Journal of Artificial Intelligence Research 56 (2016) 269–327 Submitted 12/15; published 05/16

Combining the Delete Relaxation with Critical-Path Heuristics: A Direct Characterization

Maximilian Fickert S9MAFICK@STUD.UNI-SAARLAND.DE
Jörg Hoffmann HOFFMANN@CS.UNI-SAARLAND.DE
Marcel Steinmetz STEINMETZ@CS.UNI-SAARLAND.DE
Saarland University, Saarbrücken, Germany

Abstract

Recent work has shown how to improve delete relaxation heuristics by computing relaxed plans, i. e., the h^{FF} heuristic, in a compiled planning task Π^C which represents a given set C of fact conjunctions explicitly. While this compilation view of such partial delete relaxation is simple and elegant, its meaning with respect to the original planning task is opaque, and the size of Π^C grows exponentially in $|C|$. We herein provide a direct characterization, without compilation, making explicit how the approach arises from a combination of the delete-relaxation with critical-path heuristics. Designing equations characterizing a novel view on h^+ on the one hand, and a generalized version h^C of h^m on the other hand, we show that $h^+(\Pi^C)$ can be characterized in terms of a combined h^{C+} equation. This naturally generalizes the standard delete-relaxation framework: understanding that framework as a relaxation over singleton facts as atomic subgoals, one can refine the relaxation by using the conjunctions C as atomic subgoals instead. Thanks to this explicit view, we identify the precise source of complexity in $h^{\mathrm{FF}}(\Pi^C)$, namely maximization of sets of supported atomic subgoals during relaxed plan extraction, which is easy for singleton-fact subgoals but is **NP**-complete in the general case. Approximating that problem greedily, we obtain a polynomial-time h^{CFF} version of $h^{\mathrm{FF}}(\Pi^C)$, superseding the Π^C compilation, and superseding the modified Π^C_{ce} compilation which achieves the same complexity reduction but at an information loss. Experiments on IPC benchmarks show that these theoretical advantages can translate into empirical ones.

1. Introduction

The *delete relaxation* in classical planning (McDermott, 1999; Bonet & Geffner, 2001) originates from work using a STRIPS representation, where state variables are Boolean, action effects are conjunctions of literals, and action preconditions as well as the goal are restricted to conjunctions of positive literals (*facts*). The relaxation assumes that there are no negative ("delete") effect literals, hence the name. More generally, i. e., with non-Boolean state variables, this amounts to assuming that state variables accumulate their values, rather than switching between them. While optimal delete-relaxed planning is still **NP**-hard, satisficing delete-relaxed planning is polynomial-time (Bylander, 1994). *Relaxed plan heuristics*, employing satisficing delete-relaxed planning for the generation of inadmissible heuristic functions, have proved highly successful (e. g. Hoffmann & Nebel, 2001; Gerevini, Saetti, & Serina, 2003; Richter & Westphal, 2010). They form a key ingredient for successful *satisficing planning* (not giving an optimality guarantee), in particular in almost all winners of the satisficing-planning tracks of the *International Planning Competitions (IPC)*.

Despite this success, the pitfalls of delete-relaxation heuristics (for example, ignoring resource consumption) have been known since a long time, and there have been intense efforts from the outset to "take some deletes into account" (e. g., Fox & Long, 2001; Do & Kambhampati, 2001; Gerevini et al., 2003; Helmert, 2004; van den Briel, Benton, Kambhampati, & Vossen, 2007; Helmert & Geffner, 2008; Cai, Hoffmann, & Helmert, 2009; Baier & Botea, 2009; Coles, Coles, Fox, & Long, 2013; Alcázar, Borrajo, Fernández, & Fuentetaja, 2013). Two recent approaches, *red-black planning* (Domshlak, Hoffmann, & Katz, 2015) and what we refer to as *explicit conjunctions*, were devised that allow to do so systematically: *partial delete relaxation*, that can in principle render the heuristic estimate perfect. We herein focus on explicit conjunctions.

To summarize our results in what follows, we need some basic notation and concepts well known in the planning community. We denote the *perfect heuristic*, returning the precise remaining cost, by h^*; the heuristic returning the cost of an optimal relaxed plan by h^+; and relaxed plan heuristics by h^{FF} (from the system FF where these were first introduced, see Hoffmann & Nebel, 2001). We assume the common method of computing relaxed plan heuristics by *relaxed plan extraction* on a *best-supporter function* (Keyder & Geffner, 2008). We will assume by default that the best-supporter function is derived from the *max heuristic* $h^{\max}$ (Bonet & Geffner, 2001). Relaxed plan extraction on a $h^{\max}$ best-supporter function is, on unit-cost problems, equivalent to the original formulation in terms of *relaxed planning graphs* by Hoffmann and Nebel (2001).

Explicit conjunctions were first introduced by Haslum (2009) as a compilation-based characterization of the *critical-path* relaxation introduced earlier on by Haslum and Geffner (2000). This relaxation assumes that, from any goal set of facts (a fact conjunction that needs to be achieved at some point during a plan), it suffices to achieve the most costly subgoal (subconjunction) of size at most m. Here, m is a parameter and the corresponding heuristic is denoted h^m. The special case $m = 1$ is equivalent to the max heuristic, i. e., $h^1 = h^{\max}$. Haslum's (2009) compiled planning task Π^m represents each size-$\leq m$ conjunction c via a newly introduced *π-fluent* π_c, and arranges the preconditions and effects on these π-fluents such that $h^1(\Pi^m) = h^m$.

Subsequently, Haslum (2012) introduced the modified compilation Π^C, which admits arbitrary sets C of conjunctions and guarantees admissibility of h^+ on the compilation, i. e., that $h^+(\Pi^C) \leq h^*$, which is not true of $h^+(\Pi^m)$. He furthermore showed that this method converges to h^*, i. e., that $h^+(\Pi^C) = h^*$ for appropriately chosen C. The downside of Π^C is that its size is worst-case exponential in $|C|$: Say that an action a can *support* a conjunction c if c can be regressed over a, i. e., a makes part of c true and makes none of c false. In order to guarantee admissibility of $h^+(\Pi^C)$, Π^C explicitly enumerates all subsets $C' \subseteq C$ of conjunctions c that any *occurrence* of an action a in the plan may support. This size explosion was tackled by the Π^C_{ce} compilation (Keyder, Hoffmann, & Haslum, 2012, 2014), which handles each possibly-supported c by a separate conditional effect. Π^C_{ce} still guarantees convergence, yet loses information as it ignores *cross-context* conditions, i. e., precondition π-fluents which arise only from the combination of several supported $c \in C'$.

One thing evident from this history of explicit conjunctions is that the resulting heuristic functions combine information inherent in the critical-path relaxation, with information inherent in the delete relaxation. But in what way, exactly? While the compilation view is simple and elegant, its meaning with respect to the original planning task is opaque.

A first simple observation is that the step from size-$\leq m$ conjunctions to arbitrary conjunctions C is not specific to the, historically, simultaneous step from critical-path to partial delete relaxation heuristics. The h^m heuristic is straightforwardly generalizable to consider, not all size-$\leq m$ subgoals, but an arbitrary set C of subgoals. Intuitively, we can choose any set of *atomic* subgoals, to be kept intact by the critical-path relaxation. We denote the generalized heuristic by h^C.

A second simple observation is that the delete relaxation can be viewed as "allowing to achieve each fact in separation": achieving the facts p in a goal set/conjunction one-by-one, the negative effects within each step do not matter because they concern facts other than p. Given this, h^+ can be characterized in terms of an equation related to that characterizing h^1, but requiring to achieve *all* size-1 subgoals instead of achieving only the single most costly one.

Putting the two observations together, we obtain a natural generalization of the standard delete-relaxation framework: *where the standard delete relaxation, like h^1, works with singleton facts as its atomic subgoals, one can use the conjunctions C as atomic subgoals instead*. We spell this out in the form of two heuristic functions we denote by h^{C+} and h^{CFF}:

(1) The h^1-like equation characterizing h^+ translates into a h^C-like equation characterizing h^{C+}, equivalent to $h^+(\Pi^C)$.

(2) Relaxed plan extraction to obtain h^{FF} from a h^1 best-supporter function translates into relaxed plan extraction to obtain h^{CFF} (a relaxed plan for Π^C) from a h^C best-supporter function.

Result (1) is of theoretical interest. It formulates $h^{C+} = h^+(\Pi^C)$ without a compilation, shedding a different light on Haslum's (2012) equivalent proposal. Result (2) has more immediate practical ramifications. It provides an alternative technique to obtain relaxed plans for Π^C, exponentially more efficient in the worst case because it does not require to exhaustively enumerate subsets $C' \subseteq C$. The h^C best-supporter function can be computed in time polynomial in $|C|$, similar to h^m. Intuitively, as the critical path pertains to single atomic subgoals, there is no need to enumerate combinations of atomic subgoals here. For relaxed plan extraction, we avoid such enumeration by identifying and tackling the precise source of complexity.

Relaxed plan extraction on h^C is more complex than relaxed plan extraction on h^1 for two reasons, of which the first corresponds to Haslum's (2012) observations, yet the second one only becomes apparent in our new direct formulation:

(a) To ensure convergence, allowing $h^{\text{FF}}(\Pi^C)$ to find real plans in the limit, we need to collect a set of *action occurrences*, i. e., pairs (a, C') of action a and set of supported conjunctions C', instead of just a set of actions as in the standard setting (where actions merely support the facts in their direct effect).

(b) In every occurrence (a, C') selected during relaxed plan extraction, C' should be as large as possible, as atomic subgoals (conjunctions) may now overlap, incurring the risk of dramatic overestimation (e. g., achieving every fact pair in the global goal separately). But it is **NP**-complete to find a cardinality-maximal C' that does not incur infeasible cross-context conditions.

To understand this, consider an action a which can support at least one subgoal $c \in C$ during relaxed plan extraction. We need to decide which other current subgoals $c' \in C$

to support with that same action occurrence. In the standard setting, where c and c' are singletons (e. g. $c = \{p\}$ and $c' = \{q\}$), one can simply support all c' in a's positive effects (e. g. $add(a) = \{p, q\}$). In the general case, for arbitrary C, this is no longer so because the part of c' not contained in a's positive effects gets propagated into the new subgoal regressing over a (e. g. r_1 for $c'_1 = \{q, r_1\}$ and r_2 for $c'_2 = \{q, r_2\}$). Combinations of several c' may incur cross-context conditions (e. g. $\{r_1, r_2\}$) harder to achieve than the conjunctions c' themselves in isolation.

We will refer to (b), maximization of $|C'|$ during relaxed plan extraction, as the *subgoal-support selection problem*. It is striking here that the underlying phenomena – supported conjunction sets C' and cross-context conditions – were previously identified and addressed, yet not put into the specific context relevant for relaxed plan extraction. From this perspective, Haslum (2012) solves an **NP**-complete problem enumeratively, putting all solution candidates (choices of C') into memory in the form of the compiled task Π^C; and Keyder et al.'s (2012, 2014) Π^C_{ce} compilation over-simplifies the problem, ignoring cross-context conditions completely. Yet, if cardinality-maximal C' is the real issue, why don't we simply select a subset-maximal C' instead? Using a simple greedy approximation to this effect, we obtain h^{CFF}, extracting relaxed plans for Π^C in polynomial time without having to ignore cross-context conditions. That heuristic supersedes, from a theoretical perspective and as far as h^{FF} is concerned, both the Π^C and Π^C_{ce} compilations.

It is at this point necessary to mention that our observation (2) is not entirely new. Alcázar et al. (2013) already devised a heuristic they call FF^m, extracting a relaxed plan from a h^m best-supporter function (they implement this for $m = 2$). This is essentially (2), without the generality of an arbitrary conjunction set C (which can easily be fixed). However, Alcazar et al.'s work was conducted as part of a much broader scope addressing heuristic search regression planning, and does not investigate (2) in detail. The design of FF^m does not recognize, and therefore not appropriately address, (a) and (b). Regarding (b), FF^m always selects a single conjunction $C' = \{c\}$ to support, a trivial approximation of the **NP**-complete $|C'|$-maximization problem, which may lead to dramatic overestimation. The overestimation is counter-acted given that FF^m also disregards (a), collecting a set of actions as in standard relaxed plan extraction methods. But that loses convergence – the value of FF^m is bounded by the number of actions – defeating the purpose of the method.

From an empirical perspective, matters are not as clear-cut. Obviously, one can construct cases in which our computational advantage over Π^C, and our information advantage over Π^C_{ce} and FF^2, leads to exponential savings. IPC benchmarks are another matter. Evaluating all heuristic functions, we find that larger conjunction sets C do typically lead to smaller search spaces, and that h^{CFF} indeed is much faster than $h^{\text{FF}}(\Pi^C)$ for large C. Unfortunately, even the slowdown in h^{CFF} typically outweighs the search space reduction, and best overall performance is most often obtained with small C. On the positive side, our techniques can yield advantages even with small C, and in some IPC benchmarks large C is beneficial.

We next introduce our basic notation as well as the Π^C and Π^C_{ce} compilations in Section 2. We spell out our direct characterization of $h^+(\Pi^C)$ in Section 3, and we spell out our generalized relaxed plan extraction methods in Section 4. We summarize our implementation and experiments in Section 5, before concluding in Section 6. Most proofs are replaced in the main text by brief proof sketches. Full proofs are available in Appendix A.

2. Notations and Technical Background

We use the STRIPS framework. A planning *task* is a tuple $\Pi = (\mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G})$ where $\mathcal{F}$ is a set of *facts*, $\mathcal{A}$ a set of *actions*, $\mathcal{I} \subseteq \mathcal{F}$ is the *initial state*, and $\mathcal{G} \subseteq \mathcal{F}$ is the *goal*. Each action $a \in \mathcal{A}$ is a triple $(pre(a), add(a), del(a))$ of *precondition*, *add list*, and *delete list*, each a subset of $\mathcal{F}$. We henceforth tacitly assume a given input task $\Pi = (\mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G})$.

A *state* s is a subset of facts $s \subseteq \mathcal{F}$. Action a is applicable to s if $pre(a) \subseteq s$; in that case, applying a in s leads to the state $(s \cup add(a)) \setminus del(a)$. A *plan* for s is a sequence of iteratively applicable actions leading from s to a state that contains the goal $\mathcal{G}$. A plan for the task Π is a plan for the initial state $\mathcal{I}$. A plan is *optimal* if its length is minimal among all plans.

We assume throughout that $add(a) \cap del(a) = \emptyset$. This is a natural and common assumption – an action adding p does not also delete it – and is without loss of generality as any facts in the intersection can be equivalently removed from $add(a)$. The assumption is necessary for the "achieving each fact in separation" view of the delete relaxation, as outlined in the introduction.

Note that, for simplicity, we consider *unit costs*: all action costs are 1, and plan quality is just plan length. All our results straightforwardly extend to arbitrary non-negative action costs, and plan quality measured in terms of summed-up cost.

Example 1 *For illustration, we will frequently consider the following* car-driving example*. A car moves on a one-way line $X \to Y \to Z$ of locations, from X to Z. Each car move consumes a fuel unit, and the car's tank only holds one unit so we must refuel in Y.*

To encode this in STRIPS, we design the task $\Pi = (\mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G})$ as follows. $\mathcal{F} = \{carX, carY, carZ, fuel\}$, $\mathcal{I} = \{carX, fuel\}$, and $\mathcal{G} = \{carZ\}$. $\mathcal{A}$ consists of: a_{XY} with precondition $\{carX, fuel\}$, add list $\{carY\}$, and delete list $\{carX, fuel\}$; a_{YZ} with precondition $\{carY, fuel\}$, add list $\{carZ\}$, and delete list $\{carY, fuel\}$; and a_{refuel} with precondition $\{carY\}$, add list $\{fuel\}$, and empty delete list. The only plan for this task is $\langle a_{XY}, a_{refuel}, a_{YZ} \rangle$.

Given a planning task Π, we denote the set of all states by S. A *heuristic* (also *heuristic function*) is a function $h : S \mapsto \mathbb{N}_0^+ \cup \{\infty\}$ mapping states to natural numbers including 0, or to ∞ to indicate that the state is a dead-end. The *perfect heuristic* h^* maps any state s to the length of an optimal plan for s (or to ∞ if there is no plan for s). A heuristic h is *admissible* if $h(s) \leq h^*(s)$ for all $s \in S$. Abusing notation, we will often identify a heuristic h with its value $h(\mathcal{I})$ in the initial state. All statements made generalize to arbitrary states s by setting $I := s$. By $h(\Pi')$, we denote a heuristic for Π whose value is given by applying h in a modified task Π'. To make explicit that h is computed on Π itself, we write $h(\Pi)$.

We will characterize heuristic functions in terms of equations over regressed subgoals. The *regression of fact set G over action a*, $R(G, a)$, is defined if $add(a) \cap G \neq \emptyset$ and $del(a) \cap G = \emptyset$. In that case, $R(G, a) = (G \setminus add(a)) \cup pre(a)$; otherwise, we write $R(G, a) = \perp$.

The *critical-path* relaxation (Haslum & Geffner, 2000) assumes that, from any goal set of facts, it suffices to achieve the most costly subgoal of size at most m. Here, m is a parameter and the corresponding heuristic is denoted h^m. Precisely, h^m is defined as $h^m := h(\mathcal{G})$ where h is a function on fact sets G that satisfies

$$h(G) = \begin{cases} 0 & G \subseteq \mathcal{I} \\ 1 + \min_{a \in \mathcal{A}, R(G,a) \neq \perp} h(R(G, a)) & |G| \leq m \\ \max_{G' \subseteq G, |G'| \leq m} h(G') & \text{else} \end{cases} \tag{1}$$

It is easy to see that there is exactly one such h, as assuming two such functions h, h' where $h(G) \neq h'(G)$ recursively leads to a contradiction on the initial state.[1] The same argument applies to all h-defining equations considered herein, so henceforth we will assume uniqueness as given.

For $m = 1$, the definition of h^m becomes identical to that of the *max heuristic* $h^{\max}$ (Bonet & Geffner, 2001), which assumes that, to achieve a goal fact set, it is enough to achieve the maximum costly single fact. For $m = |\mathcal{F}|$, $h^m = h^*$ simply because subgoals of size $> |\mathcal{F}|$ do not exist. Computing h^m takes time exponential in m but polynomial in the size of Π.

The *delete relaxation* assumes that all delete lists are empty; a plan under this relaxation is a *relaxed plan*. The *ideal delete-relaxation heuristic* h^+ maps s to the length of an optimal relaxed plan for s. But optimal relaxed planning is **NP**-complete (Bylander, 1994). A *relaxed plan heuristic* maps s to the length of some, not necessarily optimal, relaxed plan for s, which can be computed easily (Hoffmann & Nebel, 2001). The resulting heuristic functions are not admissible, but are often very informative in practice for satisficing planning. We will follow the common approach of considering the idealized heuristic h^+ in theoretical examinations of the delete relaxation (compare, e. g., Hoffmann, 2005, 2011; Bonet & Helmert, 2010), and considering its effective approximation through relaxed plan heuristics in practice.

Relaxed plan heuristics differ in how they find the relaxed plan. A flexible way of specifying this are the *best-supporter functions* introduced by Keyder and Geffner (2008). A best-supporter function maps each fact p to the action the relaxed plan should use to support p. Given such a function, *relaxed plan extraction* starts at the goals, and keeps selecting best supporters, opening their preconditions as new subgoal facts, until initial state facts are reached. We will denote any heuristic arising from such a process by h^{FF} (disambiguating in context where needed). A detailed and formal characterization of relaxed plan extraction will be given in Section 4, where these details are technically relevant.

Practical best-supporter functions are based on $h^{\max} = h^1$, selecting for each p an action $a \in \mathcal{A}, R(G, a) \neq \perp$ minimizing the expression in the middle case of Equation 1 with $m = 1$ (where the subgoal G is a singleton set $\{p\}$ which can be identified with its element p). Alternatively, one can assign best supporters based on the *additive heuristic* h^{add} (Bonet & Geffner, 2001) instead, which differs from h^1 by using the sum, rather than the maximum, over the estimated cost of the facts in a goal set (bottom case in Equation 1). Note that (in both $h^{\max}$ and h^{add}) there may be several actions eligible as best supporter. Hence the construction of a best-supporter functions encompasses *tie-breaking*, in the sense of choosing an action from a set of actions supporting a given fact p. Such tie-breaking can have a large effect on the empirical performance of a relaxed plan heuristic. We will get back to this in detail in our experiments.

Throughout the paper, we will be concerned with *conjunctions* c. Following the STRIPS convention of formulating conjunctive conditions (action preconditions and the goal) as fact sets, a conjunction c here is a fact set $c \subseteq \mathcal{F}$, e. g. $c = \{p, q\}$. However, to improve readability, we will often notate c as a conjunctive formula instead, e. g. $c = p \wedge q$.

1. Matters are more complicated in case of 0-cost actions, where the recursion may lead into cycles and only the point-wise maximal h is unique.

We will henceforth assume a given set C of conjunctions. In practice, C will be computed once on the input task Π, prior to search.[2] We assume throughout that C contains all singleton conjunctions, $\{\{p\} \mid p \in \mathcal{F}\} \subseteq C$. This is just for convenience, notating facts as a special case of conjunctions. We will sometimes identify singleton conjunctions $\{p\}$ with the respective facts p, i. e., notate them without set brackets to avoid clutter; note that "p" also is the notation we get when writing $\{p\}$ as a conjunctive formula.

It will be convenient to introduce a shorthand for the operation of *collecting the atomic conjunctions contained in a fact set*. Given a set of facts $X \subseteq \mathcal{F}$, and assuming the given conjunction set C as described, we define $X^C := \{c \mid c \in C, c \subseteq X\}$. We will sometimes extend this notation to sets $\mathcal{X} = \{X_1, \dots, X_n\}$ of fact sets, where $\mathcal{X}^C$ is defined pointwise, i. e., $\mathcal{X}^C := \bigcup_i X_i^C$.

The Π^C compilation and its relatives are based on representing conjunctions explicitly, in terms of introducing new facts which are called *π-fluents*. They introduce one such fluent, π_c, for each $c \in C$. In correspondence to the shorthand just introduced, for a fact set $X \subseteq \mathcal{F}$, by $X^{\pi C} := \{\pi_c \mid c \in C, c \subseteq X\}$ we denote the set collecting the π-fluents for all atomic conjunctions entailed by X. We extend this notation to sets of fact sets in a pointwise manner as above.

Using these notations, Π^C can be defined as follows:

Definition 1 *Let $\Pi = (\mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G})$ be a planning task, and C a set of conjunctions in Π containing all singleton conjunctions. The* explicit-C compilation Π^C *is the planning task $(\mathcal{F}^{\pi C}, \mathcal{A}^{\pi C}, \mathcal{I}^{\pi C}, \mathcal{G}^{\pi C})$. Here, $\mathcal{F}^{\pi C}$, $\mathcal{I}^{\pi C}$, and $\mathcal{G}^{\pi C}$ are defined as per the shorthand. The set of actions $\mathcal{A}^{\pi C}$ contains an action $a[C']$, for every pair $a \in \mathcal{A}$ and $\emptyset \neq C' \subseteq \{c \in C \mid R(c, a) \neq \bot\}$, with $a[C']$ given by*

- $pre(a[C']) = [\bigcup_{c \in C'}(pre(a) \cup (c \setminus add(a)))]^{\pi C}$*, and*
- $add(a[C']) = \{\pi_c \mid c \in C'\}$*.*

The no cross-context explicit-C compilation Π^C_{nc} *is identical to Π^C except that $pre(a[C']) = \{pre(a) \cup (c \setminus add(a)) \mid c \in C'\}^{\pi C}$.*

We refer to pairs (a, C') of action a and set C' of supported conjunctions, corresponding to the compiled actions $a[C']$ in Π^C and Π^C_{nc}, as *action occurrences*. We refer to $c \setminus add(a)$, for $c \in C'$, as the *context* of c in a: for a to support c, the context must be true in the preceding state. This is captured by the preconditions in Π^C and Π^C_{nc}, which extend the original precondition with the contexts of the supported conjunctions. For Π^C, the context is collected across all supported conjunctions, for Π^C_{nc} this is done for each supported conjunction individually.

Our definition of Π^C diverges from the original definition by Haslum (2012) in several minor ways. First, we do not distinguish explicitly between the action's original effects vs. its supported conjunctions, instead expressing the add list of a as part of the set C' of conjunctions that may be supported. This is possible as C is assumed to contain all singleton conjunctions. As a consequence, we can demand that $C' \neq \emptyset$ (otherwise the action would have no effect and thus be useless). Second, we do not automatically include

2. The details of exactly how this is done are not relevant to our contribution. We will briefly describe the methods we use (adopted from Keyder et al., 2012, 2014) in the discussion of experiments, Section 5.

π_c facts relying on a context consisting only of non-deleted preconditions. Third, we do not demand C' to be "downward closed", i. e., to contain all subsumed conjunctions c', where there exists $c \in C'$ such that $c' \subseteq c$. Fourth, we do not include any delete effects. None of these changes have any consequences for the results we present. Changes two to four introduce some superfluous actions, simplifying our presentation while not affecting our results. The fourth change is suitable as we will use Π^C only for generating delete-relaxation heuristics. For consistent use of language, we will speak of "relaxed plans" for Π^C nevertheless.

We also modify the notation a bit, relative to Haslum (2012). We notate the actions as $a[C']$ instead of $A^{C'}$. This will be more convenient. We furthermore somewhat modified the definition of $a[C']$ preconditions, exploiting that $C' \neq \emptyset$. Namely, $pre(a[C']) = [\bigcup_{c \in C'}(pre(a) \cup (c \setminus add(a)))]^{\pi C}$ in Π^C is equivalent to the perhaps more intuitively straightforward definition, namely $pre(a[C']) = [pre(a) \cup \bigcup_{c \in C'}(c \setminus add(a))]^{\pi C}$ as used by Haslum. For Π^C_{nc}, the precondition $pre(a[C']) = \{pre(a) \cup (c \setminus add(a)) \mid c \in C'\}^{\pi C}$, given the point-wise interpretation of the "πC" superscript, equals $\bigcup_{c \in C'}[pre(a) \cup (c \setminus add(a))]^{\pi C}$, which itself is the same as the perhaps more intuitively straightforward definition $pre(a[C']) = pre(a)^{\pi C} \cup \bigcup_{c \in C'}[pre(a) \cup (c \setminus add(a))]^{\pi C}$. Observe that our modifications allow to write the action preconditions in terms of regression, thanks to $R(c,a) = pre(a) \cup (c \setminus add(a))$. In Π^C, the precondition then reads $pre(a[C']) = [\bigcup_{c \in C'} R(c,a)]^{\pi C}$. In Π^C_{nc}, it reads $pre(a[C']) = \{R(c,a) \mid c \in C'\}^{\pi C}$. These simplified notations will conveniently link-in with the concepts we introduce later on.

