting. In Schaub, T. (Ed.), *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI'14)*, Prague, Czech Republic. IOS Press. To appear.