Note finally that the Π^C compilation introduces *all* atomic conjunctions into action preconditions and the goal, even ones subsumed by other, larger, atomic conjunctions contained in the same precondition/goal. We stick to this convention throughout, for simplicity. In practice, we ignore the subsumed conjunctions. In the remainder of the paper, this corresponds to a modified "C" superscript, only including conjunctions $c \in C$, $c \subseteq X$, where there does not exist $c' \in C$, $c' \subseteq X$, so that $c \subsetneq c'$; correspondingly for the "πC" superscript. This leaves all results intact exactly as stated.

Example 2 *Reconsider our car-driving example task Π from Example 1. As the delete relaxation ignores the negative effect of a_{XY}, a shortest relaxed plan is $\langle a_{XY}, a_{YZ} \rangle$ and $h^+ = 2$.*

However, say we set C to contain (all singleton conjunctions as well as) $c = carY \wedge fuel$. Then, in Π^C, π_c is a precondition of all actions $a_{YZ}[C']$, i. e., of all actions adding the goal carZ. The only actions adding π_c have the form $a_{refuel}[C']$ where $c \in C'$. Hence $\langle a_{XY}, a_{YZ} \rangle$ is not *a relaxed plan for Π^C. Instead, we need to perform a refueling action, for example in the relaxed plan $\langle a_{XY}[\{carY\}], a_{refuel}[\{carY \wedge fuel\}], a_{YZ}[\{carZ\}] \rangle$. We get $h^+(\Pi^C) = 3 = h^*(\Pi)$.*

The growth of Π^C and Π^C_{nc} is exponential in $|C|$ because action occurrences enumerate subsets $C' \subseteq C$ of supported conjunctions. This complexity is necessary because, otherwise, $h^+(\Pi^C)$ and $h^+(\Pi^C_{nc})$ would not be admissible. As a simple example, say the goal in Π_n is $\{g_1, \dots, g_n\}$, C contains the singleton conjunctions as well as all fact pairs, and there is a single action a achieving all of $\{g_1, \dots, g_n\}$. Then $h^*(\Pi_n) = 1$, and $h^+(\Pi^C_n) = 1$ thanks to the optimal plan $\langle a[C'] \rangle$ where C' is the set of all conjunctions, $C' = C = \{\{g_i\} \mid 1 \leq i \leq n\} \cup \{\{g_i, g_j\} \mid 1 \leq i \neq j \leq n\}$. However, if we had to achieve every conjunction separately in Π^C_n, that is, if we included into $\mathcal{A}^{\pi C}$ only actions of the form $a[\{c\}]$ for $c \in C$, then we would get $h^+(\Pi^C_n) = n + \frac{n*(n-1)}{2}$ because we would have to achieve every conjunction $c \in C$ with a separate compiled action. (This observation will become relevant again later on, compare Example 7 in Section 4.3.1.)

The difference between Π^C and Π^C_{nc} is that the latter, but not the former, ignores what has been termed *cross-context conditions*: conjunction preconditions of $a[C']$ in Π^C which arise only from the combination of several $c \in C'$. Precisely, a cross-context condition for a and C' is a conjunction $c \in C$ where $c \subseteq \bigcup_{c \in C'}[pre(a) \cup (c \setminus add(a))]$, but there does not exist any single $c \in C'$ such that $c \subseteq pre(a) \cup (c \setminus add(a))$. In the Π^C_{nc} compilation, the precondition of $a[C']$ does not contain any cross-context conditions, because the superscript "πC" in $pre(a[C']) = \{pre(a) \cup (c \setminus add(a)) \mid c \in C'\}^{\pi C}$, i. e., the collection of conjunctions, is done for the context of each $c \in C'$ separately. This is in contrast to Π^C where, in $pre(a[C']) = [\bigcup_{c \in C'}(pre(a) \cup (c \setminus add(a)))]^{\pi C}$, the conjunctions are collected from the *union of contexts* across $c \in C'$.

Example 3 *To illustrate cross-context conditions, we will consider the following* abstract example *(given by Keyder et al., 2014, as part of the proof of their Theorem 3).* $\Pi = (\mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G})$ *where* $\mathcal{F} = \{g_1, g_2, p, q_1, q_2\}$, $\mathcal{I} = \{q_1\}$, $\mathcal{G} = \{g_1, g_2\}$ *and* $\mathcal{A}$ *consists of:* a_{g1} *with precondition* $\{p, q_1\}$, *add list* g_1, *and empty delete list;* a_{g2} *with precondition* $\{p, q_2\}$, *add list* g_2, *and empty delete list;* a_p *with empty precondition, add list* $\{p\}$, *and empty delete list;* a_{q2} *with precondition* $\{q_1\}$, *add list* $\{q_2\}$, *and delete list* $\{q_1, p\}$. *In this construction,* q_1 *and* q_2 *are mutex; achieving* g_1 *requires* q_1 *and thus has to be done first, via* a_p *and* a_{g1}. *To achieve* g_2, *we require* q_2. *Getting* q_2 *through* a_{q2} *deletes* p, *so that we must apply* a_p *a second time before applying* a_{g2}. *As in the delete relaxation there never is a need to apply the same action twice,* $h^+ = 4 < 5 = h^*$.

Say we set C *to contain* $c_{q1p} = q_1 \wedge p$, $c_{q2p} = q_2 \wedge p$, *and* $c_{q1q2} = q_1 \wedge q_2$. *Then any relaxed plan for* Π^C *must contain two occurrences of* a_p: c_{q1p} *and* c_{q2p} *are required to achieve the goal;* a_p *is the only action that can support these conjunctions (note here that* a_{q2} *deletes* p*); and* $a_p[\{c_{q1p}, c_{q2p}\}]$ *supporting both conjunctions with a single action occurrence has the unreachable cross-context condition* c_{q1q2}. *Consequently,* $h^+(\Pi^C) = 5 = h^*(\Pi)$.

In contrast, in Π^C_{nc}, $a_p[\{c_{q1p}, c_{q2p}\}]$ *does not have the cross-context condition, so* $\langle a_{q2}[\{q_2\}], a_p[\{p, c_{q1p}, c_{q2p}\}], a_{g1}[\{g_1\}], a_{g2}[\{g_2\}]\rangle$ *is a relaxed plan and* $h^+(\Pi^C_{nc}) = 4 = h^+(\Pi) < h^*(\Pi)$.

Keyder et al. (2012, 2014) introduced the Π^C_{ce} compilation, which achieves the same effect as Π^C_{nc} but has size polynomial in $|C|$. This is done by augmenting every original action $a \in \mathcal{A}$ with one conditional effect for each $c \in C$ that can be regressed over a, adding π_c and requiring the context $c \setminus add(a)$ as the effect condition.

The Π^C_{ce} compilation is equivalent to Π^C_{nc} in the sense that $h^+(\Pi^C_{ce}) = h^+(\Pi^C_{nc})$. Intuitively, any occurrence of an action in Π^C_{ce}, where the conditional effects for the set of conjunctions C' fire, is equivalent to the Π^C_{nc} action $a[C']$ because in Π^C_{ce} the conjunctions $c \in C'$ are handled separately, ignoring cross-context conditions.[3] Given this equivalence, in our theoretical discussion of heuristic functions and their properties – where the size difference between Π^C_{nc} and Π^C_{ce} does not matter – we will refer throughout to Π^C_{nc} rather than Π^C_{ce}. This simplifies matters because we do not have to switch between formalisms (STRIPS with vs. without conditional effects).

3. Technically, $h^+(\Pi^C_{ce}) \geq h^+(\Pi^C_{nc})$ was proved by Keyder et al. (2014) in the proof to their Lemma 2, and the opposite direction $h^+(\Pi^C_{ce}) \leq h^+(\Pi^C_{nc})$ is symmetric.

We will see in Section 4.3 that the complexity reduction from Π^C to Π^C_{ce} has its correspondence in a complexity reduction of the subgoal-support selection problem (maximization of $|C'|$ during relaxed plan extraction): while that problem is **NP**-complete for Π^C, it is polynomial-time for Π^C_{nc}.

3. Combining the Delete Relaxation with Critical Paths: h^{C+}

We now spell out observation (1) from the introduction, characterizing the combination of the delete relaxation with critical paths directly, without a compilation, in terms of a heuristic function we call h^{C+}. Section 3.1 starts with simple novel views on each of the two components, and Section 3.2 combines these into an equation characterizing h^{C+}. Section 3.3 sketches our proof of correctness i. e., that $h^{C+} = h^+(\Pi^C)$. Section 3.4 summarizes the properties of h^{C+}, pointing out that the combination of the delete relaxation with critical paths naturally generalizes its components.

3.1 Novel Views on h^m and h^+

First, consider the following straightforward characterization of h^*, which will be relaxed in different manners below: $h^* := h(\mathcal{G})$ where h is the function on fact sets G that satisfies

$$h(G) = \begin{cases} 0 & G \subseteq \mathcal{I} \\ 1 + \min_{a \in \mathcal{A}, R(G,a) \neq \perp} h(R(G,a)) & \text{else} \end{cases} \tag{2}$$

This equation obviously characterizes optimal planning, and therewith h^*: we minimize plan length over all actions that can support our subgoal G.

Slightly rephrasing the critical-path relaxation, it assumes that, to achieve a subgoal G, it suffices to achieve the most costly *atomic subgoal*, where the notion of "atomic subgoal" is a parameter. In the traditional formulation, that parameter is instantiated with "all fact sets of size at most m". But there is no need to be so restrictive. The atomic subgoals can be an arbitrary set of fact-sets, in other words: an arbitrary set C of conjunctions. We merely need to replace the subgoal-selection mechanisms in h^m (Equation 1) with accordingly generalized ones. We denote the resulting heuristic by h^C, defined as $h^C := h(\mathcal{G})$ where h is the function on fact sets G that satisfies

$$h(G) = \begin{cases} 0 & G \subseteq \mathcal{I} \\ 1 + \min_{a \in \mathcal{A}, R(G,a) \neq \perp} h(R(G,a)) & G \in C \\ \max_{G' \subseteq G, G' \in C} h(G') & \text{else} \end{cases} \tag{3}$$

Trivially, $h^C = h^m$ if C consists of all conjunctions of size $\leq m$. As we shall see below, $h^C = h^1(\Pi^C)$ as one would expect. The latter property is useful only from a theoretical perspective though, connecting h^C to known results about h^1. In practice, h^C can be computed like h^m, by a fixed point process on value assignments to the atomic subgoals C, taking time polynomial in $|C|$, in contrast to the size of Π^C. Our particular implementation will be described in Section 5.1.

It is worth pointing out that the simple generalization from h^m to h^C already can be quite useful:

Example 4 *Consider our car-driving example from Example 1, but without the refuel action. This modified task is unsolvable, and $h^2 = \infty$ recognizes that. Yet, there is no need to reason about* all *fact pairs to arrive at this conclusion: Considering the* single *fact pair $C = \{c\}$ where $c = carY \wedge fuel$, like in Example 2, suffices to get $h^C = \infty$, as c becomes a precondition for achieving the goal, and there is no action over which c can be regressed. While this particular example only has* 4 *facts and thus* 6 *fact pairs, we could scale it arbitrarily by adding solvable parts, blowing up the computational overhead of h^2 while still recognizing unsolvability using the single fact pair c.*

Getting back to our discussion of alternate ways to relax h^*, i. e., Equation 2, observe that Equation 3 uses the correct regression semantics, but relaxes the subgoals considered. The delete relaxation can be viewed as approaching this vice-versa, keeping the correct subgoaling but relaxing the regression semantics. This is immediately visible in the following straightforward characterization of h^+, as $h^+ := h(\mathcal{G})$ where h is the function on fact sets G that satisfies

$$h(G) = \begin{cases} 0 & G \subseteq \mathcal{I} \\ 1 + \min_{a \in \mathcal{A}, \emptyset \neq G \cap add(a)} h((G \setminus add(a)) \cup pre(a)) & \text{else} \end{cases} \quad (4)$$

This is identical to Equation 2 except for pretending that $R(G, a) \neq \bot$ even if $del(a) \cap G \neq \emptyset$, i. e., replacing $R(G, a)$ with the relaxed concept that only asks for non-empty add-list intersection.

The basic observation towards combining h^C with h^+ is that the underlying relaxation principles, though they seem unrelated given Equations 3 and 4, can *both* be viewed as relaxations pertaining to the subgoaling structure. This becomes visible in the following alternative characterization of h^+:

Lemma 1 *Let $\Pi = (\mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G})$ be a planning task, and let h be the function on fact sets G that satisfies*

$$h(G) = \begin{cases} 0 & G \subseteq \mathcal{I} \\ 1 + \min_{a \in \mathcal{A}, \emptyset \neq G' = \{p \mid p \in G, R(\{p\}, a) \neq \bot\}} h((G \setminus G') \cup \bigcup_{p \in G'} R(\{p\}, a)) & \textit{else} \end{cases} \quad (5)$$

Then $h(\mathcal{G}) = h^+$.

Proof: Observe that, for singleton fact sets $G = \{p\}$, (a) regressability of G over a trivializes to add-list intersection, i. e., $R(\{p\}, a) \neq \bot$ iff $p \in add(a)$, because with $add(a) \cap del(a) = \emptyset$ we get $p \notin del(a)$; and (b) if G can be regressed over a then the regression simply generates the action precondition as the new subgoal, i. e., $R(\{p\}, a) = pre(a)$. So Equation 5 simplifies to

$$h(G) = \begin{cases} 0 & G \subseteq \mathcal{I} \\ 1 + \min_{a \in \mathcal{A}, \emptyset \neq G' = G \cap add(a)} h((G \setminus G') \cup pre(a)) & \text{else} \end{cases}$$

With $G' = G \cap add(a)$ we have $G \setminus G' = G \setminus add(a)$, so this is equivalent to Equation 4. ■

As per Equation 5, the delete relaxation can be understood as *splitting subgoals up into singleton facts, and considering regression separately with respect to each of these*. As singleton regression trivializes, in effect we need to worry only about the part of the subgoal we can support, not about other parts that the same action may contradict.[4] While this reformulation is awkward and not useful in the standard setting, it exhibits a possible refinement to that setting: instead of singleton facts, consider atomic subgoals in the form of an arbitrary set C of conjunctions.

3.2 The Combined Heuristic

Consider again Equation 5, and compare it with the following equation characterizing h^1:

$$h(G) = \begin{cases} 0 & G \subseteq \mathcal{I} \\ 1 + \min_{a \in \mathcal{A}, R(\{p\},a) \neq \bot} h(R(\{p\}, a)) & G = \{p\} \\ \max_{p \in G} h(\{p\}) & \text{else} \end{cases} \tag{6}$$

Equation 5 can be understood as a less relaxed version of Equation 6. Both decompose a subgoal G into its atomic subgoals, instantiated as singleton facts, and both minimize over actions regressing atomic subgoals. The difference is that, while Equation 6 picks the single most costly atomic subgoal, Equation 5 requires to achieve *every* atomic subgoal (in particular, including the ones not supported by a, i. e., $G \setminus G'$, in the recursive invocation of h). As the set G consists exactly of its atomic subgoals, Equation 5 does not need a third case identifying the atomic subgoals.

Now, h^C generalizes h^1 in considering the more general atomic subgoals C. Applying a similar generalization to Equation 5, we obtain our desired combination of the delete relaxation with critical paths:

Definition 2 *Let $\Pi = (\mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G})$ be a planning task, and C a set of conjunctions in Π containing all singleton conjunctions. The* critical-path delete relaxation heuristic, *short* C-relaxation heuristic, *is defined as $h^{C+} := h(\mathcal{G}^C)$, where h is the function on conjunction sets G that satisfies*

$$h(G) = \begin{cases} 0 & \forall c \in G : c \subseteq \mathcal{I} \\ 1 + \min_{a \in \mathcal{A}, \emptyset \neq G' \subseteq \{c \mid c \in G, R(c,a) \neq \bot\}} h((G \setminus G') \cup G_r'^C) & \textit{else} \end{cases} \tag{7}$$

with G'_r defined as $G'_r := \bigcup_{c \in G'} R(c, a)$.

The no cross-context critical-path delete relaxation heuristic, *short* nc-C-relaxation heuristic, *denoted h^{C+}_{nc}, is defined identically to h^{C+} except that we define $G'_r := \{R(c, a) \mid c \in G'\}$.*

Recall here that, for a fact set X, $X^C := \{c \mid c \in C, c \subseteq X\}$ denotes the set of atomic conjunctions contained in X, and that for a set of fact sets (as in the case of G'_r for h^{C+}_{nc}) we apply this notation pointwise. As a convention, we will refer to the expression "$(G \setminus G') \cup G_r'^C$" in Equation 7, and in related equations, as the *recursive subgoal*, and to G'_r as the *regressed subgoal*.

Intuitively, h^{C+} supports atomic subgoals from C individually by regression as in h^C, but instead of achieving only the most costly one, it achieves all of them. This parallels our previous comparison between h^+ and h^1. The subgoals G recursed over now are *sets of conjunctions*, because in difference to h^+ (Equation 5) atomic subgoals are conjunctions

4. The independence assumptions identified by Keyder and Geffner (2009) are somewhat related to this. But our formulation pertains to the delete relaxation heuristic h^+ itself, ignoring negative side effects; whereas Keyder and Geffner's observations pertain to simplifying assumptions in *approximations* of h^+, ignoring *positive* side effects.

instead of single facts, and in difference to h^C (Equation 3) we estimate the cost of sets of atomic subgoals instead of single atomic subgoals. The initializing call on $\mathcal{G}^C$ inserts all atomic conjunctions from the global goal, and the recursive subgoals insert all atomic conjunctions from the regressed subgoal G'_r. Hence, like in Equation 5 the set G consists exactly of its atomic subgoals, and we do not need a third case identifying the atomic subgoals.

The top case in Equation 7 is self-explanatory. The bottom case generalizes that in Equation 5. Because atomic subgoals now are non-unit conjunctions, in difference to our arguments in Lemma 1, regression no longer trivializes: a is not allowed to contradict any conjunction $c \in G'$, and $R(c,a)$ may be a proper superset of $pre(a)$, no longer trivializing to the action precondition. Hence, in difference to Equation 5, the more complex notation is now necessary. There also is a major new source of complexity, relative to Equation 5, namely the need to allow G' to be a *subset* of the supportable atomic subgoals, rather than just setting G' to that entire set. This corresponds to the aforementioned *subgoal-support selection problem*. We illustrate that problem in Example 5 below; Section 4.3 conducts an in-depth analysis in the context of relaxed plan extraction.

The difference between h^{C+} and h^{C+}_{nc} is, as the notation suggests, designed to match the difference between Π^C and Π^C_{nc}. The expressions "$\bigcup_{c \in G'} R(c,a)$" vs. "$\{R(c,a) \mid c \in G'\}$" are in obvious correspondence with the action preconditions in Definition 1 (thanks to our modifications with respect to the original definition). A pair (a, G') of action and subset of supported atomic subgoals in the h^{C+} equation corresponds to the Π^C action $a[C']$ where $C' = G'$, similarly for h^{C+}_{nc} and Π^C_{nc}. An instructive alternative way to read the regressed subgoals G'_r is in terms of conjunctions. This gives $\bigcup_{c \in G'} R(c,a) = \bigwedge_{c \in G'} R(c,a) = \bigwedge_{c \in G', p \in R(c,a)} p$, vs. $\{R(c,a) \mid c \in G'\} = \{\bigwedge_{p \in R(c,a)} p \mid c \in G'\}$: one large conjunction vs. several small ones. This makes a difference because larger conjunctions may contain larger atomic subgoals, as captured in Definition 2 through the respective use of ${G'_r}^C$.

Example 5 *Consider, as in Example 2 (page 276), our car-driving example with C containing the singleton conjunctions as well as* $c = carY \wedge fuel$. *We get* $h^{C+} = h(\{carZ\})$, *i.e., h defined as per Equation 7, applied to the conjunction set containing the single goal atomic conjunction carZ. The only* (a, G') *pair supporting carZ is* $(a_{YZ}, \{carZ\})$. *Selecting* (a, G'), *we get the recursive subgoal* $G = \{carY, fuel, carY \wedge fuel\}$. *As* $carY \wedge fuel$ *cannot be supported by* a_{XY} *which deletes fuel, the only supporting action for that subgoal is* a_{refuel}. *Say we select that action, and* $G' := \{carY \wedge fuel\}$. *The recursive subgoal then is* $\{carY, fuel\}$ *because the conjunctions carY and fuel in G are not included in* G'. *In detail, the recursive subgoal results from the expression* $(G \setminus G') \cup [\bigcup_{c \in G'} R(c,a)]^C = (\{carY, fuel, carY \wedge fuel\} \setminus \{carY \wedge fuel\}) \cup R(carY \wedge fuel, a_{refuel})^C = \{carY, fuel\} \cup \{carY\}^C = \{carY, fuel\} \cup \{carY\}$. *That subgoal can be resolved using* $(a_{XY}, \{carY\})$, *yielding* $h^{C+} = h^+(\Pi^C) = 3$ *due to the same relaxed plan as in Example 2.*

Now consider, as in Example 3 (page 277), our abstract example with C containing the singleton conjunctions as well as $c_{q1p} = q_1 \wedge p$, $c_{q2p} = q_2 \wedge p$, *and* $c_{q1q2} = q_1 \wedge q_2$. *We have* $h^{C+} = h(\{g_1, g_2\})$, *requiring to support each of the two goal facts (written as singleton conjunctive formulas here). This can be done only by* $(a_{g1}, \{g_1\})$ *and* $(a_{g2}, \{g_2\})$ *respectively; after using these, we get the recursive subgoal* $G = \{q_1, q_2, p, q_1 \wedge p, q_2 \wedge p\}$. *Now we have a non-trivial subgoal-support selection problem. Ignoring the subsumed subgoals* q_1, q_2, p *which can be tackled as a side effect of tackling the non-subsumed ones* $q_1 \wedge p$ *and* $q_2 \wedge p$, *we can choose any of (a)*

$(a_p, \{q_1 \wedge p, q_2 \wedge p\})$, (b) $(a_p, \{q_1 \wedge p\})$, or (c) $(a_p, \{q_2 \wedge p\})$. If we choose (a), then our subgoal is fully supported i. e., $G \setminus G' = \emptyset$, yet $[\bigcup_{c \in G'} R(c, a_p)]^C = \{q_1, q_2\}^C = \{q_1, q_2, q_1 \wedge q_2\}$. The cross-context conjunction $q_1 \wedge q_2$ is not supported by any action, so we cannot get to the initial state this way. Instead, we need to take either (b) or (c), yielding the recursive subgoals (b) $\{q_2 \wedge p, q_1\}$ respectively (c) $\{q_1 \wedge p, q_2\}$, each of which necessitates support by a_{q2} as well as another occurrence of a_p, leading to $h^{C+} = h^+(\Pi^C) = h^* = 5$.

Using h^{C+}_{nc} instead, option (a) produces the different subgoal $\{R(c, a) \mid c \in G'\}^C = \{\{q_1\}, \{q_2\}\}^C = \{q_1, q_2\}$, not containing the cross-context conjunction $q_1 \wedge q_2$. This subgoal is feasible, and only requires support by a_{q2}, leading to $h^{C+}_{nc} = h^+(\Pi^C_{nc}) = h^+ = 4$.

3.3 Proof of Correctness

We prove that Equation 7 does indeed capture $h^+(\Pi^C)$, i. e., that $h^+(\Pi^C) = h^{C+}(\Pi)$. For illustration, we first consider the simple case where C contains only the singleton conjunctions:

Proposition 1 *Let $\Pi = (\mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G})$ be a planning task, and $C = \{\{p\} \mid p \in \mathcal{F}\}$. Then $h^+ = h^{C+}$.*

Proof: With $C = \{\{p\} \mid p \in F\}$, the recursive subgoals G in Equation 7 are sets of singleton fact-sets, so we can instead perceive G as a set of facts. We can then re-write Equation 7 to:

$$h(G) = \begin{cases} 0 & G \subseteq \mathcal{I} \\ 1 + \min_{a \in \mathcal{A}, \emptyset \neq G' \subseteq \{p \in G \mid R(\{p\}, a) \neq \bot\}} h((G \setminus G') \cup \bigcup_{p \in G'} R(\{p\}, a)) & \text{else} \end{cases}$$

This is identical to Equation 5 except that G' is allowed to be a subset of $\{p \in G \mid R(\{p\}, a) \neq \bot\}$. However, because $R(\{p\}, a) = pre(a)$, the minimum in the bottom case can always be achieved using $G' = \{p \in G \mid R(\{p\}, a) \neq \bot\}$, which can only yield smaller recursive subgoals than $G' \subset \{p \in G \mid R(\{p\}, a) \neq \bot\}$. This concludes the proof with Lemma 1. ■

Observe that Proposition 1 proves that $h^+(\Pi^C) = h^{C+}(\Pi)$ for $C = \{\{p\} \mid p \in F\}$: with singleton conjunctions only, $h^+ = h^+(\Pi^C)$, so by Proposition 1 we have $h^+(\Pi^C) = h^+ = h^{C+}(\Pi)$ as desired. We now extend this to the general case, for arbitrary conjunction sets:

Theorem 1 *Let $\Pi = (\mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G})$ be a planning task, and C a set of conjunctions in Π containing all singleton conjunctions. Then $h^+(\Pi^C) = h^{C+}(\Pi)$.*

Proof Sketch: We apply Equation 5 to Π^C, characterizing $h^+(\Pi^C)$. Making explicit that the individual facts in Π^C all are π-fluents, we obtain: $h^+(\Pi^C) = h(\{\pi_c \mid \pi_c \in \mathcal{G}^{\pi C}\})$, where h is the function on fact sets G that satisfies $h(G) =$

$$\begin{cases} 0 & \forall \pi_c \in G : \pi_c \in \mathcal{I}^{\pi C} \\ 1 + \min_{a[C'] \in \mathcal{A}^{\pi C}, \emptyset \neq G' = \{\pi_c \mid \pi_c \in G, R(\{\pi_c\}, a[C']) \neq \bot\}} h((G \setminus G') \cup G'_r) & \text{else} \end{cases}$$

with G'_r defined as $G'_r := \bigcup_{\pi_c \in G'} R(\{\pi_c\}, a[C'])$.

The condition $R(\{\pi_c\}, a[C']) \neq \bot$ here simplifies to $c \in C'$, because these are exactly the π-fluents added by $a[C']$. So we have $G' = \{\pi_c \mid \pi_c \in G, c \in C'\}$ and the minimization is over those $a[C']$ supporting a non-empty subset of subgoals π_c. The c we can in principle include into C' are, by the definition of Π^C, exactly those where $R(c,a) \neq \bot$. There is no point in including c where $\pi_c \notin G$, as this will support the same subgoals yet can only result in a larger precondition. Hence, renaming C' into G' in order to unify notation, we obtain $h(G) =$

$$\begin{cases} 0 & \forall \pi_c \in G : \pi_c \in \mathcal{I}^{\pi C} \\ 1 + \min_{a[G'] \in \mathcal{A}^{\pi C}, \emptyset \neq G' \subseteq \{c \mid \pi_c \in G, R(c,a) \neq \bot\}} h((G \setminus G') \cup G'_r) & \text{else} \end{cases}$$

with G'_r defined as $G'_r := \bigcup_{c \in G'} R(\{\pi_c\}, a[G'])$.

Comparing this equation with Equation 7, it is easy to see that the equations are in exact correspondence via $(*)$ $G = \{\pi_c \mid c \in G[7]\}$, where $G[7]$ denotes the subgoal sets in Equation 7. (*) is true by definition for the initializing calls, $h^{C+}(\Pi) = h(\mathcal{G}^C)$ respectively $h^+(\Pi^C) = h(\{\pi_c \mid \pi_c \in \mathcal{G}^{\pi C}\})$. (*) is invariant over the bottom cases in both equations, as $G'_r = \bigcup_{c \in G'} R(\{\pi_c\}, a[G']) = pre(a[G']) = [\bigcup_{c \in G'}(pre(a) \cup (c \setminus add(a)))]^{\pi C} = [\bigcup_{c \in G'} R(c,a)]^{\pi C}$, which matches the regressed subgoal $G'_r[7] = [\bigcup_{c \in G'} R(c,a)]^C$ of Equation 7 as desired. ■

A similar proof shows the same correspondence for Π^C_{nc} and h^{C+}_{nc}:

Theorem 2 *Let $\Pi = (\mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G})$ be a planning task, and C a set of conjunctions in Π containing all singleton conjunctions. Then $h^+(\Pi^C_{nc}) = h^{C+}_{nc}(\Pi)$.*

3.4 Properties of the Combination

The combination of the delete relaxation with critical paths, as per Definition 2, naturally generalizes the properties of its components. This follows from known results along with the following simple observation:[5]

Theorem 3 *Let $\Pi = (\mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G})$ be a planning task, and C a set of conjunctions in Π containing all singleton conjunctions. Then $h^1(\Pi^C) = h^1(\Pi^C_{nc}) = h^C(\Pi)$.*

Proof Sketch: Consider first Π^C. Applying Equation 6 (page 6) characterizing h^1 to Π^C, we get $h^1(\Pi^C) = h(\mathcal{G}^{\pi C})$ where h is the function on fact sets G that satisfies

$$h(G) = \begin{cases} 0 & G \subseteq \mathcal{I}^{\pi C} \\ 1 + \min_{a[C'] \in \mathcal{A}^{\pi C}, R(G, a[C']) \neq \bot} h(R(G, a[C'])) & G = \{\pi_c\}, c \in C \\ \max_{\pi_c \in G} h(\{\pi_c\}) & \text{else} \end{cases}$$

Observe that, in the middle case, we must have $c \in C'$ because otherwise $\pi_c \notin add(a[C'])$; and that there is no point in including any other conjunctions into C', i. e., $C' \supsetneq \{c\}$, because this can only yield a larger recursive subgoal $R(G, a[C'])$. Hence we can re-write the previous equation to:

$$h(G) = \begin{cases} 0 & G \subseteq \mathcal{I}^{\pi C} \\ 1 + \min_{a[\{c\}] \in \mathcal{A}^{\pi C}, R(G, a[\{c\}]) \neq \bot} h(R(G, a[\{c\}])) & G = \{\pi_c\}, c \in C \\ \max_{\pi_c \in G} h(\{\pi_c\}) & \text{else} \end{cases}$$

5. Keyder et al. (2014) already proved the $h^1(\Pi^C) \leq h^1(\Pi^C_{nc})$ part of this observation, using a different proof argument.

Comparing this equation with Equation 3 (page 278) characterizing h^C, it is easy to see that the equations are in exact correspondence via $(*)$ $G = \{\pi_c \mid c \in C, c \subseteq G[3]\}$, where $G[3]$ denotes the subgoal (fact) sets in Equation 3.

The argument for Π^C_{nc} is identical because, for single-conjunction sets $C' = \{c\}$, the two compilations coincide. ∎

Note that, in the step from the first to the second equation stated in this proof, the exponential size of Π^C is reduced to the polynomial-size compilation which is like Π^C but includes only the actions $a[\{c\}]$ for pairs $a \in A$ and $c \in C$ where c can be regressed through a. Intuitively, as h^1 only considers singleton subgoals, there is no need to enumerate supported conjunction sets of size greater than 1. This reduced compilation is essentially a version of Haslum's (2009) Π^m compilation for arbitrary conjunction sets C. (This simple generalization was not mentioned by Haslum in his works on Π^m and Π^C.)

Together with results by Haslum (2012) and Keyder et al. (2014), as well as basic known results about h^1 and h^+, Theorems 1 – 3 immediately imply all the properties one would naturally expect h^{C+} and h^{C+}_{nc} to have:

Corollary 1 *Let Π be a planning task. Then, for any set C of conjunctions in Π containing all singleton conjunctions, we have:*

(i) $h^C, h^+ \leq h^{C+}_{nc} \leq h^{C+} \leq h^*$; and

(ii) $h^{C+} = \infty$ iff $h^{C+}_{nc} = \infty$ iff $h^C = \infty$.

Furthermore, both h^{C+} and h^{C+}_{nc} converge to h^*, i. e., there exist sets C of conjunctions such that (iii) $h^{C+} = h^*$ respectively (iv) $h^{C+}_{nc} = h^*$.

Proof: Regarding (i): By Theorem 2, $h^{C+}_{nc} = h^+(\Pi^C_{nc})$ and by Theorem 3 $h^C = h^1(\Pi^C_{nc})$, so $h^C \leq h^{C+}_{nc}$ follows from $h^1 \leq h^+$. As $h^+(\Pi^C_{nc}) = h^+(\Pi^C_{ce})$ and hence $h^{C+}_{nc} = h^+(\Pi^C_{ce})$, $h^+ \leq h^{C+}_{nc}$ holds by the corresponding result of Keyder et al. (2014) ($h^+ \leq h^+(\Pi^C_{ce})$, their Corollary 1). As Π^C_{nc} drops preconditions from Π^C, we get $h^{C+}_{nc} = h^+(\Pi^C_{nc}) \leq h^+(\Pi^C)$, where $h^+(\Pi^C) = h^{C+}$ by Theorem 1. Finally, $h^{C+} \leq h^*$ holds by the corresponding result of Haslum (2012) ($h^+(\Pi^C) \leq h^*$, his Theorem 4).

Regarding (ii): As $h^{C+} = h^+(\Pi^C)$, $h^{C+}_{nc} = h^+(\Pi^C_{nc})$, and $h^C = h^1(\Pi^C) = h^1(\Pi^C_{nc})$, this follows from $h^+ = \infty$ iff $h^1 = \infty$.

Finally, (iii) holds by convergence of $h^+(\Pi^C)$ (Haslum, 2012, Theorem 5) because $h^{C+} = h^+(\Pi^C)$ as per Theorem 1, and (iv) holds by convergence of $h^+(\Pi^C_{ce})$ (Keyder et al., 2014, Theorem 5) because $h^+(\Pi^C_{nc}) = h^+(\Pi^C_{ce})$ and $h^{C+}_{nc} = h^+(\Pi^C_{nc})$ as per Theorem 2. ∎

4. Extracting Relaxed Plans: $h^{\mathbf{CFF}}$

We just observed that, like in the standard setting, $h^{C+} = \infty$ iff $h^C = \infty$, i. e., a relaxed plan exists iff the critical-path component of our heuristic is solvable. So h^C behaves like h^1 in the role of deciding relaxed plan existence. But then, can h^C also fulfill the role of h^1 in *relaxed plan extraction*, i. e., finding some not necessarily optimal relaxed plan?

Implicitly, this is already being done in the Π^C compilation, via relaxed plan extraction on $h^1(\Pi^C) = h^C$, but this construction is wasteful as computing h^C does not actually require the exponential blow-up inherent in Π^C. Can we make do without this blow-up?

As we indicated in the introduction (in observation (2)), the answer is "yes", in the form of a heuristic function we denote by h^{CFF}: *Relaxed plan extraction to obtain h^{FF} from a h^1 best-supporter function translates into relaxed plan extraction to obtain h^{CFF} from a h^C best-supporter function.* Thanks to this direct formulation, not using a compilation, h^{CFF} computes relaxed plans for Π^C in time polynomial in $|C|$.

To spell this out in detail, we start in Section 4.1 by, similarly as before, reformulating standard relaxed plan extraction in a way preparing the generalization to arbitrary conjunction sets C. Section 4.2 specifies that generalization and proves it correct. Section 4.3 analyzes the subgoal-support selection problem, which is benign in the standard setting but is **NP**-complete in the general case; we define our heuristic function h^{CFF} using a greedy solution to that problem.

Throughout, we use equation-based formulations as these generalize directly to arbitrary action costs. To improve readability, we also include pseudo-code formulations which apply only to the simpler unit-cost case. Like before, we distinguish between variants taking cross-context conditions into account (Π^C) vs. not doing so (Π^C_{nc}).

4.1 Relaxed Plan Extraction from h^1

Relaxed plan extraction was first formulated in terms of best-supporter functions by Keyder and Geffner (2008). The advantage over more traditional relaxed planning graph formulations (Hoffmann & Nebel, 2001) is that best-supporter functions are more flexible, allowing to use h^{add} instead of h^1, and generalizing to arbitrary action costs. As we shall see, the best-supporter formulation also generalizes easily to the use of h^C instead of h^1.

Best-supporter functions map facts to actions. Based on h^1, any fact p is mapped to an action achieving $h^1(\{p\})$, i.e., achieving the minimum in the h^1 equation (Equation 6, page 279). For unit-cost actions, the latter is equivalent to $h^1(pre(a)) = h^1(\{p\}) - 1$. We can hence write Keyder and Geffner's formulation of relaxed plans π^{FF} for the input task $\Pi = (\mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G})$ as $\pi^{\text{FF}} := \bigcup_{p \in \mathcal{G}} \pi(g)$, where π is a function on facts p that satisfies:

$$\pi(p) = \begin{cases} \emptyset & p \in \mathcal{I} \\ \bigcup_{q \in pre(a)} \pi(q) \cup \{a\} \text{ where } a \in \mathcal{A}, & \\ \quad p \in add(a), \text{and } h^1(pre(a)) = h^1(\{p\}) - 1 & \text{else} \end{cases} \tag{8}$$

It is easy to see that the action set π^{FF} can be sequentialized to form a relaxed plan for Π. The generalization to arbitrary cost can be done by working with $h^1(pre(a)) = h^1(\{p\}) - c(a)$ instead, and using a modified action-costs function $c' := c + \epsilon$, for some $\epsilon > 0$, in case there are 0-cost actions (ϵ can in principle be chosen so as to preserve optimality; this is a minor concern here as relaxed-plan heuristic functions are inadmissible anyway).

Note the special case where $h^1(pre(a)) = \infty$ for all a with $p \in add(a)$, and hence $h^1(\{p\}) = \infty$. In this situation, p does not have a best supporter in Keyder and Geffner's formulation. In our formulation, $\pi(p)$ is undefined (i.e., in our notation ∞ is *not* equal to $\infty - 1$). We abstract from this issue throughout the present subsection, just assuming that $h^1(\{p\}) < \infty$ for all facts. We will deal with the issue below in our extension to conjunction sets C, where π will be a *partial* function.

Note furthermore that we intentionally specify π to be "a" function that satisfies Equation 8. The relaxed plan π^{FF} is unique only up to tie-breaking. Keyder and Geffner's formulation moves the tie-breaking into the definition of best supporters. We find it more convenient, for our purposes here, to make the tie-breaking an explicit part of our equations (i. e., of Equation 8 and all relaxed-plan equations below).

Towards our generalization to arbitrary C, we first change Keyder and Geffner's equation to account for positive side effects, to the extent of supporting, with the same action, *all* open subgoals for which that action is a best supporter. Reformulating Equation 8 to this end, we obtain $\pi^{\mathrm{FF}} := \pi(\mathcal{G})$, where π is a function on fact sets G that satisfies:

$$\pi(G) = \begin{cases} \emptyset & G \subseteq \mathcal{I} \\ \pi((G \setminus G') \cup pre(a)) \cup \{a\} \text{ where } a \in \mathcal{A}, & \\ \quad \emptyset \neq G' = \{p \in G \mid p \in add(a), h^1(pre(a)) = h^1(\{p\}) - 1\} & \text{else} \end{cases} \tag{9}$$

Compared to Equation 8, we need to recurse not over single facts but over sets of facts, so that each recursive call "knows" the open facts and can select the entire best-supported subset thereof.[6]

Equation 9 is in correspondence with typical relaxed planning graph based implementations, as depicted in Algorithm 1. The definition of G' in the equation corresponds to the maintenance of "TRUE" flags for facts at relaxed planning graph layers, where upon selecting an action a at layer i all of a's add effects are marked as TRUE at i (to see this, observe that, with $h^1(pre(a)) = i - 1$, we have $h^1(pre(a)) = h^1(\{p\}) - 1$ iff $h^1(\{p\}) = i$). We will extend Algorithm 1 to relaxed plan extraction from h^C below. Note that Algorithm 1 deviates a bit from more common descriptions, explicitly including the computation of h^1 instead of assuming an input relaxed planning graph caching the outcome of this computation. This is just to simplify notation and to tie in easily with our extension below.

The step from Equation 8 to Equation 9 is benign in the standard setting, in the sense that its practical impact can be expected to be small: a single action typically does not add many open facts, i. e., does not support many open atomic subgoals. Yet this step is of paramount importance for our generalization of atomic subgoals to arbitrary conjunctions C. In that general setting, atomic subgoals typically overlap, and supporting a subgoal just means to add *part of it*, which may very well be the case for many subgoals.

We finally need to formulate the delete relaxation, not in terms of relaxing the regression semantics, but in terms of splitting subgoals up into singleton facts, and considering the correct regression semantics but separately with respect to each of these singleton-fact subgoals. In other words, we need to use the formulation underlying h^+ in Equation 5. As a reminder for convenience, that equation is: $h(G) =$

$$\begin{cases} 0 & G \subseteq \mathcal{I} \\ 1 + \min_{a \in \mathcal{A}, \emptyset \neq G' = \{p \mid p \in G, R(\{p\}, a) \neq \bot\}} h((G \setminus G') \cup \bigcup_{p \in G'} R(\{p\}, a)) & \text{else} \end{cases}$$

Doing a similar transformation step to Equation 9, we obtain $\pi^{\mathrm{FF}} := \pi(\mathcal{G})$, where π is a function on fact sets G that satisfies $\pi(G) =$

$$\begin{cases} \emptyset & G \subseteq \mathcal{I} \\ \pi((G \setminus G') \cup G'_r) \cup \{a\} \text{ where } a \in \mathcal{A}, & \\ \quad \emptyset \neq G' = \{p \mid p \in G, R(\{p\}, a) \neq \bot, h^1(R(\{p\}, a)) = h^1(\{p\}) - 1\} & \text{else} \end{cases} \tag{10}$$

6. Note that this selection is dynamic as a function of the open facts, as opposed to the up-front design of a best-supporter function sharing supporting actions as much as possible. This is not important in the standard setting here. Yet, as we discuss in detail below, it does become important when using arbitrary conjunctions C as atomic subgoals.

Algorithm 1: Relaxed plan extraction from h^1.

compute $h^1(\{p\})$ for all $p \in \mathcal{F}$
$m := \max_{p \in \mathcal{G}} h^1(\{p\})$
if $m = \infty$ **then**
    return $\perp$
for $i := m, \ldots, 1$ **do**
    $G_i := \{p \mid p \in \mathcal{G}, h^1(\{p\}) = i\}$
$\pi := \emptyset$
for $i := m, \ldots, 1$ **do**
    while ex. $p \in G_i$ s.t. p not TRUE at i **do**
        select $a \in \mathcal{A}$ where $\emptyset \neq \{p \in G_i \mid p \text{ not TRUE at } i\} \cap add(a)$,
        and $h^1(pre(a)) = i - 1$
        foreach $p \in G_i \cap add(a)$ **do**
            mark p TRUE at i
        foreach $q \in pre(a)$ **do**
            $G_{h^1(\{q\})} := G_{h^1(\{q\})} \cup \{q\}$
        $\pi := \pi \cup \{a\}$
return π

with G'_r defined as $G'_r := \bigcup_{p \in G'} R(\{p\}, a)$.

Relative to Equation 9, this is a simple reformulation, using regression notation which trivializes for singleton conjunctions. In particular, the subgoal $G'_r = \bigcup_{p \in G'} R(\{p\}, a)$ generated by the action simplifies to $pre(a)$ here. Relative to Equation 5, instead of a heuristic value, we compute a relaxed plan. Instead of minimizing over all action choices which corresponds to h^+, we impose the use of best supporters which corresponds to relaxed plan extraction from h^1.

4.2 Relaxed Plan Extraction from h^C

From Equation 10, we obtain π^{CFF} by similar generalizations as we made to get from h^+ to h^{C+}. Extending h^C to sets G of conjunctions by $h^C(G) := \max_{c \in G} h^C(c)$, our definition reads:

Definition 3 *Let $\Pi = (\mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G})$ be a planning task, and C a set of conjunctions in Π containing all singleton conjunctions. A h^C*-based critical-path delete-relaxed plan, *short C*-relaxed plan, *is a set π^{CFF} of action occurrences (a, G') where $\pi^{\mathrm{CFF}} = \pi(\mathcal{G}^C)$, with π being a partial function on conjunction sets G that is defined on $\mathcal{G}^C$ and satisfies $\pi(G) =$*

$$\begin{cases} \emptyset & \forall c \in G : c \subseteq \mathcal{I} \\ \pi((G \setminus G') \cup {G'_r}^C) \cup \{(a, G')\} \text{ where } a \in \mathcal{A}, & \\ \quad \emptyset \neq G' \subseteq \{c \mid c \in G, R(c,a) \neq \perp, h^C(R(c,a)) = h^C(c) - 1\}, & \\ \quad \text{and } h^C(G'_r) = h^C(G') - 1 & \text{else} \end{cases} \tag{11}$$

with G'_r *defined as* $G'_r := \bigcup_{c \in G'} R(c, a)$.

A h^C-based no cross-context critical-path delete-relaxed plan*, short* nc-*C*-relaxed plan*, is a set* π^{CFF}_{nc} *of action occurrences with the same property, except that we define* $G'_r := \{R(c, a) \mid c \in G'\}$.

This definition parallels the definition of h^{C+} (Definition 2). The subgoaling structure is the same, over sets of conjunctions from C each of which must be achieved through regression. Instead of a heuristic value, we compute a relaxed plan (consisting of action *occurrences*, action a plus supported subgoals G' as in h^{C+} and Π^C, as opposed to actions as in the standard case). Instead of minimizing over all action occurrence choices, we impose the use of best supporters according to h^C. The major new source of complexity, relative to Equation 10, is that we allow G' to be a *subset* of the best-supported atomic subgoals, similarly as in Definition 2. Because a relaxed plan does not always exist, we allow π to be partial and define π^{CFF} only if π is defined on the goal. As we show below, this is possible iff $h^C < \infty$, i. e., a C-relaxed plan exists iff any relaxed plan for Π^C exists.

Observe that, relative to Equation 10, we have added the new additional condition $h^C(G'_r) = h^C(G') - 1$. To understand this condition, consider that (a) $G'_r = \bigcup_{c \in G'} R(c, a)$ is the *union* over the regressions from each individual supported subgoal $c \in G'$, and that (b) for every such subgoal $c \in G'$ we have $h^C(R(c, a)) = h^C(c) - 1$, i. e., the action a selected is a best supporter for c. From (b), one would surmise that $h^C(G'_r) = h^C(G') - 1$, because the regressions $R(c, a)$ in G'_r each are one step easier to solve than their original counterparts $c \in G$. That is only so, however, if there are no cross-context conditions: otherwise, the union (a) may be more difficult to achieve than each of its components. We get back to this in detail below in Section 4.3. For now, just keep in mind that the additional condition $h^C(G'_r) = h^C(G') - 1$ is required due to possible cross-context conditions. (We remark that the condition is equivalent to $h^C(G'_r) < h^C(G')$, as the h^C value cannot decrease by more than 1 in a single regression step; we have written it as $h^C(G'_r) = h^C(G') - 1$ merely to use the most specific write-up.)

It is instructive to consider π^{CFF} from a procedural perspective. Algorithm 2 provides a corresponding extension of Algorithm 1. Where previously we computed h^1 for all facts, now we compute h^C for all conjunctions in C. Where previously our subgoal sets G_i were sets of facts, now they are sets of conjunctions from C. Where previously the new subgoals generated were the selected action's precondition facts, now they are the atomic conjunctions contained in the regressed subgoal G'_r. Note here the pointwise interpretation for nc-C-relaxed plans, where $G'_r = \{R(c, a) \mid c \in G'\}$ is a set of fact sets. In line 10, we now select an action occurrence (a, G') instead of just an action a, resulting in the additional choice of supported atomic subgoals G', and the accordingly more complicated structure of the regressed subgoal G'_r. Note here that, for every $c \in G_i$, we have $h^C(c) = i$ and, for any action a', $h^C(R(c, a')) \geq h^C(c) - 1 = i - 1$. Furthermore, if $h^C(G'_r) = i - 1$ then $h^C(R(c, a)) \leq i - 1$ for every $c \in G'$. Putting these observations together, we get $h^C(R(c, a)) = h^C(c) - 1$ for every $c \in G'$, and the choice of G' in Algorithm 2 is equivalent to that in Equation 11.

Example 6 *Consider, as in Example 5 (page 281), our car-driving example with* C *containing the singleton conjunctions as well as* $c = carY \wedge fuel$*. We have* $h^C(\{carX\}) = 0$, $h^C(\{fuel\}) = 0$, $h^C(\{carY\}) = 1$, $h^C(\{carY, fuel\}) = 2$*, and* $h^C(\{carZ\}) = 3$.

Algorithm 2: Relaxed plan extraction from h^C. For C-relaxed plan, use $G'_r = \bigcup_{c \in G'} R(c, a)$; for nc-$C$-relaxed plan, use $G'_r = \{R(c, a) \mid c \in G'\}$.

compute $h^C(c)$ for all $c \in C$
$m := \max_{c \in C, c \subseteq \mathcal{G}} h^C(c)$
if $m = \infty$ **then**
  return $\perp$
for $i := m, \ldots, 1$ **do**
  $G_i := \{c \mid c \in C, c \subseteq \mathcal{G}, h^C(c) = i\}$
$\pi := \emptyset$
for $i := m, \ldots, 1$ **do**
  while ex. $c \in G_i$ s.t. c not TRUE at i **do**
    select (a, G') where $a \in \mathcal{A}, \emptyset \neq G' \subseteq \{c \in G_i \mid R(c, a) \neq \perp, c$ not TRUE at $i\}$, and $h^C(G'_r) = i - 1$
    foreach $c' \in G'$ **do**
      mark c' TRUE at i
    foreach $c' \in C$ s.t. ex. $c \in G'_r$ with $c' \subseteq c$ **do**
      $G_{h^C(c')} := G_{h^C(c')} \cup \{c'\}$
    $\pi := \pi \cup \{(a, G')\}$
return π

Tracing Equation 11 from the initializing call $\pi^{\text{CFF}} = \pi(\{carZ\})$, we get the exact same recursive development as in Example 5. First, $carZ$ is supported only by $(a_{YZ}, \{carZ\})$, where $h^C(\{carY, fuel\}) = 2 = 3 - 1 = h^C(\{carZ\}) - 1$ as required for the supported conjunction $c = carZ$. Similarly, $h^C(G'_r) = h^C(G') - 1$ for $G'_r = \{carY, fuel\}$ and $G' = \{carZ\}$ as for single supported conjunctions there is no difference. The recursive subgoal is $\{carY, fuel, carY \wedge fuel\}$. The only supporting action for $carY \wedge fuel$ is a_{refuel}. That action is a best supporter for $carY \wedge fuel$ as $h^C(\{carY\}) = 1 = h^C(\{carY, fuel\}) - 1$. The action is not a best supporter for fuel though, because fuel is true initially $h^C(\{fuel\}) = 0$. So the only possible choice for supporting $carY \wedge fuel$ is a_{refuel} with $G' := \{carY \wedge fuel\}$. We get $G'_r = \{carY\}$ and $h^C(G'_r) = 1 = h^C(G') - 1$ as desired. The recursive subgoal is $\{fuel, carY\}$, supported by $(a_{XY}, \{carY\})$ yielding $G'_r = \{carX\}$ with $h^C(G'_r) = 0 = h^C(G') - 1$.

Taking the procedural perspective in Algorithm 2, we start by inserting $carZ$ into G_3. At layer $i = 3$ we support this by $(a_{YZ}, \{carZ\})$ with the same $G'_r = \{carY, fuel\}$ and $h^C(G'_r) = 2 = i - 1$. Similarly, layers 2 and 1 mirror exactly the respective recursive invocations of Equation 11.

We next prove that C-relaxed plans and nc-C-relaxed plans do indeed correspond to relaxed plans for Π^C respectively Π^C_{nc}. We start with C-relaxed plans:

Theorem 4 *Let $\Pi = (\mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G})$ be a planning task, and C a set of conjunctions in Π containing all singleton conjunctions. Then any C-relaxed plan π^{CFF} can be sequentialized to form a relaxed plan for Π^C.*

Proof Sketch: Sequencing π^{CFF} as $\langle(a_0, G'_0), \ldots, (a_{n-1}, G'_{n-1})\rangle$ in inverse order of action occurrence selection in Equation 11, i. e., placing the outcome of recursive invocations up front, $\langle a_0[G'_0], \ldots, a_{n-1}[G'_{n-1}]\rangle$ is a relaxed plan for Π^C. This is easy to show by induction over the length of the sequence. With $G_0, \ldots, G_n$ being the recursive subgoals generated in Equation 11, and s_i being the state after applying $a_i[G'_i]$ in Π^C, it holds that $\{\pi_c \mid c \in G_i\} \subseteq s_i$. This is obvious for $i = 0$. If it holds at i, it also holds at $i+1$ because (a) the $G_{i+1} \setminus G'_i$ part of G_{i+1} is also part of G_i and hence true by induction hypothesis; and (b) the G'_i part of G_{i+1} is made true by $a_i[G'_i]$, which is applicable to s_i by induction hypothesis because its precondition conjunctions are contained in G_i. ∎

An almost identical proof shows the corresponding property for nc-C-relaxed plans:

Theorem 5 *Let $\Pi = (\mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G})$ be a planning task, and C a set of conjunctions in Π containing all singleton conjunctions. Then any nc-C-relaxed plan π^{CFF}_{nc} can be sequentialized to form a relaxed plan for Π^C_{nc}.*

Finally, a relaxed plan for Π^C respectively Π^C_{nc} exists if and only if a C-relaxed plan respectively an nc-C-relaxed plan exists. This is simply because all of these properties are fully determined by the critical-path component. Our proof shows this via deriving an intermediate equation, Equation 12 below, which characterizes the behavior of π^{CFF} and π^{CFF}_{nc} when restricting the choice of supported subgoal sets G' to singletons. Equation 12 will play an important role in the comparison to related work, and in our experiments.

Theorem 6 *Let $\Pi = (\mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G})$ be a planning task, and C a set of conjunctions in Π containing all singleton conjunctions. Then a C-relaxed plan exists if and only if an nc-C-relaxed plan exists if and only if $h^C < \infty$.*

Proof Sketch: We show the claim in two parts, (a) a C-relaxed plan exists if and only if $h^C < \infty$, and (b) an nc-C-relaxed plan exists if and only if $h^C < \infty$. The "only if" directions follow as corollaries of (a) Theorems 3 and 4 respectively (b) Theorems 3 and 5: if $h^C = \infty$, neither a C-relaxed plan nor an nc-C-relaxed plan can exist, because otherwise a relaxed plan for Π^C respectively Π^C_{nc} would exist by Theorem 4 respectively Theorem 5, in contradiction to $h^C = \infty = h^1(\Pi^C) = h^1(\Pi^C_{nc})$ as per Theorem 3.

For the "if" directions, we consider versions of π^{CFF} and π^{CFF}_{nc} restricting the choice of supported subgoal sets G' to singletons, i. e., to single conjunctions $G' = \{c\}$. Each of π^{CFF} and π^{CFF}_{nc} then simplifies to $\bigcup_{c \in \mathcal{G}^C} \pi(c)$, with $\pi(.)$ being a partial function on conjunctions c that satisfies

$$\pi(c) = \begin{cases} \emptyset & c \subseteq \mathcal{I} \\ \bigcup_{c' \in R(c,a)^C} \pi(c') \cup \{(a, \{c\})\} \text{ where } a \in \mathcal{A}, & \\ \quad R(c,a) \neq \bot, \text{and } h^C(R(c,a)) = h^C(c) - 1 & \text{else} \end{cases} \tag{12}$$

Note the similarity to Equation 8 (page 285): we are now back to a more common notation for relaxed plan extraction (over C instead of singleton facts), extracting best supporters one-by-one.

By changing the subgoaling structure, one can transform Equation 12 into the form $\pi(\mathcal{G})$ where π is a partial function on fact sets G that satisfies

$$\pi(G) = \begin{cases} \emptyset & G \subseteq \mathcal{I} \\ \pi(R(G,a)) \cup \{(a,G)\} \text{ where } a \in \mathcal{A}, & \\ \quad R(G,a) \neq \bot, \text{ and } h^C(R(G,a)) = h^C(G) - 1 & G \in C \\ \bigcup_{G' \subseteq G, G' \in C} \pi(G') & \text{else} \end{cases} \tag{13}$$

Comparing this to the h^C equation (Equation 3, page 278), it is clear that the subgoaling structure of the two equations coincides for subgoals c with $h^C(c) < \infty$, and in particular, if $h^C < \infty$ then Equation 13 has a solution π defined on $\mathcal{G}$. Therefore, Equation 12 has a solution defined on all $c \in \mathcal{G}^C$. As Equation 12 captures a restricted version of π^{CFF} and π^{CFF}_{nc}, C-relaxed and nc-C-relaxed plans exist as desired. ■

4.3 The Subgoal-Support Selection Problem

We have so far shown how C-relaxed plans and nc-C-relaxed plans can be extracted. We have not yet explained how our actual heuristic functions h^{CFF} and h^{CFF}_{nc} are defined. Given Theorem 6, both h^{CFF} and h^{CFF}_{nc} return ∞ in the case $h^C = \infty$. For the case $h^C < \infty$, our description of relaxed plan extraction so far does not specify how to choose the supported subgoal sets G'. Taking that choice in particular ways yields the functions h^{CFF} and h^{CFF}_{nc}. The choice is non-trivial because the number of possible action occurrences is worst-case exponential in $|C|$. We refer to this choice as the *subgoal-support selection problem*.

We start by discussing the optimization objective for that problem. Then we fix solutions, for π^{CFF}_{nc} and π^{CFF} in this order, defining the desired heuristics h^{CFF}_{nc} and h^{CFF} through corresponding specializations of Equation 11. We close the section with a brief discussion of prior work in the light of our findings.

4.3.1 The Optimization Objective

Assume that $h^C < \infty$. As argued in the proof of Theorem 6, we know that Equation 12 has a solution, so in principle we could restrict ourselves to $|G'| = 1$, resulting in at most $|\mathcal{A}| * |C|$ different action occurrence choices. However, this can result in dramatic overestimation:

Example 7 *Consider the task $\Pi = (\mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G})$ where $\mathcal{F} = \{g_1, \dots, g_n\}$, $\mathcal{I} = \emptyset$, $\mathcal{G} = \{g_1, \dots, g_n\}$ and $\mathcal{A}$ contains the single action a whose precondition and delete list are empty and whose add list is $\{g_1, \dots, g_n\}$. Obviously, $h^* = h^+ = h^{\text{FF}} = 1$. However, even with C containing only the singleton conjunctions $\{g_i\}$, Equation 12 results in dramatic overestimation: the C-relaxed plan will collect a separate occurrence $(a, \{g_i\})$ for every g_i, resulting in relaxed plan length n. If C also contains all fact-pair conjunctions $\{g_i, g_j\}$ then we get a C-relaxed plan of size $n + \frac{n*(n-1)}{2}$. In general, we get a C-relaxed plan of size $|C|$.*

While this is an extreme example, similar situations arise whenever conjunctions overlap, because an action a adding a single fact p then is a possible supporter of all conjunctions that contain p. With, e.g., the conjunctions containing all fact pairs, this means that the number of top-level goal conjunctions supported by a is at least $|\mathcal{G}|$. In domains with many top-level goal facts – including most current IPC benchmarks and, more generally, e.g. typical transportation, construction, puzzle problems – this is clearly detrimental. (Replacing $\mathcal{G}$ by a single fact and a new goal-achiever action only moves the problem to the precondition of that action.)

This is essentially the same observation made by Haslum (2009, 2012), non-admissibility of $h^+(\Pi^m)$ as every conjunction must be achieved separately, which prompted the design of Π^C where every action may achieve arbitrary subsets of conjunctions. What is new here is the particular context in which we consider this issue, namely the choice of G' in π^{CFF} and π^{CFF}_{nc} as per Equation 11: We moved the issue from the generic planning-task level to the specific subgoal-support selection level. This more specific perspective identifies the precise source of complexity, as far as relaxed plan extraction is concerned: *How to choose the sets G' in Equation 11 – equivalently, how to implement line 10 in Algorithm 2 – in a manner avoiding overestimation to the extent possible?*

The intuitive answer to this question, given Example 7, certainly is *choose G' to be as large as possible*. This intuition is not entirely correct. As we detail in Example 9 (Appendix A), there are cases where supporting a conjunction c, even though it is feasible, is better done later on in the recursion, with an action whose precondition is easier to combine with c. Nevertheless, we employ $|G'|$ maximization here, deeming it safe to presume that overlapping conjunctions as per Example 7 are much more practically relevant than contrived situations as per Example 9.

It will be convenient to introduce a terminology for feasible choices of G'. As per Equation 11, the possible choices of G' are those where a is a best supporter for every $c \in G'$, i. e., $h^C(R(c,a)) = h^C(c) - 1$, and where the overall regressed subgoal is feasible, $h^C(G'_r) = h^C(G') - 1$. In this case, we say in the π^{CFF} context, i. e., with $G'_r = \bigcup_{c \in G'} R(c,a)$, that G' is *C-feasible*. We say in the π^{CFF}_{nc} context, i. e., with $G'_r = \{R(c,a) \mid c \in G'\}$, that G' is nc-*C-feasible*.

Our maximization problems then are:

Definition 4 *By* C-SubgoalSupport *we denote the following problem:*

Given a planning task Π, a set of conjunctions C in Π containing all singleton conjunctions, $G \subseteq C$, an action a in Π, and $K \in \mathbb{N}$. Does there exist $G' \subseteq \{c \in G \mid R(c,a) \neq \bot, h^1(R(c,a)) = h^1(c) - 1\}$ such that G' is C-feasible and $|G'| \geq K$?

We define nc-C-SubgoalSupport *accordingly for nc-C-feasible G'.*

4.3.2 THE h^{CFF}_{nc} HEURISTIC

The subgoal-support selection problem for π^{CFF}_{nc}, i. e., nc-C-SubgoalSupport, is easy to solve. Indeed, *any* choice of G' is nc-C-feasible:

Proposition 2 *Let Π be a planning task, C a set of conjunctions in Π containing all singleton conjunctions, $G \subseteq C$, and a an action in Π. Then any $G' \subseteq \{c \in G \mid R(c,a) \neq \bot, h^1(R(c,a)) = h^1(c) - 1\}$ is nc-C-feasible.*

Proof: By definition, G' is nc-C-feasible if $h^C(G'_r) = h^C(\{R(c,a) \mid c \in G'\}) = h^C(G') - 1$. Now, $h^C(\{R(c,a) \mid c \in G'\}) = \max_{c \in G'} h^C(R(c,a))$ which by construction equals $\max_{c \in G'}(h^C(c) - 1)$. The latter equals $(\max_{c \in G'} h^C(c)) - 1 = h^C(G') - 1$ as desired. ∎

In other words, for π^{CFF}_{nc}, the additional condition $h^C(G'_r) = h^C(G') - 1$ in Definition 3 is redundant. To maximize $|G'|$, we can simply include into G' *all* c where $h^C(R(c,a)) = h^C(G') - 1$. Accordingly, we define our heuristic function h^{CFF}_{nc} as:

Definition 5 *Let $\Pi = (\mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G})$ be a planning task, and C a set of conjunctions in Π containing all singleton conjunctions. The* nc-C-relaxed plan heuristic *is defined as $h^{\text{CFF}}_{nc} = \infty$ if $h^C = \infty$, and otherwise $h^{\text{CFF}}_{nc} = |\pi^{\text{CFF}}_{nc}|$ where $\pi^{\text{CFF}}_{nc} = \pi(\mathcal{G}^C)$ and π satisfies $\pi(G) =$*

$$\begin{cases} \emptyset & \forall c \in G : c \subseteq \mathcal{I} \\ \pi((G \setminus G') \cup {G'_r}^C) \cup \{(a, G')\} \text{ where } a \in \mathcal{A}, & \\ \quad \emptyset \neq G' = \{c \mid c \in G, R(c,a) \neq \bot, h^C(R(c,a)) = h^C(c) - 1\} & \text{else} \end{cases} \tag{14}$$

with G'_r defined as $G'_r := \{R(c,a) \mid c \in G'\}$.

In words, we restrict Equation 11 to a maximal choice of G' in the middle case, using $G' = \{c \in G \mid R(c,a) \neq \bot, h^C(R(c,a)) = h^C(c) - 1\}$ instead of $G' \subseteq \{c \in G \mid R(c,a) \neq \bot, h^C(R(c,a)) = h^C(c) - 1\}$. We use the Π^C_{nc} variant of the regressed subgoal G'_r, and we drop the condition $h^C(G'_r) = h^C(G') - 1$ which is redundant for that variant.

4.3.3 THE h^{CFF} HEURISTIC

Matters are not that simple for π^{CFF}, i. e., C-SubgoalSupport, which requires C-feasible sets G' as opposed to nc-C-feasible ones. The feasible choice of G' in the π^{CFF}_{nc} setting is trivial (Proposition 2) *because π^{CFF}_{nc} ignores cross-context conditions*. Not ignoring these conditions, in π^{CFF}, this is no longer true:

Example 8 *Consider, as in Example 5 (page 281), our abstract example with conjunctions $c_{q1p} = q_1 \wedge p$, $c_{q2p} = q_2 \wedge p$, and $c_{q1q2} = q_1 \wedge q_2$. After supporting each of the goal facts, we get the subgoal $G = \{q_1, q_2, p, q_1 \wedge p, q_2 \wedge p\}$. Ignore, like in Example 5, the subsumed subgoals q_1, q_2, p which can be tackled as a side effect of tackling the non-subsumed ones $q_1 \wedge p$ and $q_2 \wedge p$. The only possible supporting action for the latter subgoals is a_p which adds p (q_1 is true initially, and the action adding q_2 deletes p so cannot support $q_2 \wedge p$). There are three possible choices of G': $G'_{12} := \{q_1 \wedge p, q_2 \wedge p\}$, $G'_1 := \{q_1 \wedge p\}$, or $G'_2 := \{q_2 \wedge p\}$.*

For G'_1, $G'_r = \bigcup_{c \in G'_1} R(c,a) = \{q_1\}$ and $h^C(\{q_1\}) = 0 = h^C(\{q_1, p\}) - 1$. So G'_1 is C-feasible. For G'_2, $G'_r = \bigcup_{c \in G'_2} R(c,a) = \{q_2\}$ and $h^C(\{q_2\}) = 1 = h^C(\{q_2, p\}) - 1$. So G'_2 is C-feasible as well. However, G'_{12} is not *C-feasible, because $G'_r = \bigcup_{c \in G'_{12}} R(c,a) = \{q_1, q_2\}$, corresponding to the atomic conjunction $q_1 \wedge q_2$. Selecting both atomic subgoals $q_1 \wedge p$ and $q_2 \wedge p$, even though each is feasible individually, incurs the cross-context condition $q_1 \wedge q_2$, an atomic conjunction not present in the regression from either of $q_1 \wedge p$ or $q_2 \wedge p$ individually. The example is constructed so that $h^C(\{q_1, q_2\}) = \infty$, hence in particular $h^C(\{q_1, q_2\}) = \infty \neq h^C(G'_{12}) - 1 = h^C(\{\{q_1, p\}, \{q_2, p\}\}) - 1 = 1$.*

Note here that, for G'_{12} in π^{CFF}_{nc}, we get $G'_r = \{R(c,a) \mid c \in G'_{12}\} = \{\{q_1\}, \{q_2\}\}$ instead. In other words, we get a set containing two small conjunctions q_1 and q_2, instead of a set containing one big conjunction $q_1 \wedge q_2$. We have $h^C(\{\{q_1\}, \{q_2\}\}) = 1 = h^C(\{\{q_1, p\}, \{q_2, p\}\}) - 1$, so G'_{12} is (not C-feasible but) nc-C-feasible.

As the example shows, *cross-context conditions may render particular combinations of supported conjunctions in G' infeasible*. Having used this formulation, it should come as no surprise that maximizing $|G'|$ while avoiding such combinations is computationally hard:

Theorem 7 *C-SubgoalSupport is **NP**-complete.*

Proof Sketch: Membership by guess and check. Hardness via a reduction of Hitting Set: Given a set of elements E and a collection of subsets $b \subseteq E$ of elements, the construction is such that, at a particular point during C-relaxed plan extraction, choosing G' amounts to choosing $E' \subseteq E$, where E' is C-feasible (results in a h^C value $\neq \infty$) iff there exists no b with $b \subseteq E'$. Given this, $E' \setminus E$ is a hitting set, and maximizing $|E'|$ is equivalent to finding a minimum-size such set. ■

So it is hard to find a cardinality-maximal feasible set of supported conjunctions in π^{CFF}. Presuming that we do not want to invest the effort to solve that problem exactly (many times during the extraction of a C-relaxed plan on every search state), we need an approximate solution. A canonical choice for approximating cardinality-maximality is subset-maximality. We say that $G' \subseteq \{c \in G \mid R(c,a) \neq \bot, h^1(R(c,a)) = h^1(c) - 1\}$ is *subset-maximally C-feasible* if G' is C-feasible and, for every G'' such that $G' \subsetneq G'' \subseteq \{c \in G \mid R(c,a) \neq \bot, h^1(R(c,a)) = h^1(c) - 1\}$, G'' is not C-feasible. Our heuristic function h^{CFF} is defined using the corresponding restriction of Equation 11:

Definition 6 *Let $\Pi = (\mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G})$ be a planning task, and C a set of conjunctions in Π containing all singleton conjunctions. The C-*relaxed plan heuristic *is defined as $h^{\text{CFF}} = \infty$ if $h^C = \infty$, and otherwise $h^{\text{CFF}} = |\pi^{\text{CFF}}|$ where $\pi^{\text{CFF}} = \pi(\mathcal{G}^C)$ and π satisfies $\pi(G) =$*

$$\begin{cases} \emptyset & \forall c \in G : c \subseteq \mathcal{I} \\ \pi((G \setminus G') \cup {G'_r}^C) \cup \{(a, G')\} \textit{ where } a \in \mathcal{A}, & \\ \quad \emptyset \neq G' \subseteq \{c \mid c \in G, R(c,a) \neq \bot, h^C(R(c,a)) = h^C(c) - 1\}, & \\ \quad \textit{and } G' \textit{ is subset-maximally } C\textit{-feasible} & \textit{else} \end{cases} \tag{15}$$

with G'_r defined as $G'_r := \bigcup_{c \in G'} R(c,a)$.

A subset-maximally C-feasible set G' can be found through simple greedy algorithms, adding conjunctions one-by-one as shown in Algorithm 3. The candidate conjunctions are those $c \in G$ with $R(c,a) \neq \bot$ and $h^C(R(c,a)) = h^C(c) - 1$. Starting with empty G', we just try each candidate c exactly once. This suffices to get a subset-maximal G' because, as G' can only grow, if adding c was not feasible the first time around then adding c cannot be feasible later on either.

Algorithm 3: Greedy selection of a subset-maximally C-feasible set of supported subgoals G' in C-relaxed plan extraction. Implements line 10 in Algorithm 2 to obtain the heuristic function h^{CFF}.

```
select c ∈ G_i, c not TRUE at i
select a ∈ A with R(c, a) ≠ ⊥ and h^C(R(c, a)) = h^C(c) − 1
G' := {c}
foreach c' ∈ G_i s.t. c' not TRUE at i, R(c', a) ≠ ⊥, and h^C(R(c', a)) = h^C(c') − 1 do
    if G' ∪ {c'} is C-feasible then
        G' := G' ∪ {c'}
```

We remark that, as Example 9 (Appendix A) shows, there are cases where selecting a *non*-subset-maximally C-feasible G' leads to a strictly smaller C-relaxed plan. In other words, like cardinality maximization, subset-maximization is not fail-safe.

4.3.4 PREVIOUS WORKS RELATED TO THE SUBGOAL-SUPPORT SELECTION PROBLEM

Interestingly, while the subgoal-support selection problem has never previously been identified, it has already been solved. More plainly put, the previous works in this area can be viewed as solving the problem at a very abstract level, not identifying what precisely the problem is, and thus ending up with solutions that do solve the problem, but using unnecessarily drastic measures. Mostly this is due to the compilation view, where relaxed plan extraction becomes a standard technique, yet the subgoal-support selection problem has to be solved at the STRIPS level, in the form of the compiled task. The single non-compilation-view prior work, by Alcázar et al. (2013), was conducted as part of a much broader scope, and does not address the subgoal-support selection problem in detail.

Let us have a closer look at Alcazar et al.'s heuristic, FF^m (which they implement for $m = 2$). This extracts a relaxed plan from h^m, restricting C to contain exactly the conjunctions of size $\leq m$. That restriction is easily removed. In our notation, FF^m then corresponds to a C-relaxed plan extracted using this equation: $\mathrm{FF}^m = \bigcup_{c \in \mathcal{G}^C} \pi(c)$ where

$$\pi(c) = \begin{cases} \emptyset & c \subseteq \mathcal{I} \\ \bigcup_{c' \in R(c,a)^C} \pi(c') \cup \{a\} \text{ where } a \in \mathcal{A}, & \\ \quad R(c,a) \neq \bot, \text{and } h^C(R(c,a)) = h^C(c) - 1 & \text{else} \end{cases} \tag{16}$$

This is almost exactly what the definitions of both π^{CFF} and π^{CFF}_{nc} simplify to when restricting the choice of G' to support only a single conjunction $G' = \{c\}$, i. e., it is almost identical to Equation 12 as derived in the proof to Theorem 6. Repeating Equation 12 for convenience: $\pi^{\mathrm{CFF}} = \pi^{\mathrm{CFF}}_{nc} = \bigcup_{c \in \mathcal{G}^C} \pi(c)$ where

$$\pi(c) = \begin{cases} \emptyset & c \subseteq \mathcal{I} \\ \bigcup_{c' \in R(c,a)^C} \pi(c') \cup \{(a, \{c\})\} \text{ where } a \in \mathcal{A}, & \\ \quad R(c,a) \neq \bot, \text{and } h^C(R(c,a)) = h^C(c) - 1 & \text{else} \end{cases}$$

The only difference between these two equations is that FF^m collects *a set of actions* as in the standard setting, while π^{CFF} and π^{CFF}_{nc} collect *a set of (single-supported-subgoal) action occurrences*.

In this sense, Alcazar et al.'s approach over-simplifies the choice of G', to singleton sets. It effectively tackles Π^C_{nc} rather than Π^C because, with $|G'| = 1$, cross-context conditions never occur. It would furthermore run the risk of excessive overestimation as pointed out in Example 7 – if it did actually collect action occurrences, rather than actions. The latter might be viewed as a trick to avoid overestimation, yet from a theoretical perspective it rather defeats the purpose of using explicit conjunctions in the first place. Whereas relaxed planning on Π^C converges to h^*, FF^m is bounded from above by the number of actions, $|\mathcal{A}|$.

Altogether, our findings allow to understand prior work on this subject as follows:

- Π^C **Compilation (Haslum, 2012):** Includes one compiled action $a^{G'}$ for every possible pair of action a and possible set of supported subgoals G'. In this sense, it solves the **NP**-complete problem C-SubgoalSupport enumeratively, in-memory.[7]

  Lesson learned in h^{CFF}: *There is no need to pre-generate all possible conjunction subsets an action could support. We can focus on the subgoals that actually arise during relaxed plan extraction.*

7. Plus, without actually giving an optimality guarantee: the optimal $a^{G'}$ will be in the set of choices for relaxed plan extraction/the best-supporter function, but there is no guarantee that it will be selected.

- Π^C_{ce} **Compilation (Keyder et al., 2012, 2014):** Includes one conditional effect for every pair of action a and possibly supported conjunction c. This ignores cross-context conditions and hence trivializes C-SubgoalSupport into nc-C-SubgoalSupport.
  Lesson learned in h^{CFF}: *There is no need to ignore cross-context conditions completely. We can greedily select supported conjunctions whose cross-context conditions are feasible.*
- **FF^m (Alcázar et al., 2013):** Restricts the conjunction set C to the size-$\leq m$ conjunctions as in h^m. Restricts the supported subgoals G' to single conjunctions, thus trivializing C-SubgoalSupport and ignoring cross-context conditions like the Π^C_{ce} compilation. Collects actions instead of action occurrences, losing convergence to h^*.
  Lesson learned in h^{CFF}: *All of these weaknesses can be avoided.*

5. Experiments

We evaluate the benefits of the h^{CFF} and h^{CFF}_{nc} heuristic functions relative to the most closely related previous heuristics. We state the key issues that we will consider in terms of four *hypotheses*, formulating our major expectations regarding algorithm behavior *on IPC benchmarks*, the standard means for evaluation in the planning community:[8]

(H1) For h^{CFF} relative to $h^{\text{FF}}(\Pi^C)$, the hypothesis is that *(H1) avoiding the exponential blow-up in $|C|$ typically yields a faster heuristic and thus improved performance*.

(H2) For h^{CFF} relative to $h^{\text{FF}}(\Pi^C_{ce})$, the hypothesis is that *(H2) accounting for cross-context conditions can yield a more informed heuristic and thus improved performance*.
The difference between "typically" and "can" in (H1) vs. (H2) is intended. Cross-context conditions presumably are important only in particular cases, whereas the advantage of h^{CFF}'s smaller representation presumably helps in most cases.

(H3) For both h^{CFF} and h^{CFF}_{nc} relative to $h^{\text{FF}}(\Pi^C_{ce})$, the hypothesis is that *(H3) the implementation of h^{CFF} and h^{CFF}_{nc} typically is more effective and thus yields improved performance*.
We expect this to be so as h^{CFF} and h^{CFF}_{nc} are direct, not using a compilation, and thus are more specialized than that of $h^{\text{FF}}(\Pi^C_{ce})$.
Note here that h^{CFF}_{nc} and $h^{\text{FF}}(\Pi^C_{ce})$ are equivalent except for the implementation, in that they use the same information *and* have the same scaling behavior in $|C|$. (In contrast to the comparison between h^{CFF} vs. $h^{\text{FF}}(\Pi^C)$, which is dominated by (H1) the drastically different scaling behavior in $|C|$.)

(H4) We furthermore compare h^{CFF} to a variant we denote $h^{\text{CFF}}_{|G'|=1}$, as per Equation 12 where we restrict to $|G'| = 1$, the hypothesis being that *(H4) the non-trivial subgoal support selection in h^{CFF} typically yields a more informed heuristic and thus improved performance*.

We finally include a variant we denote $h^{\text{CFF}}_{|G'|=1\mathcal{A}}$, as per Equation 16 where $|G'| = 1$ and a set of actions (as opposed to action occurrences) is selected. This serves as a comparison to Alcázar et al.'s (2013) work.

8. The hypotheses are not intended as formal statements that we will statistically accept or reject; nor are they intended as an exhaustive representation of all issues we will discuss. They merely serve as a "red thread" in the discussion of our large-scale experiments.

Section 5.1 describes some key points of our implementation, Section 5.2 explains our experimental setup. Section 5.3 provides comprehensive results, across planner variants, for small conjunction sets C which turn out to be best in terms of overall performance. Section 5.4 then analyzes behavior as a function of growing C.

5.1 Implementation

We implemented h^C, h^{CFF}, and h^{CFF}_{nc} in FD (Helmert, 2006). We furthermore implemented the *C-additive heuristic* h^{Cadd}, defined exactly like h^C (Equation 3) except that the maximization over atomic subgoals is replaced by a summation over these subgoals:

$$h(G) = \begin{cases} 0 & G \subseteq \mathcal{I} \\ 1 + \min_{a \in \mathcal{A}, R(G,a) \neq \bot} h(R(G,a)) & G \in C \\ \sum_{G' \subseteq G, G' \in C} h(G') & \text{else} \end{cases} \tag{17}$$

In other words, the step from h^C to h^{Cadd} parallels that from $h^{\max}$ to h^{add} (Bonet & Geffner, 2001). We don't use h^{Cadd} as a heuristic function per se: in contrast to the standard setting, atomic subgoals overlap in the general case, so that summation doesn't make sense. We use h^{Cadd} as an alternative best-supporter function for relaxed plan extraction. For that purpose, it turns out to be fairly useful empirically.

As computing critical-path heuristics becomes expensive with many conjunctions, a key to practicality is an efficient implementation of h^C. To that end, we extend the counter-based algorithm originally implemented in FF (Hoffmann & Nebel, 2001) for computing h^1 (aka a relaxed planning graph). Our extended algorithm is easily described as a modification of the original algorithm. Assume an input state s. FF's original algorithm associates the precondition of each action $a \in \mathcal{A}$ with a *counter*, denoted here $count(a)$, initialized to $|pre(a)|$. Facts p are maintained in a priority queue ordered by an associated $v(p)$ value, which equals $h^1(p)$ once p has been dequeued. The queue is initialized with the facts p true in s, each associated with value $v(p) = 0$. The main loop dequeues facts, activates new actions, and maintains the v values. When a fact p is dequeued, a loop over all actions a with $p \in pre(a)$ decrements $count(a)$. If this results in $count(a) = 0$ then the action is activated, enqueuing every $q \in add(a)$ with value $v(q) = 1 + \max_{p' \in pre(a)} v(p')$, or reducing $v(q)$ to that value in case q is already in the queue with a higher value.[9] The algorithm stops if either all goal facts have been dequeued and $h^1(s) = \max_{p \in \mathcal{G}} v(p)$, or the queue has become empty and $h^1(s) = \infty$.

Our extension to h^C works in much the same way. We just need to maintain the *values* $v(.)$ *for conjunctions* $c \in C$ instead of single facts, and we need to maintain *counters for pairs of action and supported conjunction* instead of just actions. Precisely, we create a counter $count(c, a)$ for every $c \in C$ and $a \in \mathcal{A}$ where $R(c, a) \neq \bot$ and $R(c, a)$ does not contain a mutex, i. e., a fact pair known to be unreachable. The latter corresponds to *mutex pruning* as discussed by Keyder et al. (2014) for Π^C and Π^C_{ce}. It reduces computational effort as well as strengthens the heuristic.

In the extended algorithm, each counter $count(c, a)$ is initialized to $|\{c' \mid c' \in C, c' \subseteq R(c, a)\}|$, i. e., to the number of sub-conjunctions we need to make true in order to be able to achieve c using a (remember here that C contains all singleton conjunctions). The

9. For general action costs $c(a)$, one can simply use $v(q) = c(a) + \max_{p' \in pre(a)} v(p')$ here.

queue now contains conjunctions $c' \in C$ instead of facts. It is initialized with the conjunctions $c' \subseteq s$, with $v(c') = 0$. Dequeueing a conjunction c', we loop over all counters $count(c,a)$ where $c' \subseteq R(c,a)$, decrementing $count(c,a)$. If this results in $count(c,a) = 0$, we enqueue/upgrade c with value $1 + \max_{c' \in C, c' \subseteq R(c,a)} v(c')$.

To compute h^{Cadd}, we use the exact same algorithm except that maximization is replaced by summation, i. e., instead of $1 + \max_{c' \in C, c' \subseteq R(c,a)} v(c')$ we use $1 + \sum_{c' \in C, c' \subseteq R(c,a)} v(c')$.

Based on the conjunction values $v(c)$ computed for h^C respectively h^{Cadd}, the implementation of h^{CFF} and h^{CFF}_{nc} follows Algorithm 2 (page 288). In particular, we select the action/supported subgoals (a, G') as previously discussed. For h^{CFF}_{nc} we fix $G' = \{c \mid c \in G, R(c,a) \neq \perp, v(R(c,a)) = v(c) - 1\}$. For h^{CFF}, we select G' as per Algorithm 3 (page 294). Here, the $v(c)$ values of single conjunctions c are readily available. Values $v(X)$ for a fact set X are required for $X := R(c,a)$, as well as during the check for C-feasibility in h^{CFF} (Algorithm 3 line 5), where $X := \bigcup_{c'' \in G' \cup \{c'\}} c''$ with G' being the current set of supported subgoals and c' being a candidate for inclusion into that set. We compute $v(X)$ by a loop over the facts $p \in X$, using lists $C[p]$ containing the $c \in C$ where $p \in c$, and maximizing respectively summing over $v(c)$ for those $c \in C[p]$ where $c \subseteq X$.

Helpful actions (Hoffmann & Nebel, 2001), i. e., FD preferred operators, are defined similarly as for h^{FF}. An action a applicable to state s is preferred in s if the C-relaxed plan contains a, i. e., an action/supported subgoals pair of the form (a, G'). This corresponds to the selection of preferred operators a in $h^{\text{FF}}(\Pi^C)$ and $h^{\text{FF}}(\Pi^C_{ce})$, based on compiled actions $a[C']$ occurring in relaxed plans in the respective compilations (Keyder et al., 2014).

The performance of satisficing search in planning is known to be brittle with respect to minor differences in the heuristic functions (e. g. Valenzano, Sturtevant, Schaeffer, & Xie, 2014). This is important also in our setting. The unavoidable implementation differences between our new heuristics and their predecessors turn out to be a major complication for a fair comparison. All heuristics extract relaxed plans from a h^C or h^{Cadd} best-supporter function, yet $h^{\text{FF}}(\Pi^C)$ and $h^{\text{FF}}(\Pi^C_{ce})$ do so via a compilation, while h^{CFF} and h^{CFF}_{nc} do not. The relaxed plan extraction algorithms work on different representations. In particular, the choice of an "action" in $h^{\text{FF}}(\Pi^C)$, i. e., of a compiled action $a[C']$, corresponds to the choice of an action/supported subgoals pair (a, G') in h^{CFF}. By design, $h^{\text{FF}}(\Pi^C)$ cannot distinguish between choosing a vs. choosing G'. In contrast, by design h^{CFF} chooses first only the action a and then assembles G' by greedy C-feasible maximization.

To offset these unavoidable differences in relaxed plan extraction, we experiment across a variety of tie-breaking strategies in the choice of best supporters. In Π^C and Π^C_{ce}, the tie-breaking applies to compiled actions $a[C']$, in h^{CFF} and h^{CFF}_{nc} it applies to actions a supporting the same conjunction c and where $h^C(R(c,a)) = h^C(c) - 1$ respectively $h^{\text{add}}(R(c,a)) = h^{\text{add}}(c) - 1$. Our strategies are:

Arbitrary: Choose an arbitrary best supporter, i. e., the first one we find. This is used (with h^{add} best supporters) in FD's implementation of h^{FF}, as well as $h^{\text{FF}}(\Pi^C)$ and $h^{\text{FF}}(\Pi^C_{ce})$.

Random: Choose a random best supporter. We use 3 different random seeds in our experiments to gauge the performance variance incurred by this criterion. It turns out that, in almost all cases, the variance is small and the performance change relative to arbitrary tie-breaking is consistent across random seeds, i. e., consistently positive or consistently negative. Thus random tie-breaking exhibits reliable behavior as an algorithm option.

Difficulty (h^C only): This is the tie breaking mechanism used in FF (Hoffmann & Nebel, 2001). It selects a best supporter that minimizes the summed-up $h^{\max}$ values of the supporter's preconditions. In h^{CFF} and h^{CFF}_{nc}, this translates to summing up the h^C values of the conjunctions contained in the regressed subgoals $R(c, a)$. Remaining ties are broken arbitrarily.

We use each of these 3 tie-breaking strategies with h^C, and the first 2 strategies with h^{Cadd}, in each of the 6 heuristics h^{CFF}, h^{CFF}_{nc}, $h^{\mathrm{FF}}(\Pi^C)$, $h^{\mathrm{FF}}(\Pi^C_{ce})$, $h^{\mathrm{CFF}}_{|G'|=1}$, and $h^{\mathrm{CFF}}_{|G'|=1\mathcal{A}}$.

5.2 Experiments Setup and Design

To be able to compare the different heuristic functions directly, in all such comparisons we use the same conjunction set C for every heuristic. We find these sets C using the exact same methods, and implementation, as used by Keyder et al. (2014). The motivation is that our contribution here does not pertain to methods finding C, and re-using the established methods provides for better comparability. To understand our experiments, it is not necessary to understand the generation of C in detail, so we give a brief summary only.

Keyder et al.'s (2014) method is a variant of the method proposed earlier on by Haslum (2012). In a pre-process to the actual search, C is learned by iteratively refining a delete-relaxed plan on the initial state. Starting with empty C, a relaxed plan π^+ for Π^C is generated. If π^+ is a real plan (a plan for the original input task), the process stops. Else, an analysis step finds a set C' of new conjunctions which exclude π^+, i. e., such that π^+ is no longer a relaxed plan for Π^C when setting $C := C \cup C'$. Then the process iterates. Running this ad infinitum, one will eventually find a plan for the input task. But that is typically not feasible. To find instead a set C for heuristic search, the algorithm applies both, a *time limit T*, and a *size limit x* on Π^C relative action set size increase, i. e., on $|\mathcal{A}^{\pi C}|/|\mathcal{A}|$. If either of the two criteria applies, the process stops and hands over the current set C to the search. (Each limit may be set to ∞, meaning that this termination criterion is disabled.)

As all heuristics in our experiments use explicit conjunctions, and all use the same set C, we separate the generation of C from the actual experiments. We apply separate runtime limits for C-generation and search respectively, and we will report only about the performance of search not about that of C-learning. Given this, T merely serves as a means to keep the experiments feasible even for large size limits x. We fix T to 30 minutes.

Throughout, we use FD's lazy-greedy best-first search with a dual open queue for preferred operators (Helmert, 2006), which profits from the search space pruning afforded by preferred operators, yet preserves completeness by keeping the pruned nodes in the second open queue. This is the canonical search algorithm for satisficing planning with delete-relaxation heuristics, widely used as a baseline that yields competitive performance while being reasonably simple. (Textbook single-queue greedy best-first search lags far behind the state of the art, as it can either not use preferred operators, or loses the solutions in those cases where preferred operators are too restrictive.) The experiments were run on a cluster of machines with Intel Xeon E5-2660 processors running at 2.2 GHz. The memory limit was set to 4 GB. We used the benchmarks from the satisficing tracks of the two most recent International Planning Competitions, IPC'11 and IPC'14. We do not include

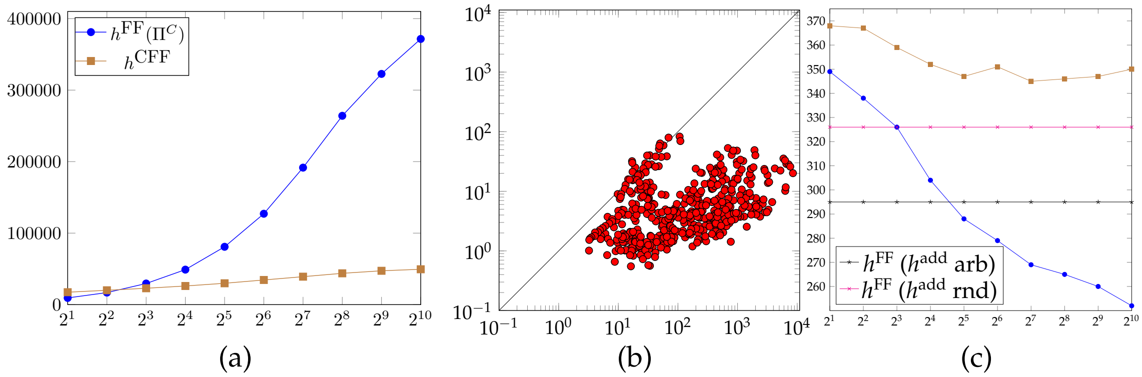


Figure 1: Data preview for h^{CFF} vs. $h^{FF}(\Pi^C)$. (a) Number of counters in h^{CFF} vs. number of actions $|\mathcal{A}^{\pi C}|$ in Π^C, as a function of the size limit x. (b) States per second with $x = \infty$, x-axis h^{CFF}, y-axis $h^{FF}(\Pi^C)$. (c) Total coverage as a function of x. In (b) and (c), each of h^{CFF} and $h^{FF}(\Pi^C)$ is run with h^{Cadd} using random tie-breaking. In (c), we also include h^{FF} as a baseline, with two tie-breaking variants: h^{add} with arbitrary tie-breaking corresponds to the FD default, h^{add} with random tie-breaking has better performance on these benchmarks. Recall in this comparison that the effort for C-learning is *not* included in h^{CFF} and $h^{FF}(\Pi^C)$ (see text).

the IPC'14 CityCar domain, in which the FD translator generates actions with conditional effects, not supported by our implementation. For each domain, each test suite has 20 instances (some domains have been used in both IPC'11 and IPC'14 so have two test suites).

With 6 heuristic functions, 5 best-supporter definitions (h^C vs. h^{Cadd}, tie breaking), and the numeric size-limit parameter x, the experiments space is large. To motivate how we organize our exploration of that space in what follows, Figure 1 gives a data preview.

The major benefit of h^{CFF} over $h^{FF}(\Pi^C)$ is the better scaling in $|C|$. One would expect this to manifest itself, for large C, in (a) a smaller representation and thus (b) a faster heuristic function. Figure 1 (a) and (b) confirm that this is indeed so.[10] For $x = \infty$, where C contains the conjunctions learned within 30 minutes, we get speed-ups of 1–4 orders of magnitude. Now, while this is good news, it turns out that in most cases large C is detrimental. While search space size generally does decrease when increasing the size limit x, all heuristic functions also become slower. The slowdown is dramatic for $h^{FF}(\Pi^C)$. It is much less dramatic for h^{CFF}, but still typically enough to outweigh the search space reduction. Figure 1 (c) shows the effect: overall coverage becomes worse with growing x, dramatically for $h^{FF}(\Pi^C)$, in a more benign manner but still almost monotonically for h^{CFF}. The best overall coverage is most often (across heuristics, configurations, domains) obtained at $x = 2$, which is also the best setting of x in Keyder et al.'s (2014) experiments.

The h^{FF} baselines in Figure 1 (c) are based on h^{CFF} using only the singleton conjunctions ($x = 1$), for better comparability with our methods, and to have the same 5 tie-breaking strategies at our disposal. In comparison to these baselines, h^{CFF} consistently out-

10. In Figure 1 (a), for $x = 2$ the number of counters in h^{CFF} exceeds $|\mathcal{A}^{\pi C}|$. This cannot happen in theory, as per Definition 1, because Π^C includes an action $a[\{c\}]$ for every counter $count(c, a)$. It does happen in practice only due to the handling of facts, i. e., the action's original pre/add/del lists: while Definition 1 handles these as singleton conjunctions, our implementation of Π^C uses the more effective special-case handling as per Haslum's (2012) definition.

	h^{CFF}					$h^{\text{FF}}(\Pi^C)$					$h^{\text{FF}}(\Pi^C_{ce})$					h^{CFF}_{nc}				
	h^C			h^{Cadd}		$h^1(\Pi^C)$			$h^{\text{add}}(\Pi^C)$		$h^1(\Pi^C_{ce})$			$h^{\text{add}}(\Pi^C_{ce})$		h^C			h^{Cadd}	
domain	arb	rnd	dif	arb	rnd	arb	rnd	dif	arb	rnd	arb	rnd	dif	arb	rnd	arb	rnd	dif	arb	rnd
Barman'11	3	2	8	8	12	0	0	1	14	1	1	0	2	**15**	2	2	2	8	11	13
Barman'14	1	0	3	3	7	1	0	0	7	0	1	0	1	**8**	1	1	0	3	4	7
CaveDiving'14	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7
ChildSnack'14	2	0	**14**	2	1	2	0	0	0	0	2	0	0	0	0	2	0	**14**	2	1
Elevators'11	11	19	18	**20**	**20**	8	12	15	19	18	9	12	15	**20**	19	11	**20**	18	**20**	**20**
Floortile'11	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
Floortile'14	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
GED'14	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
Hiking'14	15	**17**	14	14	16	16	10	16	15	13	16	11	15	15	12	15	16	12	15	15
Maintenance'14	10	13	10	9	**14**	11	10	11	11	11	11	9	11	11	12	10	12	11	10	**14**
Nomystery'11	7	6	10	7	7	7	6	10	**12**	6	7	6	11	**12**	6	7	6	10	7	7
Openstacks'11	19	19	19	19	**20**	**20**	16	**20**	**20**	**20**	**20**	16	**20**	19	**20**	**20**	19	19	**20**	**20**
Openstacks'14	**17**	**17**	15	15	16	15	8	15	14	16	14	9	13	12	16	**17**	16	15	13	**17**
Parcprinter'11	**16**	10	13	12	12	12	12	13	12	7	8	11	10	9	10	**16**	10	14	12	13
Parking'11	5	**20**	8	8	14	2	19	4	11	**20**	2	16	4	12	**20**	5	**20**	7	8	14
Parking'14	1	17	1	4	9	0	9	0	5	**20**	0	8	0	5	19	1	**20**	1	5	9
Pegsol'11	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
Scanalyzer'11	17	18	**20**	**20**	**20**	18	18	**20**	19	**20**	18	19	**20**	19	**20**	18	**20**	**20**	**20**	**20**
Sokoban'11	**18**	16	17	17	17	**18**	16	**18**	17	17	**18**	16	16	17	17	**18**	17	17	17	17
Tetris'14	7	7	8	8	8	4	7	3	5	8	4	**9**	4	4	**9**	7	**9**	8	8	7
Thoughtful'14	13	13	9	10	11	13	10	**15**	12	10	13	9	**15**	11	10	12	12	9	10	11
Tidybot'11	11	15	13	14	16	10	15	15	14	**19**	13	17	15	14	18	11	15	13	14	16
Transport'11	0	2	6	12	13	2	1	5	10	11	2	3	4	10	11	0	2	5	**14**	13
Transport'14	0	0	2	9	7	0	0	2	7	7	0	0	1	**10**	9	0	0	2	6	9
Visitall'11	2	**18**	3	16	17	15	13	16	16	14	3	13	3	16	14	2	**18**	3	**18**	**18**
Visitall'14	0	**4**	0	**4**	**4**	**4**	**4**	**4**	**4**	**4**	0	**4**	0	**4**	**4**	0	**4**	0	**4**	**4**
Woodworking'11	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
$\sum$	282	340	318	338	368	285	293	310	351	349	269	295	287	350	356	282	345	316	345	**372**

Table 1: Coverage results with $x = 2$, for h^{CFF}, $h^{\text{FF}}(\Pi^C)$, $h^{\text{FF}}(\Pi^C_{ce})$, and h^{CFF}_{nc}, with different best-supporter functions (h^C vs. h^{Cadd}) and tie-breaking strategies. Best results highlighted in **boldface**. Abbreviations: "arb" arbitrary tie-breaking; "rnd" random tie-breaking (per-instance median seed, see also Table 2); "dif" difficulty tie-breaking.

performs even the more competitive, non-standard h^{FF} variant with random tie-breaking. For $h^{\text{FF}}(\Pi^C)$, the same is true with small x values. Recall, however, that we report only the performance of search, *not* that of C-learning: the data here evaluates exclusively the merits of the respective heuristic functions, not the overhead required to obtain them in the first place. We will stick to this rationale throughout, as the differences between explicit-conjunction heuristics are our contribution here. For completeness, Appendix B shows coverage plots counting C-learning as part of the solving effort, i. e., with a 30-minute limit on the time taken by C-learning and search together.

Given the typically detrimental effect of large x, in what follows we first (Section 5.3) explore the case $x = 2$, examining in detail the space of heuristic functions and best supporters. Subsequently (Section 5.4), we examine in more detail what happens as we scale x. To make the latter experiments feasible, we will fix for each heuristic function the most performant best-supporter method; as it will turn out, for scaling x, this is the same method for every heuristic, namely h^{Cadd} with random tie-breaking.

5.3 Small C: Heuristics, Best Supporters, and Tie Breaking for $x = 2$

We examine first the performance of the "main" heuristic functions, i. e., h^{CFF} vs. the competing previous variants $h^{\text{FF}}(\Pi^{\text{C}})$ and $h^{\text{FF}}(\Pi^{\text{C}}_{ce})$, as well as h^{CFF}_{nc} which can essentially be perceived as an alternative, no-compilation, implementation of $h^{\text{FF}}(\Pi^{\text{C}}_{ce})$. We will discuss the behavior of $h^{\text{CFF}}_{|G'|=1}$ and $h^{\text{CFF}}_{|G'|=1\mathcal{A}}$, relative to h^{CFF}, below. Consider Table 1.

The most striking observation in this data is that *the differences between heuristic functions are dominated by those between tie-breaking strategies*. As a function of tie-breaking, the range of overall coverage is 282–368 for h^{CFF}, 285–351 for $h^{\text{FF}}(\Pi^{\text{C}})$, 269–356 for $h^{\text{FF}}(\Pi^{\text{C}}_{ce})$, and 282–372 for h^{CFF}_{nc}. This relatively small role of heuristic function differences, for $x = 2$, makes sense as the advantages of h^{CFF} and h^{CFF}_{nc} – different scaling in C, cross-context conditions, non-compilation implementation – naturally have more impact the larger C is. There are cases though where even small C makes a difference.

Comparing tie-breaking strategies, h^{Cadd} best supporters are superior to h^{C} best supporters, typically per domain and almost consistently in the total. This makes sense in that all heuristics here run the risk of over-estimation, and h^{Cadd} is better than h^{C} at finding cheap relaxed plans. There are several cases where some combination of heuristic and tie-breaking method works exceptionally well, e. g. $h^{\text{CFF}}/h^{\text{CFF}}_{nc}$ with h^{C} and difficulty tie-breaking in ChildSnack'14, $h^{\text{CFF}}/h^{\text{CFF}}_{nc}$ with h^{C} and arbitrary tie-breaking in Parcprinter'11, $h^{\text{FF}}(\Pi^{\text{C}})/h^{\text{FF}}(\Pi^{\text{C}}_{ce})$ with h^{add} and arbitrary tie-breaking in Barman'11. As these performance peaks are not consistent across tie-breaking methods for the respective heuristics, we consider them to be outliers caused by the brittleness of search.

Comparing heuristic functions h vs. h', a way of identifying strong advantages is to consider those domains in Table 1 where h has a *consistent* advantage over h', i. e., h is at least as good as h' for *all* tie-breaking methods, and is strictly better for at least one method. Call such an advantage *strict* if h is strictly better for all tie-breaking methods. In the comparison h^{CFF} vs. $h^{\text{FF}}(\Pi^{\text{C}})$, h^{CFF} has a consistent advantage in 5 domains (ChildSnack'14, Elevators'11, Openstacks'14, Tetris'14, and Transport'14), while $h^{\text{FF}}(\Pi^{\text{C}})$ has a consistent advantage in 2 domains (Sokoban'11 and Visitall'14). The advantage is strict only for h^{CFF} in Elevators: in all other cases, some tie-breaking methods work equally well for both heuristics. Overall, despite the noise the data is (somewhat) in favor of h^{CFF}.

The comparison h^{CFF} vs. $h^{\text{FF}}(\Pi^{\text{C}}_{ce})$ yields a similar picture, with 4 consistent (non-strict) advantages for h^{CFF} (ChildSnack'14, Elevators'11, Openstacks'14, Sokoban'11) vs. 1 consistent (non-strict) advantage for $h^{\text{FF}}(\Pi^{\text{C}}_{ce})$ (Tidybot'11). It is illuminating to offset these observations against the data for h^{CFF}_{nc}: in every domain where h^{CFF} has a consistent advantage over $h^{\text{FF}}(\Pi^{\text{C}}_{ce})$, h^{CFF}_{nc} also has a consistent advantage over $h^{\text{FF}}(\Pi^{\text{C}}_{ce})$, and the only domain where h^{CFF}_{nc} has a consistent disadvantage vs. $h^{\text{FF}}(\Pi^{\text{C}}_{ce})$ is the same as for h^{CFF}, Tidybot'11. Hence the reason for the differences between h^{CFF} and $h^{\text{FF}}(\Pi^{\text{C}}_{ce})$ here are not the cross-context conditions. Presumably, as cross-context conditions occur only in very specific situations, with small C they just do not play a role.

Further evidence towards this conclusion comes from the comparison h^{CFF} vs. h^{CFF}_{nc}. Actually, regarding cross-context conditions, that comparison is more informative than that between h^{CFF} vs. $h^{\text{FF}}(\Pi^{\text{C}}_{ce})$: after all, h^{CFF} and h^{CFF}_{nc} differ *only* in accounting respectively not accounting for cross-context conditions. In terms of consistent advantages, the comparison is clearly in favor of h^{CFF}_{nc}, with 8 consistent (non-strict) advantages for h^{CFF}_{nc}

	h^{CFF}								$h^{\text{FF}}(\Pi^C)$							
domain	s1	s2	s3	worst	med	best	var	δ	s1	s2	s3	worst	med	best	var	δ
Barman'11	13	12	12	10	12	15	1	+5	3	1	1	1	1	3	2	−13
Barman'14	8	5	7	5	7	8	3	+5	0	0	1	0	0	1	1	−7
CaveDiving'14	7	7	7	7	7	7	0	0	7	7	7	7	7	7	0	0
ChildSnack'14	2	1	2	0	1	4	1	−1	1	1	2	0	0	4	1	+2
Elevators'11	20	20	19	19	20	20	1	−1	18	17	19	17	18	19	2	−2
Floortile'11	20	20	20	20	20	20	0	0	20	20	20	20	20	20	0	0
Floortile'14	20	20	20	20	20	20	0	0	20	20	20	20	20	20	0	0
GED'14	20	20	20	20	20	20	0	0	20	20	20	20	20	20	0	0
Hiking'14	18	14	15	13	16	18	4	+4	13	13	13	11	13	15	0	−2
Maintenance'14	13	13	13	10	14	15	0	+4	11	8	12	5	11	15	4	±3
Nomystery'11	6	7	7	6	7	7	1	−1	6	6	6	6	6	6	0	−6
Openstacks'11	20	20	20	20	20	20	0	+1	20	20	19	19	20	20	1	−1
Openstacks'14	18	16	16	16	16	18	2	+3	16	16	14	14	16	16	2	+2
Parcprinter'11	13	10	12	8	12	15	3	±2	9	9	9	6	7	14	0	−3
Parking'11	14	14	16	13	14	17	2	+8	20	20	20	20	20	20	0	+9
Parking'14	9	11	8	7	9	12	3	+7	20	17	19	16	20	20	3	+15
Pegsol'11	20	20	20	20	20	20	0	0	20	20	20	20	20	20	0	0
Scanalyzer'11	20	20	20	20	20	20	0	0	20	20	20	20	20	20	0	+1
Sokoban'11	16	17	18	16	17	18	2	±1	17	17	17	17	17	17	0	0
Tetris'14	8	9	7	6	8	10	2	±1	11	9	7	7	8	12	4	+6
Thoughtful'14	11	10	12	9	11	13	2	+2	11	10	11	8	10	14	1	−2
Tidybot'11	15	16	16	14	16	17	1	+2	17	18	17	14	19	19	1	+4
Transport'11	14	12	11	9	13	15	3	±2	12	11	11	9	11	14	1	+2
Transport'14	9	7	7	7	7	9	2	−2	8	6	7	5	7	9	2	±1
Visitall'11	17	18	17	17	17	18	1	+2	14	14	13	13	14	14	1	−3
Visitall'14	4	4	4	4	4	4	0	0	4	4	4	4	4	4	0	0
Woodworking'11	20	20	20	20	20	20	0	0	20	20	20	20	20	20	0	0
$\sum$	375	363	366	336	368	400			358	344	349	319	349	383		

Table 2: Coverage results with $x = 2$, showing the effect of different random seeds in random tie-breaking for h^{Cadd} best-supporters. "s1", "s2", "s3" denote the 3 random seeds (fixed throughout the experiment). The "best", "med", and "worst" columns assess per-instance aggregation methods, selecting the best/median/worst seed per instance respectively. "var" assesses the per-domain performance variance, in terms of the difference between the best and worst coverage. "δ" assesses the per-domain consistency of performance change relative to the baseline, i. e., relative to h^{Cadd} best-supporters with arbitrary tie-breaking. It shows the maximum absolute difference, with "+" if coverage is better for all seeds, "−" if coverage is worse for all seeds, and "±" otherwise, i. e., if coverage actually gets better or worse depending on the seed.

vs. 2 consistent (non-strict) advantages for h^{CFF}. However, examining this more closely, all these advantages are at a very small scale. Whereas, in the comparisons above, the average coverage difference in consistent-advantage domains is typically between 2 and 3, in the comparison between h^{CFF} and h^{CFF}_{nc} it is usually 0.2 and its maximum is 0.8.

What to conclude regarding our experimental hypotheses, (H1) advantage of h^{CFF} over $h^{\text{FF}}(\Pi^C)$ thanks to better scaling in $|C|$, (H2) advantage of h^{CFF} over $h^{\text{FF}}(\Pi^C_{ce})$ thanks to cross-context conditions, and (H3) advantage of h^{CFF} and h^{CFF}_{nc} over $h^{\text{FF}}(\Pi^C_{ce})$ thanks to implementation? Table 1 provides evidence in favor of (H1) and (H3), though only in few domains, and subject to substantial noise from tie-breaking. There is no support for (H2). The evidence suggests that, for $x = 2$, taking cross-context conditions into account has neither substantial positive effects nor substantial negative effects.

Some words are in order regarding our use of random tie-breaking. The crucial observations are that, per domain, (a) the variance over random seeds is typically small, while (b) the performance change relative to the baseline typically is consistent. This makes random tie-breaking comparable to other non-randomized algorithm options. That said, in some domains either (a) or (b) are false, and in the total the differences would sum up. We counteract this with a *per-instance aggregation method* to obtain per-instance data that reduces variance relative to the individual seeds, and that interpolates between the seeds in terms of overall performance. The *per-instance median seed*, of the 3 random seeds ran in our experiments, turns out to be suitable (for coverage, this counts an instance as solved if at least 2 of the 3 randomized runs solved it). We used the per-instance median seed in Figure 1 and Table 1, and will use it below in all cases where random tie-breaking is employed. Table 2 shows the data supporting these observations and design decisions.

A quick look at the "var" columns in Table 2 confirms observation (a). The difference between the best and the worst per-seed coverage is ≤ 2 except in 5 domains for h^{CFF} and in 3 domains for $h^{\text{FF}}(\Pi^C)$. On the other hand, looking at the bottom row and comparing the seeds, the differences do add up. This would be especially so if we were to select the best or worst seed per instance ("best" and "worst" columns), resulting in coverage differences of around 70 in the total. However, using the median ("med" column) seed results in a per-instance aggregation with the desired properties.

Regarding observation (b), consider the "δ" columns in Table 2. Those domains where performance relative to the baseline is *not* consistent, i. e., gets better or worse depending on the random seed, are marked with a "$\pm$" symbol. These symbols are sparse in the table. In all but 4 of the 27 domains for h^{CFF}, and in all but 2 of them for $h^{\text{FF}}(\Pi^C)$, the randomization changes performance consistently. (It rarely deteriorates performance for h^{CFF}, while for $h^{\text{FF}}(\Pi^C)$ the picture is more mixed depending on the domain.) This shows clearly that random tie-breaking is reliable against the baseline.

Indeed, these findings contradict Keyder et al.'s (2014) use of random tie-breaking as a measure of noise. Keyder et al.'s idea was to account for the brittleness of search by randomizing the baseline heuristic h (in their case, h^{FF} with h^{add} best supporters and arbitrary tie-breaking), measuring δ as in Table 2 yet ignoring the distinctions "$+$", "$-$", "$\pm$", i. e., considering only the absolute maximum difference. They deem a heuristic h' to be significantly better than h only if its improvement over h is larger than δ – intuitively, larger than the random noise. However, this approach assumes that random tie-breaking yields a distribution around the baseline average, which is very much not so in our data. Consider for example $h^{\text{FF}}(\Pi^C)$ in Barman'11. According to Keyder et al.'s method, the "random noise" here is 13, and for any other heuristic to be significantly better than $h^{\text{FF}}(\Pi^C)$ it must hence increase coverage by at least 14. But the "noise" is just the effect of random tie-breaking being consistently detrimental. Similar examples abound.

We conclude that, in the specific context of our experiments, Keyder et al.'s measure is not appropriate because random tie-breaking is typically *not* a source of noise. To the contrary, the performance of 3 separate runs of random tie-breaking can be reliably reported like that of a single planner run, through per-instance median seed aggregation.

Let us finally consider the behavior of h^{CFF} relative to $h^{\text{CFF}}_{|G'|=1}$ which trivializes the subgoal support selection, and relative to $h^{\text{CFF}}_{|G'|=1\mathcal{A}}$ which also trivializes the C-relaxed plan (into a set of actions instead of action occurrences). Table 3 shows the data.

	h^{CFF}					$h^{\text{CFF}}_{\|G'\|=1}$					$h^{\text{CFF}}_{\|G'\|=1\mathcal{A}}$				
	h^{C}			h^{Cadd}		h^{C}			h^{Cadd}		h^{C}			h^{Cadd}	
domain	arb	rnd	dif	arb	rnd	arb	rnd	dif	arb	rnd	arb	rnd	dif	arb	rnd
Barman'11	3	8	2	8	12	2	3	0	**13**	**13**	0	3	0	6	1
Barman'14	1	3	0	3	**7**	2	1	0	6	4	1	1	0	1	1
CaveDiving'14	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7
ChildSnack'14	2	**14**	0	2	1	2	10	1	1	2	2	10	0	2	0
Elevators'11	11	18	19	**20**	**20**	10	17	18	**20**	**20**	8	12	19	**20**	**20**
Floortile'11	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
Floortile'14	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
GED'14	**20**	**20**	**20**	**20**	**20**	**20**	19	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**
Hiking'14	15	14	**17**	14	16	10	10	14	14	13	12	11	15	15	14
Maintenance'14	10	10	13	9	**14**	4	5	4	6	5	7	10	10	10	10
Nomystery'11	7	**10**	6	7	7	7	6	5	6	6	6	9	5	8	7
Openstacks'11	19	19	19	19	**20**	19	19	**20**	**20**	19	**20**	19	**20**	19	**20**
Openstacks'14	17	15	17	15	16	17	17	**20**	16	15	16	17	16	16	16
Parcprinter'11	**16**	13	10	12	12	13	11	3	8	7	**16**	11	5	7	8
Parking'11	5	8	**20**	8	14	2	2	14	2	2	4	3	**20**	15	18
Parking'14	1	1	**17**	4	9	1	0	3	0	0	2	0	14	6	13
Pegsol'11	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
Scanalyzer'11	17	**20**	18	**20**	**20**	19	19	**20**	**20**	**20**	17	**20**	19	**20**	**20**
Sokoban'11	**18**	17	16	17	17	16	17	17	17	16	16	17	17	17	16
Tetris'14	7	8	7	8	8	7	4	**14**	4	11	8	10	11	8	10
Thoughtful'14	13	9	13	10	11	9	**14**	10	13	**14**	9	13	9	**14**	12
Tidybot'11	11	13	15	14	**16**	10	11	12	14	**16**	10	12	16	13	15
Transport'11	0	6	2	12	**13**	0	4	2	**13**	**13**	1	3	1	11	9
Transport'14	0	2	0	**9**	7	0	1	0	8	7	0	2	0	**9**	5
Visitall'11	2	3	18	16	17	12	11	**20**	**20**	**20**	1	1	**20**	**20**	19
Visitall'14	0	0	4	4	4	4	3	**11**	7	5	0	0	9	8	5
Woodworking'11	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
$\sum$	282	318	340	338	**368**	273	291	315	335	335	263	291	333	352	346

Table 3: Coverage results with $x = 2$, for h^{CFF} vs. $h^{\text{CFF}}_{|G'|=1}$ and $h^{\text{CFF}}_{|G'|=1\mathcal{A}}$, with different best supporter functions and tie-breaking strategies. Best results highlighted in **boldface**. Abbreviations as in Table 1.

Like in Table 1, there is a lot of noise due to tie-breaking strategies. Note though that all heuristics shown here work on the same representation and are based on the same implementation. The differences are *only* in the subgoal support selection/relaxed plan definition. Comparisons of tie-breaking strategies across heuristics hence are direct.

The data shows a clear advantage of h^{CFF} over $h^{\text{CFF}}_{|G'|=1}$. For every tie-breaking strategy, h^{CFF} is strictly better in the total (the margin varying from 3 for h^{Cadd} with arbitrary tie-breaking to 33 for h^{Cadd} with random tie-breaking). Using the per-domain comparison across tie-breaking strategies, h^{CFF} has a consistent advantage in 10 domains (3 of which are strict), and a consistent disadvantage only in 2 domains (both strict, namely the two Visitall variants). Comparing search space size and states per second on commonly solved instances, the advantages of h^{CFF} are partly due to quality (e. g. Parking'11 994.4 vs. 8,091.1 geometric mean of state evaluations), supporting our hypothesis (H4) that the non-trivial subgoal support selection in h^{CFF} yields a more informed heuristic than $h^{\text{CFF}}_{|G'|=1}$. There also are several cases where the advantage is in speed (e. g. Elevators'11 4713.0 vs. 4591.1 states per second). The only possible cause for this lies in the different states evaluated: those for h^{CFF} are easier to evaluate, which typically is the case for states closer to the goal. (We remark that, like for the standard relaxed plan heuristic, most of the heuristic function runtime, typically 90% or more, is spent on the computation of the best-supporter function, h^C respectively h^{Cadd} in our case.)

Relative to $h^{CFF}_{|G'|=1\mathcal{A}}$, h^{CFF} is still in the advantage but the picture is more mixed. In terms of total coverage, h^{CFF} is dominant except for h^{Cadd} with arbitrary tie-breaking. Per domain, h^{CFF} has a consistent advantage in 6 cases (1 strict), vs. a consistent disadvantage in 4 cases (none strict). It appears that, at least for this setting of x and the IPC benchmarks, relative to $h^{CFF}_{|G'|=1}$ which is prone to over-estimation, the trivialized C-relaxed plan in $h^{CFF}_{|G'|=1\mathcal{A}}$ does result in a better heuristic. Comparing $h^{CFF}_{|G'|=1\mathcal{A}}$ vs. $h^{CFF}_{|G'|=1}$ directly, $h^{CFF}_{|G'|=1\mathcal{A}}$ dominates in the total except for h^C with arbitrary tie-breaking, has a consistent advantage in 5 domains (3 strict), and a consistent disadvantage in 3 domains (none strict). In terms of search space size and states per second on commonly solved instances, the advantages of $h^{CFF}_{|G'|=1\mathcal{A}}$ are mostly due to quality (most notably in Parking'11, 1102.7 vs. 8091.1 state evaluations), except in Hiking'14 where $h^{CFF}_{|G'|=1\mathcal{A}}$ is faster (526.7 vs. 639.8 states per second).

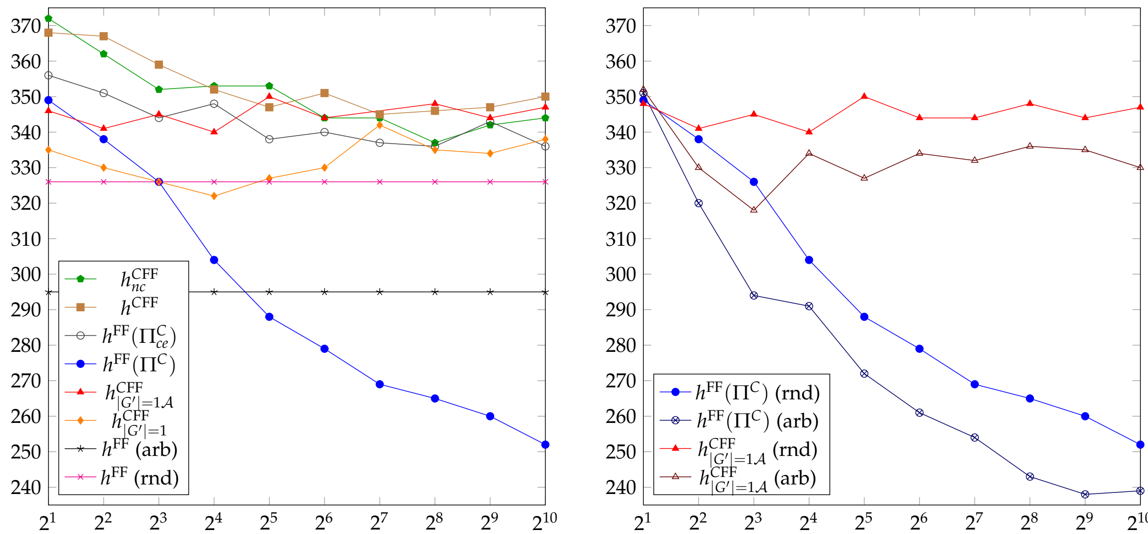
5.4 Scaling C: Performance as a Function of x

We now examine search behavior as C becomes larger. This is naturally presented in terms of plots of performance measures as a function of the size limit x. Keep in mind, especially in the comparison to the h^{FF} baselines, that the C-learning is conducted separately from the search, with a separate 30-minute time limit. Appendix B shows the same coverage plots included in the below, but when counting C-learning as part of the solving effort. In the following discussions, we include brief summaries of this data.

Figure 2 shows total coverage as a function of x. Consider first Figure 2 (b) which settles the question about the most competitive tie-breaking strategies. As we have seen in Tables 1 and 2, total coverage at $x = 2$ is maximal using h^{add} with random tie-breaking, for almost all heuristic functions. The two exceptions are $h^{FF}(\Pi^C)$ and $h^{CFF}_{|G'|=1\mathcal{A}}$. For both of these, h^{add} with arbitrary tie-breaking works better than h^{add} with random tie-breaking at $x = 2$ (2 more instances solved for $h^{FF}(\Pi^C)$, 6 more instances solved for $h^{CFF}_{|G'|=1\mathcal{A}}$). However, as Figure 2 (b) shows, the advantage of arbitrary tie-breaking at $x = 2$ turns into a substantial disadvantage for larger values of x. Hence, for the remainder of the experiments, we fix h^{add} with random tie-breaking as the best-supporter method throughout.

Consider now Figure 2 (a). As previously hinted (Figure 1 (c)), total coverage tends to decrease as a function of x. All heuristics consistently outperform both h^{FF} baselines though, except for $h^{FF}(\Pi^C)$ whose per-state runtime overhead drags coverage below that of h^{FF} once $x \geq 32$ (and except for a temporary dip of $h^{CFF}_{|G'|=1}$ below h^{FF} with random tie-breaking at $x = 16$).

Note that all of h^{CFF}, h^{CFF}_{nc}, $h^{FF}(\Pi^C)$, and $h^{FF}(\Pi^C_{ce})$ decrease, relatively speaking, more steeply up to $x = 2^5$, and less steeply afterwards. This is because, around this point, in many domains the C-learning reaches the time limit before the size limit. While $h^{FF}(\Pi^C)$ in the remaining domains still becomes substantially worse, for the other heuristics this is less pronounced. Observe here that the coverage difference between $x = 2$ and $x = 2^{10}$, except for $h^{FF}(\Pi^C)$, is small, around 20 instances. Indeed, most of this decrease is caused by a few domains only, namely Barman, Parking, Sokoban, and Visitall.



(a) h^{add}, random tie-breaking

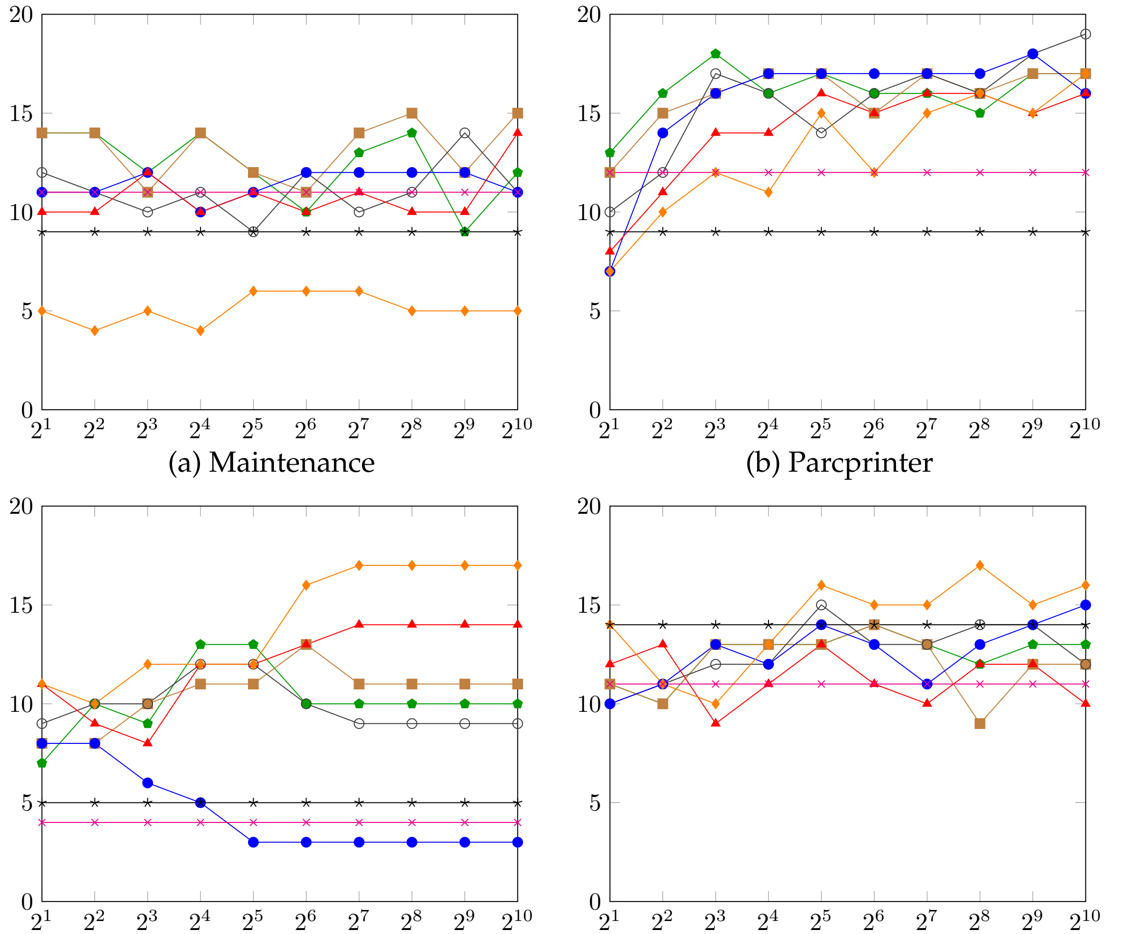
(b) h^{add}, random vs. arbitrary tie-breaking

Figure 2: Total coverage as a function of the size bound x. (a) All heuristics using h^{add} with random tie-breaking (median per-instance seed; heuristic functions listed top-down by order of their coverage for $x = 2$). (b) h^{add} with random vs. arbitrary tie-breaking for $h^{\text{FF}}(\Pi^C)$ and $h^{\text{CFF}}_{|G'|=1\mathcal{A}}$, the only two heuristics where arbitrary tie-breaking yields higher total coverage for $x = 2$. In (a), for comparison we include h^{FF} as a baseline, using h^{add} with arbitrary tie-breaking (FD default), and using h^{add} with random tie-breaking. Recall in this comparison that the effort for C-learning is *not* included in the explicit-conjunction heuristics.

The curves of h^{CFF}, h^{CFF}_{nc}, and $h^{\text{FF}}(\Pi^C_{ce})$ are fairly close to each other, and follow a similar pattern. h^{CFF} is consistently better than $h^{\text{FF}}(\Pi^C_{ce})$, and h^{CFF}_{nc} is consistently better than $h^{\text{FF}}(\Pi^C_{ce})$ except for $x = 2^9$. For $x \geq 2^6$, there is a consistent advantage for h^{CFF} over h^{CFF}_{nc}, indicating a beneficial impact of cross-context conditions.

Regarding $h^{\text{CFF}}_{|G'|=1}$ and $h^{\text{CFF}}_{|G'|=1\mathcal{A}}$, the latter is consistently much better than the former. $h^{\text{CFF}}_{|G'|=1\mathcal{A}}$ achieves very competitive performance for $x \geq 2^5$. Both heuristics exhibit no clear trend over x. The latter is not due to a difference in heuristic function speed (as we shall see below, the speed of $h^{\text{CFF}}_{|G'|=1}$ and $h^{\text{CFF}}_{|G'|=1\mathcal{A}}$ is similar to that of h^{CFF}, across x values). Rather, it is caused by particular behaviors in the few domains causing the coverage decline tendency in Figure 2 (a). Compared to the other heuristics, $h^{\text{CFF}}_{|G'|=1}$ and $h^{\text{CFF}}_{|G'|=1\mathcal{A}}$ scale better over x in Barman (only $h^{\text{CFF}}_{|G'|=1\mathcal{A}}$) and Visitall (both $h^{\text{CFF}}_{|G'|=1}$ and $h^{\text{CFF}}_{|G'|=1\mathcal{A}}$). There also are some cases, e. g. Barman for $h^{\text{CFF}}_{|G'|=1}$ and Parking for both $h^{\text{CFF}}_{|G'|=1}$ and $h^{\text{CFF}}_{|G'|=1\mathcal{A}}$, where these heuristics are bad from begin with, not solving in the first place those instances lost by other heuristics for larger x, and hence suffering less from large x.

In most domains other than Barman, Parking, Sokoban, and Visitall, the only heuristic suffering from large x is $h^{\text{FF}}(\Pi^C)$, if any heuristic suffers at all. On the other heuristics, growing x has only a marginally negative effect, an inconclusive effect, or no effect at all. There also are 4 domains where most heuristics tend to improve in coverage as x grows. Figure 3 shows the data for these. The coverage growth over x is most consistent across



(a) Maintenance (b) Parcprinter

(c) Tetris (d) Thoughtful

Figure 3: Coverage in individual domains. All explicit-conjunction heuristics use h^{add} with random tie-breaking. Recall in the comparison to the h^{FF} baselines that the effort for C-learning is *not* included in the explicit-conjunction heuristics.

heuristics in Parcprinter. In the other domains, the picture is more mixed, with a lot of variance in Maintenance and Thoughtful, and with mainly $h^{\text{CFF}}_{|G'|=1}$, $h^{\text{CFF}}_{|G'|=1\mathcal{A}}$, h^{CFF}, and h^{CFF}_{nc} profiting from large x in Tetris. (The curves remain flat in Tetris for $x \geq 2^7$ because then the C-learning time-limit applies and no more new conjunctions are added.)

How does this picture change when imposing a 30-minute limit on the time taken by C-learning and search together? Naturally, the tendency of coverage to decline over growing x becomes stronger, yet the relative performance of explicit-conjunction heuristics remains very similar.

For total coverage, shown in Figure 7 (page 323), performance is substantially worse than for search-only already at $x = 2$ (by 20-30 instances), and declines more steeply over x for all heuristics. For the relative performance of heuristics, however, our conclusions remain exactly the same as above. With respect to the baselines, only h^{CFF} and h^{CFF}_{nc} beat the non-default h^{FF} (random tie-breaking), and only at $x = 2$. The inferior default h^{FF}

baseline is beat by all explicit-conjunction heuristics up to medium-large x values ($x = 16$ or $x = 32$), except for $h^{\mathrm{FF}}(\Pi^C)$ which is worse for $x \geq 8$, and for $h^{\mathrm{CFF}}_{|G'|=1}$ which marginally beats default h^{FF} only at $x = 2$.

For the individual domains in Figure 3, the questions are whether (a) $x > 2$ still improves coverage when including the effort for C-learning, and whether (b) the explicit-conjunction heuristics can still beat the h^{FF} baselines. As Figures 8 and 9 (pages 324 and 325) show, the answer to both (a) and (b) is "yes". In Maintenance and Thoughtful, not much changes with respect to Figure 3. In Parcprinter and Tetris, very large values of x are detrimental, but moderate ones aren't. Coverage increases up to a certain point, namely $x = 2^3$ in Parcprinter and $x = 2^4$ in Tetris, then decreases after that point.

Let us get back to our hypotheses (H1)–(H4). Figure 2 (a) confirms (H1) that h^{CFF} typically has an advantage over $h^{\mathrm{FF}}(\Pi^C)$; somewhat supports that (H2) the cross-context conditions in h^{CFF} yield an advantage over $h^{\mathrm{FF}}(\Pi^C_{ce})$ (and, more directly, over h^{CFF}_{nc}); confirms that (H3) h^{CFF} and h^{CFF}_{nc} have an advantage over $h^{\mathrm{FF}}(\Pi^C_{ce})$; and (H4) confirms that h^{CFF} has an advantage over $h^{\mathrm{CFF}}_{|G'|=1}$. To examine the reason for these advantages, and thus evaluate the specific claim of each hypothesis, we now consider more fine-granular performance measures, namely search space size (number of state evaluations), states per second, and search runtime, on commonly solved instances.

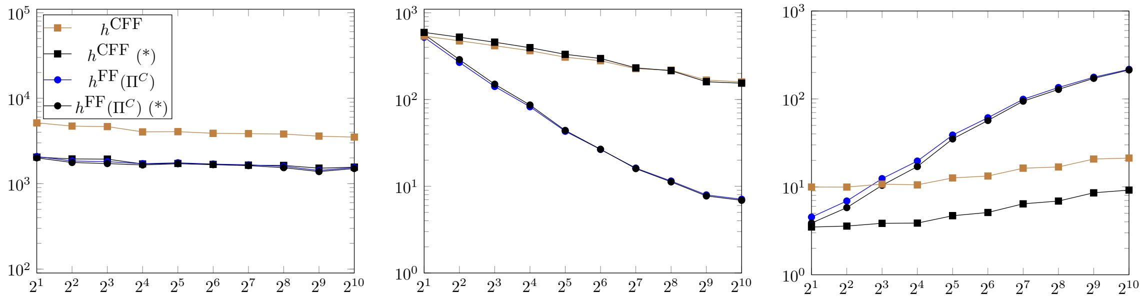
The specific claim of hypothesis (H1) is that, thanks to avoiding the exponential blow-up in $|C|$, h^{CFF} is typically faster than $h^{\mathrm{FF}}(\Pi^C)$ and thus improves performance. Figure 4 confirms this. The top row of plots shows the main data (the data for overall performance). As we see in the top left plot, in terms of quality the two heuristics are similar. In terms of speed, $h^{\mathrm{FF}}(\Pi^C)$ suffers with growing x, in the overall and consistently in individual domains, to the effect that runtime, like coverage discussed above, suffers as well. That is much less so for h^{CFF}, leading to a dramatic performance advantage for large x.[11]

On the other hand, as h^{CFF} also suffers itself from large x, though less than $h^{\mathrm{FF}}(\Pi^C)$, its advantage over $h^{\mathrm{FF}}(\Pi^C)$ in the overall is mute. Most relevant are the domains where, thanks to its speed advantage, h^{CFF} *benefits* from growing x, and hence improves over the *best* performance obtainable with $h^{\mathrm{FF}}(\Pi^C)$ for any value of x. For coverage, this happens in Maintenance and Tetris (both, with and without including C-learning, cf. Figures 3, 8, and 9). For search runtime on commonly solved instances, it happens in Hiking, Pegsol, Tetris, and Thoughtful. Figure 4 showcases Hiking and Tetris, which we will also use as showcases below as they nicely illustrate most of our main points.

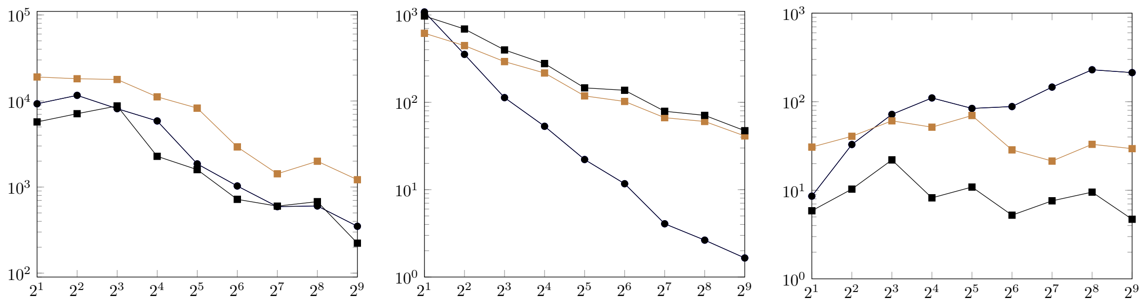
We next compare three heuristic functions with each other, namely h^{CFF}, h^{CFF}_{nc}, and $h^{\mathrm{FF}}(\Pi^C_{ce})$. This serves to examine hypotheses (H2) and (H3). The specific claim of the former asserts that, thanks to accounting for cross-context conditions, h^{CFF} can be more informed than $h^{\mathrm{FF}}(\Pi^C_{ce})$. The latter asserts that the implementation of h^{CFF} and h^{CFF}_{nc} is typically faster than that of $h^{\mathrm{FF}}(\Pi^C_{ce})$. It is of advantage to compare all three heuristics together as, to evaluate the importance of cross-context conditions, the comparison between h^{CFF} and h^{CFF}_{nc} is more direct. Figure 5 shows the data.

Hypothesis (H3) is confirmed very consistently, at a small scale. h^{CFF} and h^{CFF}_{nc} are faster than $h^{\mathrm{FF}}(\Pi^C_{ce})$ across all x values in the overall, with a small advantage that grows in x.

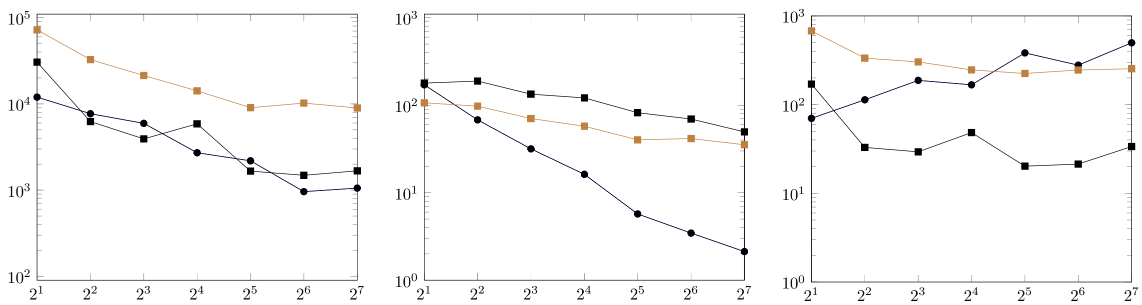
11. We remark that, for $x = 1$ where h^{CFF} and $h^{\mathrm{FF}}(\Pi^C)$ both are variants of h^{FF}, there are hardly any speed differences, neither between h^{CFF} and $h^{\mathrm{FF}}(\Pi^C)$ nor compared to FD's standard implementation of h^{FF}.



Overall: search space size, states per second, search runtime



Hiking: search space size, states per second, search runtime



Tetris: search space size, states per second, search runtime

Figure 4: Data for h^{CFF} vs. $h^{\text{FF}}(\Pi^{\text{C}})$. Geometric means. All curves use only those instances solved for all values of x, for the curves with a (*) solved by both heuristics (174 instances overall), for those without (*) solved by the respective heuristic.

Essentially the same behavior occurs in *every* individual domain, with a single exception, namely Tidybot where $h^{\text{FF}}(\Pi^{\text{C}}_{ce})$ is consistently faster. Hiking and Tetris in Figure 5 are two typical examples. In terms of coverage, h^{CFF} and h^{CFF}_{nc} consistently (across all or almost all values of x) dominate $h^{\text{FF}}(\Pi^{\text{C}}_{ce})$ in Barman, Elevators, Hiking, Maintenance, Sokoban, and Visitall; the opposite happens only in Parking and Tidybot.

Regarding (H2), as the top left plot in Figure 5 shows, all three heuristics yield similar search space sizes overall. There no domains where h^{CFF} consistently, across all values of x, yields smaller search spaces than h^{CFF}_{nc}. However, there are domains where h^{CFF} has a notable advantage for large values of x. This is mainly so for Hiking and Tetris, shown in Figure 5. In Tetris, the advantage is basically consistent beyond $x = 2^4$. Hiking behaves similarly except for a degradation at the largest two x values. In both domains, h^{CFF} also has corresponding coverage advantages over h^{CFF}_{nc}. Overall, the support for hypothesis

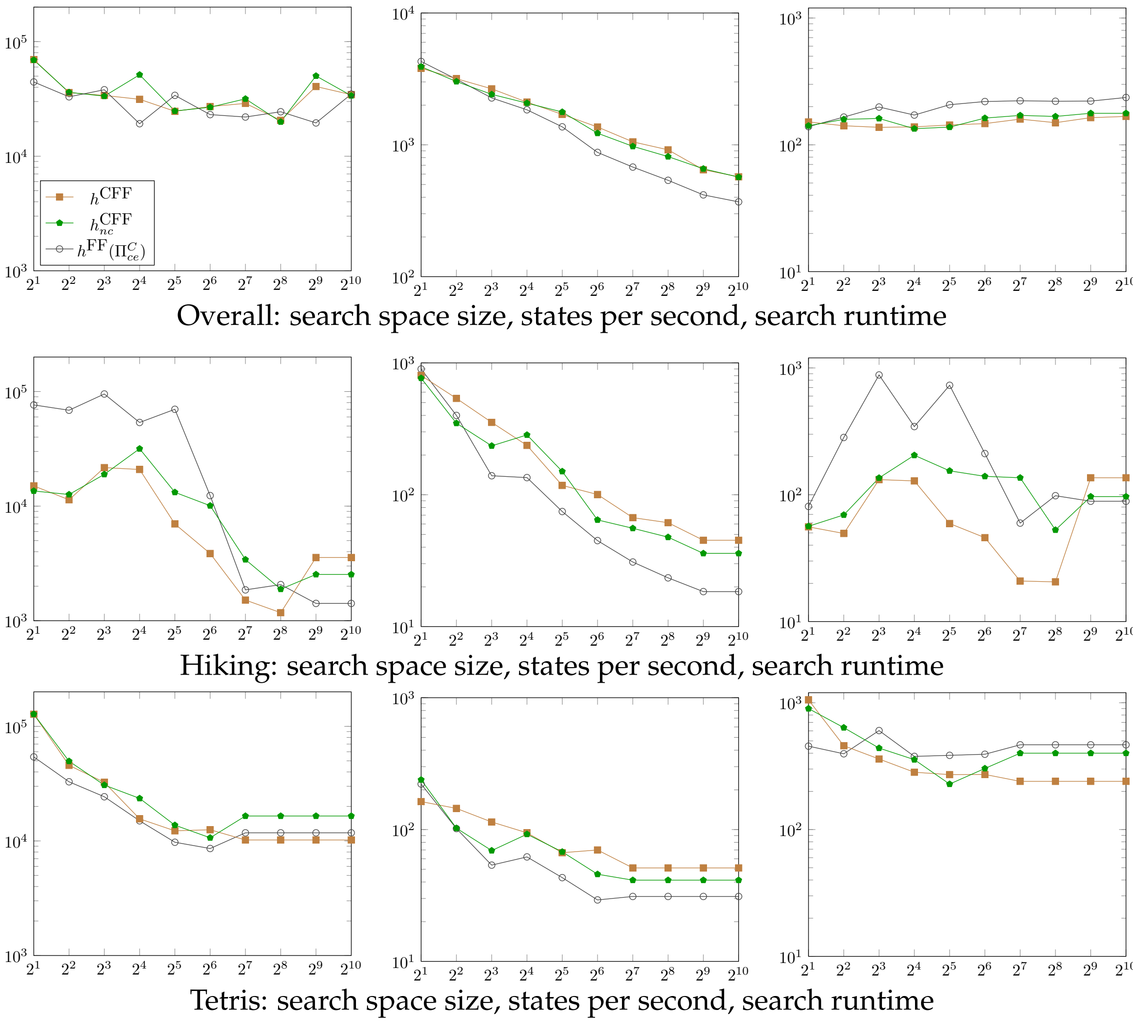
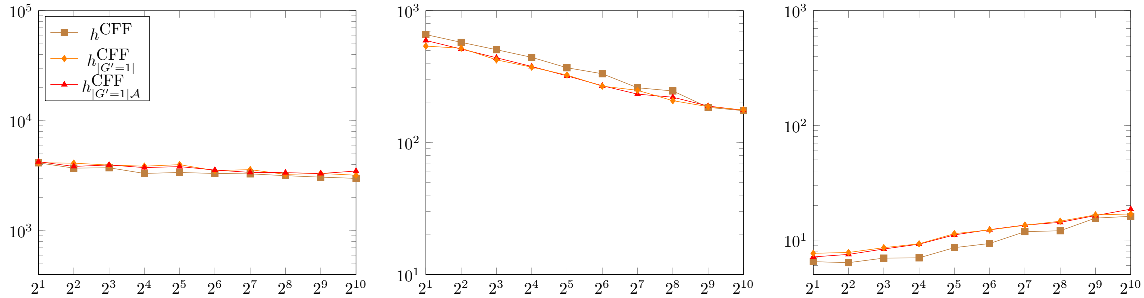


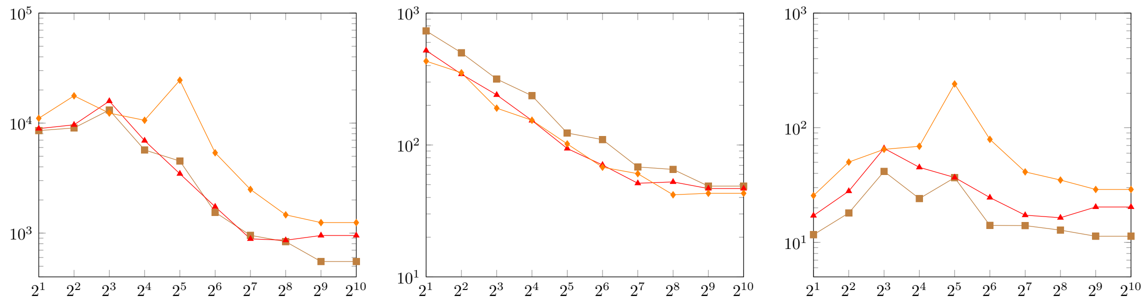
Figure 5: Data for h^{CFF} vs. h^{CFF}_{nc} vs. $h^{\mathrm{FF}}(\Pi^{\mathrm{C}}_{ce})$. Geometric means. All curves use only those instances solved for all values of x by all heuristics (225 instances overall). Note that, for better readability, the y-scales show only 2 orders of magnitude (in difference to Figure 4).

(H2) is weak, but the data does give evidence that cross-context conditions *can*, in some cases, be of advantage.

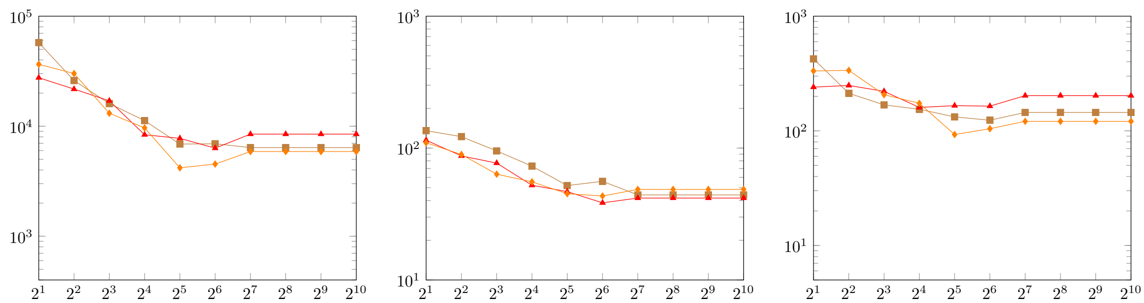
In this context, it is worth coming back briefly to the discussion of Table 1, for $x = 2$, where the comparison between h^{CFF} and h^{CFF}_{nc} was somewhat in favor of h^{CFF}_{nc} (many per-domain advantages, but at a small scale). For growing x, the picture becomes more favorable for h^{CFF}, though still at a small scale. There is no domain where h^{CFF}_{nc} consistently is faster, or of higher quality, or yields better coverage, than h^{CFF}. On the other hand, h^{CFF} is consistently faster in Barman, GED, and Openstacks, plus the favorable behavior in Hiking and Tetris as shown in Figure 5.



Overall: search space size, states per second, search runtime



Hiking: search space size, states per second, search runtime



Tetris: search space size, states per second, search runtime

Figure 6: Data for h^{CFF} vs. $h^{\text{CFF}}_{|G'|=1}$ vs. $h^{\text{CFF}}_{|G'|=1\mathcal{A}}$. Geometric means. All curves use only those instances solved for all values of x by all heuristics (208 instances overall). Note that, for better readability, the y-scales show only 2 orders of magnitude (in difference to Figure 4).

Let us finally consider hypothesis (H4), which asserts that, thanks to its non-trivial subgoal support selection, h^{CFF} typically yields a more informed heuristic than $h^{\text{CFF}}_{|G'|=1}$. We include $h^{\text{CFF}}_{|G'|=1\mathcal{A}}$ into the comparison for completeness. Figure 6 shows the data.

All three heuristics perform very similarly in the overall. This is partly due to particularities of the common instance basis. Several domains are not at all, or hardly, contained in the instance basis of Figure 6. This pertains in particular to Barman, Maintenance, Parcprinter, and Parking, where h^{CFF} has large coverage advantages over $h^{\text{CFF}}_{|G'|=1}$.

Nevertheless, Figure 6 allows to confirm (H4), albeit at a small scale. In the overall, consistently across values of x, h^{CFF} has a slightly smaller search space than $h^{\text{CFF}}_{|G'|=1}$. (It also is slightly faster than $h^{\text{CFF}}_{|G'|=1}$, and consequently results in slightly better runtime.) Per domain, h^{CFF} has search advantages in 9 cases, and disadvantages in only 3 cases. Figure 6 showcases Hiking and Tetris, which respectively represent these domain classes, and where the heuristics benefit from growing x.

In most domains, like in the overall h^{CFF} has a slight speed advantage over $h^{\text{CFF}}_{|G'|=1}$. As we already observed in our discussion of Table 3, these must be caused by the different states evaluated, with h^{CFF} evaluating more states close to the goal and thus being faster.

In terms of coverage, h^{CFF} clearly dominates $h^{\text{CFF}}_{|G'|=1}$ in the overall (Figure 2), and has strong advantages in 8 domains (e. g. Maintenance and Parcprinter, cf. Figure 3), while $h^{\text{CFF}}_{|G'|=1}$ has advantages only in 3 domains (Tetris cf. Figure 3, Thoughtful, and Visitall).

Let us finally compare $h^{\text{CFF}}_{|G'|=1}$ with $h^{\text{CFF}}_{|G'|=1\mathcal{A}}$. Their overall coverage difference is clearly in favor of $h^{\text{CFF}}_{|G'|=1\mathcal{A}}$. Per domain, $h^{\text{CFF}}_{|G'|=1\mathcal{A}}$ has a coverage advantage in 5 domains and a disadvantage in 9, yet the disadvantages are typically marginal whereas the advantages are substantial. Regarding speed and search space size on commonly solved instances, speed is very similar almost universally. Search space size also is often very similar for both (in the overall, $h^{\text{CFF}}_{|G'|=1}$ and $h^{\text{CFF}}_{|G'|=1\mathcal{A}}$ are almost indistinguishable). There are exceptions in individual domains, specifically Elevators, Pegsol, and Visitall where $h^{\text{CFF}}_{|G'|=1}$ is better, and GED, Hiking, and Parking where $h^{\text{CFF}}_{|G'|=1\mathcal{A}}$ is better.

Summing up our observations, the data confirms (H1) impressively, with the caveat that there are only few IPC domains where large C is beneficial. (H3) and (H4) are confirmed consistently, in the overall and across most domains. The evidence for (H2) is weaker, with good cases only in Hiking and Tetris. This is not entirely unexpected given that cross-context conditions occur only in specific situations, important in theory but, evidently, rare in practice as far as reflected by the IPC benchmarks.

6. Contribution Summary and Future Work

Our work contributes a new understanding of recent compilation-based partial delete relaxation heuristics, in terms of a combination of the delete relaxation with critical-path heuristics. The key insight is to view each of these a priori unrelated relaxations as being defined through an underlying set of *atomic subgoals*, where the relaxation consists in decomposing non-atomic conjunctive goals into their atomic subgoals. Critical-path heuristics require to achieve only the most costly atomic subgoal, the delete relaxation requires to achieve all atomic subgoals. The standard delete-relaxation framework now becomes the special case where the atomic subgoals are singleton facts, and the entire standard machinery – h^+, relaxed plan existence testing based on h^1, the additive heuristic h^{add}, relaxed plan extraction from a best-supporter function – extends naturally along the dimension of allowing arbitrary atomic subgoals C instead.

Our direct characterization identifies the precise new source of complexity in the relaxed plan extraction process, namely selecting the subset of atomic subgoals to support with a given action. Thanks to this, we design new C-relaxed plan heuristics, h^{CFF} and h^{CFF}_{nc}, avoiding the shortcomings of previous compilation-based heuristics. The theoretical advantages of h^{CFF} are reflected empirically in IPC benchmarks. The improvement over the state of the art, overall, is marginal though, and relates more to the new heuristics' implementation advantages than to their theoretical ones.

In our view, the main value of this work lies in understanding what the compilation heuristics actually do, spelling out the framework of C-delete relaxation, and replacing $h^{\mathrm{FF}}(\Pi^C)$ and $h^{\mathrm{FF}}(\Pi^C_{ce})$ with the more direct and natural h^{CFF} respectively h^{CFF}_{nc}. While the new heuristics may not yield dramatic benefits in most cases, they are certainly more reliable and somewhat more efficient than their predecessors, and there is no reason not to use them. A nice side benefit is the simple yet useful generalization from h^m to h^C.

We believe that there are still many exciting avenues of future research in this area. We expect that our results will help with many of them, through the more efficient and direct implementation, or through the alternate and less opaque formulation.

An obvious topic is to use backward search instead of forward search, paralleling the design of HSP-r (Bonet & Geffner, 1999) where we need to compute h^C only once on the initial state. Alcázar et al. (2013) already took this direction, but didn't explore it in detail.

There is still a glaring hole in the understanding of C-relaxation heuristics, namely the role of the conjunction set C. *What are good sets C? How to find them?* The literature so far offers preliminary answers to the second question, and offers no answer at all to the first one. In particular, the proof of convergence is via h^m: if we select m large enough then $h^m = h^*$, and if we simulate m via C then $h^C = h^m$, and $h^C \leq h^{C+}$ so we can get $h^{C+} = h^*$ QED. But this completely ignores that (a) we are free to choose *any* set C, not just the size-m conjunctions, and (b) while h^C is a lower bound on h^{C+}, it is a trivial one and h^{C+} typically is *much* higher (similarly as for the well-known relation between h^1 and h^+). So how many/which conjunctions are actually needed to render h^{C+} perfect?

Preliminary results have been obtained with a "bottom-up" approach trying to identify planning fragments where small sets C suffice to obtain $h^{C+} = h^*$ (Hoffmann, Steinmetz, & Haslum, 2014). This approach has proved to be exceedingly difficult though, with complex case considerations already in trivial fragments. Can we instead explore this "top-down", identifying conjunctions *not* needed to render h^{C+} perfect, and thus guide the C-learning mechanisms? In which IPC benchmarks does it suffice to use all fact pairs, and how does h^+ topology (Hoffmann, 2005) change in the other ones? Can we learn something from that about how particular planning sub-structures should be handled?

A more practical approach is the design of alternate C-learning methods. In particular, can we learn C during search? Learning "from mistakes" as has proved extremely successful in constraint-satisfaction problems like SAT (e. g. Marques-Silva & Sakallah, 1999; Moskewicz, Madigan, Zhao, Zhang, & Malik, 2001)? Recent work (Steinmetz & Hoffmann, 2016) has devised an approach doing so for dead-end detection, refining h^C on dead-ends as they become known during the search (i. e., once the search has explored all their descendants). In a depth-first search, this algorithm approaches the elegance of clause learning in SAT, learning generalizing knowledge from refuted search subtrees.

But what about search guidance on non-dead-end states? Can we usefully refine h^{CFF} during search? A difficulty is that, whenever C was increased, to re-adjust the *relative* ordering of states in principle we would need to re-evaluate h^{CFF} on the entire open list. An interesting option is local search: use hill-climbing until a local minimum is reached, then refine C to eliminate that local minimum from h^{CFF}'s search surface. In other words: if caught in a local minimum, rather than giving up on the heuristic and relying on search instead – as is commonly done across AI sub-areas – *refine the heuristic to exhibit the exit*.

In summary, we are now satisfied with the understanding of C-relaxation heuristics, and we believe that the key to fully exploiting their power lies in a better understanding and design of methods finding the atomic subgoals C.

Acknowledgments

This work was partially supported by the German Research Foundation (DFG), under grant HO 2169/5-1. We thank the anonymous reviewers, whose comments helped to improve the paper.

Appendix A. Proofs

Theorem 1 *Let $\Pi = (\mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G})$ be a planning task, and C a set of conjunctions in Π containing all singleton conjunctions. Then $h^+(\Pi^C) = h^{C+}(\Pi)$.*

Proof: Denote $\Pi = (\mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G})$. Our proof is via comparing two equations. *Equation I* simply is Equation 7 (page 280, the h^{C+} equation), characterizing $h^{C+}(\Pi)$. We derive *Equation II* by applying Equation 5 (page 279, our non-standard characterization of h^+) to Π^C, characterizing $h^+(\Pi^C)$.

Repeating Equation 7, for convenience: $h^{C+}(\Pi) = h(\mathcal{G}^C)$, where h is the function on conjunction sets G that satisfies $h(G) =$

$$\begin{cases} 0 & \forall c \in G : c \subseteq \mathcal{I} \\ 1 + \min_{a \in \mathcal{A}, \emptyset \neq G' \subseteq \{c \mid c \in G, R(c,a) \neq \bot\}} h((G \setminus G') \cup {G'_r}^C) & \text{else} \end{cases}$$

with G'_r defined as $G'_r := \bigcup_{c \in G'} R(c, a)$.

Reconsider now Equation 5, which can be written as: $h^+(\Pi) = h(\mathcal{G})$, where h is the function on fact sets G that satisfies $h(G) =$

$$\begin{cases} 0 & G \subseteq \mathcal{I} \\ 1 + \min_{a \in \mathcal{A}, \emptyset \neq G' = \{p \mid p \in G, R(\{p\},a) \neq \bot\}} h((G \setminus G') \cup G'_r) & \text{else} \end{cases}$$

with G'_r defined as $G'_r := \bigcup_{p \in G'} R(\{p\}, a)$.

We now apply the previous equation to Π^C. Making explicit that the individual facts in Π^C all are π-fluents, we obtain: $h^+(\Pi^C) = h(\{\pi_c \mid \pi_c \in \mathcal{G}^{\pi C}\})$, where h is the function on fact sets G that satisfies $h(G) =$

$$\begin{cases} 0 & \forall \pi_c \in G : \pi_c \in \mathcal{I}^{\pi C} \\ 1 + \min_{a[C'] \in \mathcal{A}^{\pi C}, \emptyset \neq G' = \{\pi_c \mid \pi_c \in G, R(\{\pi_c\}, a[C']) \neq \bot\}} h((G \setminus G') \cup G'_r) & \text{else} \end{cases}$$

with G'_r defined as $G'_r := \bigcup_{\pi_c \in G'} R(\{\pi_c\}, a[C'])$.

One can already see the correspondence here to Equation I, with conjunctions c corresponding to π-fluents π_c. The only major difference is the set of action/supported-subgoal-set pairs minimized over in the bottom cases.

Consider the set $G' = \{\pi_c \mid \pi_c \in G, R(\{\pi_c\}, a[C']) \neq \bot\}$ of supported atomic subgoals as per the last equation. The condition $R(\{\pi_c\}, a[C']) \neq \bot$ simplifies to $c \in C'$, because these are exactly the π-fluents added by $a[C']$. Thus, removing the G' variable which is fixed anyway, the equation simplifies to: $h(G) =$

$$\begin{cases} 0 & \forall \pi_c \in G : \pi_c \in \mathcal{I}^{\pi C} \\ 1 + \min_{a[C'] \in \mathcal{A}^{\pi C}, \emptyset \neq \{\pi_c \mid \pi_c \in G, c \in C'\}} h((G \setminus G') \cup G'_r) & \text{else} \end{cases}$$

Now what we minimize over are the actions $a[C']$ in Π^C, where C' needs to support a non-empty subset of subgoal π-fluents/conjunctions π_c. What are the possible choices of C'? The c we can in principle include into C', i. e. the subgoals that we can in principle support using the action a are, by the definition of Π^C, exactly those where $R(c, a) \neq \perp$. Observe that there is no point in including c where $\pi_c \notin G$: This will support the same subgoals yet can only result in a larger precondition. Hence the choice of C' is exactly $\emptyset \neq C' \subseteq \{c \mid \pi_c \in G, R(c, a) \neq \perp\}$. Renaming C' into G' in order to unify notation with Equation I, this yields our final Equation II: $h^+(\Pi^C) = h(\{\pi_c \mid \pi_c \in \mathcal{G}^{\pi C}\})$, where h is the function on fact sets G that satisfies $h(G) =$

$$\begin{cases} 0 & \forall \pi_c \in G : \pi_c \in \mathcal{I}^{\pi C} \\ 1 + \min_{a[G'] \in \mathcal{A}^{\pi C}, \emptyset \neq G' \subseteq \{c \mid \pi_c \in G, R(c,a) \neq \perp\}} h((G \setminus G') \cup G'_r) & \text{else} \end{cases}$$

with G'_r defined as $G'_r := \bigcup_{c \in G'} R(\{\pi_c\}, a[G'])$.

To spell out the correspondence between Equations I and II, view each of them as a tree whose root node is the "initializing call" on the respective input task's goal. Then the two trees are isomorphic in the sense that there is a one-to-one mapping between tree nodes, and, using the suffixes $[I]$ and $[II]$ to identify the respective tree, at any pair $G[I]$ and $G[II]$ of corresponding tree nodes we have:

$$(*)\ G[II] = \{\pi_c \mid c \in G[I]\}$$

This is true by definition for the root nodes (I) $h^{C+}(\Pi) = h(\mathcal{G}^C)$ respectively (II) $h^+(\Pi^C) = h(\{\pi_c \mid \pi_c \in \mathcal{G}^{\pi C}\})$.

Consider corresponding bottom-case nodes with (*). Observe that the choice of atomic subgoals c for G' is the same on both sides: Equation I allows those $c \in G[I]$ where $R(c, a) \neq \perp$, Equation II allows those $\pi_c \in G[II]$ where $R(c, a) \neq \perp$.

We map children nodes using the same action a and the same supported subgoal set $G'[I] = G'[II] =: G'$ on both sides. We use action a in Equation I and action $a[G']$ in Equation II. Consider the recursive subgoals, $(G[I] \setminus G') \cup [\bigcup_{c \in G'} R(c, a)]^C$ in Equation I, and $(G[II] \setminus \{\pi_c \mid c \in G'\}) \cup \bigcup_{c \in G'} R(\{\pi_c\}, a[G'])$ in Equation II.

The $G \setminus G'$ parts of these expressions are in exact match by (*) and construction, so it remains to consider $[\bigcup_{c \in G'} R(c, a)]^C$ vs. $\bigcup_{c \in G'} R(\{\pi_c\}, a[G'])$. As regression over singleton fact sets just yields the action precondition, $\bigcup_{c \in G'} R(\{\pi_c\}, a[G'])$ simplifies to $pre(a[G'])$. By the definition of Π^C, this equals $[\bigcup_{c \in G'}(pre(a) \cup (c \setminus add(a)))]^{\pi C}$. As $R(c, a) = pre(a) \cup (c \setminus add(a))$, this equals $[\bigcup_{c \in G'} R(c, a)]^{\pi C}$. The desired match with $[\bigcup_{c \in G'} R(c, a)]^C$ is now obvious, showing that (*) is preserved, which concludes our argument. ■

Theorem 2 *Let* $\Pi = (\mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G})$ *be a planning task, and* C *a set of conjunctions in* Π *containing all singleton conjunctions. Then* $h^+(\Pi^C_{nc}) = h^{C+}_{nc}(\Pi)$.

Proof: The proof is very similar to that of Theorem 1. Equation I is the same, except that $G'_r := \{R(c,a) \mid c \in G'\}$ as per the definition of h^{C+}_{nc}. Equation II is exactly the same, the only difference being that $G'_r := \bigcup_{c \in G'} R(\{\pi_c\}, a[G'])$ now is interpreted as per the definition of Π^C_{nc}, as opposed to that of Π^C. The arguments then are exactly the same, except for the last part of the proof showing the correspondence of $\{R(c,a) \mid c \in G'\}^C$ vs. $\bigcup_{c \in G'} R(\{\pi_c\}, a[G'])$. As regression over singleton fact sets just yields the action precondition, $\bigcup_{c \in G'} R(\{\pi_c\}, a[G'])$ simplifies to $pre(a[G'])$. By the definition of Π^C_{nc}, this equals $\{pre(a) \cup (c \setminus add(a)) \mid c \in G'\}^{\pi C}$. As $R(c,a) = pre(a) \cup (c \setminus add(a))$, this equals $\{R(c,a) \mid c \in G'\}^{\pi C}$, matching $\{R(c,a) \mid c \in G'\}^C$ as desired. ■

Theorem 3 *Let $\Pi = (\mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G})$ be a planning task, and C a set of conjunctions in Π containing all singleton conjunctions. Then $h^1(\Pi^C) = h^1(\Pi^C_{nc}) = h^C(\Pi)$.*

Proof: Denote $\Pi = (\mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G})$. Consider first Π^C. We compare respective characterizing equations. First, Equation 6 (page 279) characterizes h^1; applying it to Π^C, we get that $h^1(\Pi^C) = h(\mathcal{G}^{\pi C})$ where h is the function on fact sets G that satisfies

$$h(G) = \begin{cases} 0 & G \subseteq \mathcal{I}^{\pi C} \\ 1 + \min_{a[C'] \in \mathcal{A}^{\pi C}, R(G, a[C']) \neq \bot} h(R(G, a[C'])) & G = \{\pi_c\}, c \in C \\ \max_{\pi_c \in G} h(\{\pi_c\}) & \text{else} \end{cases}$$

Observe that, in the middle case, we must have $c \in C'$ because otherwise $\pi_c \notin add(a[C'])$; and that there is no point in including any other conjunctions into C', i. e., $C' \supsetneq \{c\}$, because this can only yield a larger recursive subgoal $R(G, a[C'])$. Hence we can re-write the previous equation to:

$$h(G) = \begin{cases} 0 & G \subseteq \mathcal{I}^{\pi C} \\ 1 + \min_{a[\{c\}] \in \mathcal{A}^{\pi C}, R(G, a[\{c\}]) \neq \bot} h(R(G, a[\{c\}])) & G = \{\pi_c\}, c \in C \\ \max_{\pi_c \in G} h(\{\pi_c\}) & \text{else} \end{cases}$$

Refer to this as *Equation I*.

Recall that $h^C(\Pi) = h(\mathcal{G})$ where h is the function on fact sets G that satisfies

$$h(G) = \begin{cases} 0 & G \subseteq \mathcal{I} \\ 1 + \min_{a \in \mathcal{A}, R(G,a) \neq \bot} h(R(G,a)) & G \in C \\ \max_{G' \subseteq G, G' \in C} h(G') & \text{else} \end{cases}$$

Refer to this as *Equation II*.

Similarly as in the proof of Theorem 1, view each of these equations as a tree whose root node is the "initializing call" (I) $h^1(\Pi^C) = h(\mathcal{G}^{\pi C})$ respectively (II) $h^C(\Pi) = h(\mathcal{G})$. Then the two trees are isomorphic in the sense that there is a one-to-one mapping between tree nodes, and, using the suffixes [I] and [II] to identify the respective tree, at any pair $G[I]$ and $G[II]$ of corresponding tree nodes we have:

$$(*)\ G[I] = \{\pi_c \mid c \in C, c \subseteq G[II]\}$$

This is obviously true for the root nodes, and is obviously invariant over the bottom case where we can map the children node pairs corresponding to the same $\pi_c \in G[I]$ respectively $c \subseteq G[II]$.

Consider now corresponding middle-case nodes where $G[I] = \{\pi_c\}$ and $G[II] = c$. First, the Π^C actions $a[\{c\}]$ all by definition satisfy $R(G[I], a[\{c\}]) = R(\{\pi_c\}, a[\{c\}]) \neq \bot$. The choice of $a[\{c\}]$ thus corresponds to the choice of actions a from the original task Π for which an action $a[\{c\}]$ is included into Π^C. These are exactly the actions a over which c can be regressed, $R(c, a) \neq \bot$, and hence those where $R(G[II], a) = R(c, a) \neq \bot$. So the choice of actions minimized over is the same on both sides, and we can map the children node pairs corresponding to the same a.

For any such pair, the recursive subgoal $G_r[II]$ generated in (II) is $R(c, a) = (c \setminus add(a)) \cup pre(a)$. The recursive subgoal $G_r[I]$ generated in (I) is $R(\{\pi_c\}, a[\{c\}]) = pre(a[\{c\}])$, which by the definition of Π^C equals $[(c \setminus add(a)) \cup pre(a)]^{\pi C}$. The latter is defined as $\{\pi_{c'} \mid c' \in C, c' \subseteq (c \setminus add(a)) \cup pre(a)\}$. This equals $\{\pi_{c'} \mid c' \in C, c' \subseteq G_r[II]$, showing (*) and concluding our argument.

The argument for Π^C_{nc} is identical because, for single-conjunction sets $C' = \{c\}$, the two compilations coincide (specifically, the precondition of $a[\{c\}]$ is the same in Π^C and Π^C_{nc}). ∎

Theorem 4 *Let $\Pi = (\mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G})$ be a planning task, and C a set of conjunctions in Π containing all singleton conjunctions. Then any C-relaxed plan π^{CFF} can be sequentialized to form a relaxed plan for Π^C.*

Proof: Denote $\Pi = (\mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G})$. Recall the definition of π^{CFF}, as $\pi^{\mathrm{CFF}} = \pi(\mathcal{G}^C)$, with π being a partial function on conjunction sets G that is defined on $\mathcal{G}^C$ and satisfies $\pi(G) =$

$$\begin{cases} \emptyset & \forall c \in G : c \subseteq \mathcal{I} \\ \pi((G \setminus G') \cup {G'_r}^C) \cup \{(a, G')\} \text{ where } a \in \mathcal{A}, & \\ \quad \emptyset \neq G' \subseteq \{c \mid c \in G, R(c, a) \neq \bot, h^C(R(c, a)) = h^C(c) - 1\}, & \\ \quad \text{and } h^C(G'_r) = h^C(G') - 1 & \text{else} \end{cases}$$

with G'_r defined as $G'_r := \bigcup_{c \in G'} R(c, a)$.

Starting at $\pi(\mathcal{G}^C)$, say we keep tracing the recursive invocations of the equation, using a suitable (a, G') choice in π^{CFF} whenever the bottom case of the equation applies. By construction (because $\pi^{\mathrm{CFF}} = \pi(\mathcal{G}^C)$), we can make these choices in a way so that we eventually reach the top case, where we terminate. Denote by $\langle (a_0, G'_0), \ldots, (a_{n-1}, G'_{n-1}) \rangle$ the inverted sequence of action occurrences selected along our trace, i. e., deeper recursion steps correspond to smaller indices, (a_0, G'_0) is the action occurrence whose selection lead to the terminating top case, and (a_{n-1}, G'_{n-1}) is the action occurrence selected in the initializing call. We show that $\langle a_0[G'_0], \ldots, a_{n-1}[G'_{n-1}] \rangle$ is a relaxed plan for Π^C.

Denote by G_i, for $1 \leq i \leq n$, the subgoal tackled by the selection of (a_{i-1}, G'_{i-1}) in the middle case, and denote by G_0 the final subgoal tackled by the top case. Denote by s_i the state resulting from applying $\langle a_0[G'_0], \ldots, a_{i-1}[G'_{i-1}] \rangle$ in Π^C. We show by induction over i that *(*)* $\{\pi_c \mid c \in G_i\} \subseteq s_i$. For $i = n$, where $G_n = \mathcal{G}^C$, this shows that $s_n \supseteq \mathcal{G}^{\pi C}$ as desired.

Induction base case, $i = 0$: Here, (*) follows directly from definition because, the top case having fired on G_0, for all $c \in G_0$ we have that $c \subseteq \mathcal{I}$, and hence $\pi_c \in \mathcal{I}^{\pi C} = s_0$.

For the induction step, assume that (*) is true up to i. We show that it holds for $i + 1$. By construction, G_i is the recursive subgoal $(G_{i+1} \setminus G'_i) \cup [\bigcup_{c \in G'_i} R(c, a_i)]^C$. Denote the left half of this expression by $LH := G_{i+1} \setminus G'_i$, and the right half by $RH := [\bigcup_{c \in G'_i} R(c, a_i)]^C$.

By induction hypothesis, we have $(*)$ $\{\pi_c \mid c \in LH \cup RH\} \subseteq s_i$. Consider now G_{i+1} and s_{i+1}. First, those atomic subgoals not achieved by (a_i, G'_i), namely $G_{i+1} \setminus G'_i$, are tackled by LH: By $(*)$ $\{\pi_c \mid c \in LH\} = \{\pi_c \mid c \in G_{i+1} \setminus G'_i\} \subseteq s_i$. As the planning is delete-free this immediately yields $\{\pi_c \mid c \in G_{i+1} \setminus G'_i\} \subseteq s_{i+1}$. Second, those atomic subgoals that *are* achieved by (a_i, G'_i), namely G'_i, clearly will be true in s_{i+1} as well: This is simply because $add(a[G'_i]) = \{\pi_c \mid c \in G'_i\}$.

It remains to show that $a_i[G'_i]$ is applicable in s_i. By the definition of Π^C, its precondition is $[\bigcup_{c \in G'_i}(pre(a) \cup (c \setminus add(a)))]^{\pi C}$. As $R(c,a) = pre(a) \cup (c \setminus add(a))$, this equals $[\bigcup_{c \in G'_i} R(c,a)]^{\pi C}$. The latter is exactly $pre(a_i[G'_i]) = \{\pi_c \mid c \in RH\}$, so we are done by $(*)$ $\{\pi_c \mid c \in RH\} \subseteq s_i$ which concludes the proof. ∎

Theorem 5 *Let $\Pi = (\mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G})$ be a planning task, and C a set of conjunctions in Π containing all singleton conjunctions. Then any nc-C-relaxed plan π^{CFF}_{nc} can be sequentialized to form a relaxed plan for Π^C_{nc}.*

Proof: Denote $\Pi = (\mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G})$. Recall the definition of π^{CFF}_{nc}, as $\pi^{\mathrm{CFF}} = \pi(\mathcal{G}^C)$, with π being a partial function on conjunction sets G that is defined on $\mathcal{G}^C$ and satisfies $\pi(G) =$

$$\begin{cases} \emptyset & \forall c \in G : c \subseteq \mathcal{I} \\ \pi((G \setminus G') \cup {G'_r}^C) \cup \{(a, G')\} \text{ where } a \in \mathcal{A}, & \\ \quad \emptyset \neq G' \subseteq \{c \mid c \in G, R(c,a) \neq \bot, h^C(R(c,a)) = h^C(c) - 1\}, & \\ \quad \text{and } h^C(G'_r) = h^C(G') - 1 & \text{else} \end{cases}$$

with G'_r defined as $G'_r := \{R(c,a) \mid c \in G'\}$.

The proof of Theorem 4 remains valid exactly as written, except that now $RH = \{R(c,a) \mid c \in G'_i\}^C$. We need to show that $a_i[G'_i]$ is applicable in s_i. By the definition of Π^C_{nc}, its precondition is $\{pre(a) \cup (c \setminus add(a)) \mid c \in G'_i\}^{\pi C}$. As $R(c,a) = pre(a) \cup (c \setminus add(a))$, this equals $\{R(c,a) \mid c \in G'_i\}^{\pi C}$. The latter is exactly $pre(a_i[G'_i]) = \{\pi_c \mid c \in RH\}$, so again we are done by $(*)$ $\{\pi_c \mid c \in RH\} \subseteq s_i$. ∎

Theorem 6 *Let $\Pi = (\mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G})$ be a planning task, and C a set of conjunctions in Π containing all singleton conjunctions. Then a C-relaxed plan exists if and only if an nc-C-relaxed plan exists if and only if $h^C < \infty$.*

Proof: Denote $\Pi = (\mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G})$. We show the claim in two parts, (a) a C-relaxed plan exists if and only if $h^C < \infty$, and (b) an nc-C-relaxed plan exists if and only if $h^C < \infty$. We consider first part (a).

The "only if" direction is a corollary of Theorems 3 and 4: If $h^C = \infty$, then by Theorem 3 $h^1(\Pi^C) = \infty$ so a relaxed plan for Π^C does not exist, which by Theorem 4 implies that a C-relaxed plan cannot exist either.

For the "if" direction, say that $h^C < \infty$. We need to show that there exists a C-relaxed plan. To this end, we consider a simpler version of the equation defining π^{CFF} (Equation 11), restricting the choice of G' to singletons $G' = \{c\}$. After easy simplifications, we get: $\pi(G) =$

$$\begin{cases} \emptyset & \forall c \in G : c \subseteq \mathcal{I} \\ \pi((G \setminus \{c\}) \cup R(c,a)^C) \cup \{(a, \{c\})\} \text{ where } a \in \mathcal{A}, & \\ \quad c \in G, R(c,a) \neq \bot, \text{and } h^C(R(c,a)) = h^C(c) - 1 & \text{else} \end{cases}$$

Recall, here and in all equations below, that the function π is partial, which defines a C-relaxed plan only if it is defined on (the atomic conjunctions of) the global goal $\mathcal{G}^C$.

Observe that, in the previous equation, as we always support only a single atomic subgoal anyhow, there is no need to recurse over sets of atomic subgoals. We can instead recurse over single atomic subgoals, and replace the initializing and recursive calls, now over sets of atomic subgoals, by the union over a call to each of their elements. This results in the characterization given by Equation 12: $\pi^{\mathrm{CFF}} = \bigcup_{c \in \mathcal{G}^C} \pi(c)$, with π being a partial function on conjunctions c that satisfies $\pi(c) =$

$$\begin{cases} \emptyset & c \subseteq \mathcal{I} \\ \bigcup_{c' \in R(c,a)^C} \pi(c') \cup \{(a, \{c\})\} \text{ where } a \in \mathcal{A}, & \\ \quad R(c,a) \neq \perp, \text{and } h^C(R(c,a)) = h^C(c) - 1 & \text{else} \end{cases}$$

Note the similarity to Equation 8 (page 285): We are now back to a more common notation for relaxed plan extraction (over C instead of singleton facts), extracting best supporters one-by-one.

Towards proving our claim, we now transform the equation in a way making the link to h^C obvious. Instead of the union operations in the initial and recursive calls, which enumerate all atomic subgoals contained in a given set of facts ($\mathcal{G}$ respectively $R(c,a)$), we can recurse directly over these fact sets, G, and introduce a third case performing the union over all atomic conjunctions contained in G. We hence get the characterization given by Equation 13: $\pi^{\mathrm{CFF}} = \pi(\mathcal{G})$, with π being a partial function on fact sets G that satisfies $\pi(G) =$

$$\begin{cases} \emptyset & G \subseteq \mathcal{I} \\ \pi(R(G,a)) \cup \{(a, G)\} \text{ where } a \in \mathcal{A}, & \\ \quad R(G,a) \neq \perp, \text{ and } h^C(R(G,a)) = h^C(G) - 1 & G \in C \\ \bigcup_{G' \subseteq G, G' \in C} \pi(G') & \text{else} \end{cases}$$

Compare this to Equation 3 defining h^C: $h(G) =$

$$\begin{cases} 0 & G \subseteq \mathcal{I} \\ 1 + \min_{a \in \mathcal{A}, R(G,a) \neq \perp} h(R(G,a)) & G \in C \\ \max_{G' \subseteq G, G' \in C} h(G') & \text{else} \end{cases}$$

The bottom cases in both equations are in obvious correspondence. On G with $h(G) < \infty$, the middle cases are in correspondence, too, in the sense that the choice of action occurrences in Equation 13 is exactly the choice of minimizing actions in Equation 3: if $h^C(G) < \infty$, then the actions minimizing $1 + h^C(R(G,a))$ are those where $h^C(R(G,a)) = h^C(G) - 1$. So, on finite-value subgoals, the subgoaling structure of the two equations coincides, and in particular, if $h^C < \infty$, then there exists a solution π to Equation 13 such that π is defined on $\mathcal{G}$. Therefore, Equation 12 has a solution defined on all $c \in \mathcal{G}^C$. As Equation 12 captures a restricted version of π^{CFF}, applying the same conditions to a smaller choice of action occurrences, this implies that there exists a C-relaxed plan as desired, concluding part (a) of the proof.

For part (b), the "only if" direction follows in the same manner as a corollary of Theorems 3 and 5. The "if" direction also follows in the same manner, because the only difference lies in the definition of G'_r, but for singleton G' that difference disappears: for $G' = \{c\}$

we have $G'_r = \{R(c,a)\}$ in π^{CFF}_{nc}, vs. $G'_r = R(c,a)$ in π^{CFF}. The atomic conjunctions contained in these expressions are the same. Hence, restricting the choice of G' to singletons, the equation defining π^{CFF}_{nc} simplifies to Equation 12 exactly as above, and from there the proof is identical. ■

Example 9 *We construct an example where it is of advantage to select a smaller set of supported subgoals G', even though a larger set – a strict superset – would be feasible. The construction extends our abstract example (Example 3).*

Consider the planning task $\Pi = (\mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G})$ defined as follows. $\mathcal{F} = \{g_1, g_2, p, q_1, q_2, r_1, r_2, r'_1, r'_2, {q'_1}^1, \dots, {q'_1}^N, {q'_2}^1, \dots, {q'_2}^N\}$, $\mathcal{I} = \{q_1, q_2\}$, $\mathcal{G} = \{g_1, g_2\}$. $\mathcal{A}$ consists of: (abbreviating each action a in the form a : precondition facts $\rightarrow$ positive and negative effect literals)

- *Achieving the goals. $a[g_1]$: $p, q_1 \rightarrow g_1$. $a[g_2]$: $p, q_2 \rightarrow g_2$.*
- *Achieving p. $a_1[p]$: $r_1 \rightarrow p$. $a_2[p]$: $r_2 \rightarrow p$.*
- *Achieving preconditions for p. $a[r_1]$: $r'_1 \rightarrow r_1, \neg q_2$. $a[r_2]$: $r'_2 \rightarrow r_2, \neg q_1$.*
- *Achieving preconditions for r_i. $a[r'_1]$: $\rightarrow r'_1$. $a[r'_2]$: $\rightarrow r'_2$.*
- *Reachieving q_1. $a[r_2, q_1]$: ${q'_1}^1, \dots, {q'_1}^N \rightarrow r_2, q_1$. For $1 \leq i \leq N$: $a[{q'_1}^i]$: $\rightarrow {q'_1}^i$.*
- *Reachieving q_2. $a[r_1, q_2]$: ${q'_2}^1, \dots, {q'_2}^N \rightarrow r_1, q_2$. For $1 \leq i \leq N$: $a[{q'_2}^i]$: $\rightarrow {q'_2}^i$.*

In this construction, achieving $r_1 \wedge q_1$ takes 2 steps, while achieving $r_1 \wedge q_2$ takes $N+1$ steps; and symmetrically, achieving $r_2 \wedge q_2$ takes 2 steps, while achieving $r_2 \wedge q_1$ takes $N+1$ steps. We now use these properties to construct a case where any smallest-possible nc-C-relaxed plan π^{CFF}_{nc} must use $a_1[p]$ to achieve p for g_1, and use $a_2[p]$ to achieve p for g_2, thus relying on non-maximal sets of best-supported subgoals during relaxed plan extraction. The same arguments apply to C-relaxed plans π^{CFF} which, in this example, behave identically.

Say that C contains the singleton conjunctions as well as $c_{pq1} = p \wedge q_1$, $c_{pq2} = p \wedge q_2$, $c_{r1q2} = r_1 \wedge q_2$, and $c_{r2q1} = r_2 \wedge q_1$.

Constructing an nc-C-relaxed plan according to Definition 3 (page 287), we start by $\pi^{\mathrm{CFF}}_{nc} = \pi(\{g_1, g_2\})$ requiring to support each of the two (atomic-singleton-conjunction) goal facts. This can be done only by $(a[g_1], \{g_1\})$ and $(a[g_2], \{g_2\})$ respectively. After using these in the bottom case of Equation 11, we get the recursive subgoal $G = \{c_{pq1}, c_{pq2}\}$ (as well as the subsumed conjunctions p, q_1, q_2 which are irrelevant to the following discussion). Each of $a_1[p]$ or $a_2[p]$ can support each of these conjunctions. Indeed, each of them is a best supporter for each of these conjunctions.

To see this, observe first that we have $h^C(c_{pq1}) = 3$ e.g. via $a[r'_1]$, $a[r_1]$, $a_1[p]$; and $h^C(c_{pq2}) = 3$ e.g. via $a[r'_2]$, $a[r_2]$, $a_2[p]$; as is clear from this already, $a_i[p]$ is a best supporter for c_{pqi}. Regarding the cross-over combinations, $R(c_{pq2}, a_1[p]) = c_{r1q2}$; as $a[r_1]$ deletes q_2, this can only be regressed via $a[r_1, q_2]$, leading to the subgoal $\{{q'_2}^1, \dots, {q'_2}^N\}$ whose h^C value clearly is 1, so $h^C(R(c_{pq2}, a_1[p])) = 2 = h^C(c_{pq2}) - 1$ as desired. Similarly, $R(c_{pq1}, a_2[p]) = c_{r2q1}$ whose h^C value is 2 as desired.

So, at the subgoal $G = \{c_{pq1}, c_{pq2}\}$, we can choose among six action occurrences, using either of $a_1[p]$ or $a_2[p]$ to support either of $G'_{12} := \{c_{pq1}, c_{pq2}\}$, $G'_1 := \{c_{pq1}\}$, or $G'_2 := \{c_{pq2}\}$.

Now, while the cross-over combinations are suitable as far as h^C is concerned, they are not *suitable to obtain shortest relaxed plans. Say we include c_{pq2} into the supported subgoal set for $a_1[p]$. Then the regressed subgoal is c_{r1q2}, requiring us to use $a[r_1, q_2]$ as well as the N actions $a[q_2'^{\,i}]$, so $N + 1$ actions in total. On the other hand, using $a_1[p]$ to support c_{pq1}, the regressed subgoal is $\{r_1, q_1\}$ – two singleton conjunctions – which can be supported using the action occurrences $(a[r_1'], \{r_1'\})$, $(a[r_1], \{r_1\})$ (recall here that q_1 is true initially). Similarly, using $a_2[p]$ to support c_{pq1} incurs cost $N + 1$ while using $a_2[p]$ to support c_{pq2} incurs cost 2.*

Getting back to our choice at the subgoal $G = \{c_{pq1}, c_{pq2}\}$, if we use $(a_1[p], G_1')$, we can thereafter use $(a_2[p], G_2')$ and get an nc-C-relaxed plan of cost 8: 2 for previously achieving the facts g_i, 2 for these two occurrences achieving c_{pqi}, 4 for afterwards achieving the facts r_i. Similarly if we use $(a_2[p], G_2')$ first. If, however, we start with any other action occurrence, then we incur cost $N + 1$ for at least one of the c_{pqi}, exceeding the optimal cost 8 for sufficiently large N. In particular, while the action occurrence $(a_1[p], G_{12}')$ is feasible, and supports a strict superset of the atomic subgoals supported by $(a_1[p], G_1')$, it leads to a strictly larger relaxed plan.

Theorem 7 *C-SubgoalSupport is **NP**-complete.*

Proof: Membership is obvious by guess and check. For hardness, we show a polynomial reduction from the Hitting Set problem with a set B of subsets $b \subseteq E$ of a finite set of elements E, the question being whether there exists a hitting set of size at most L.

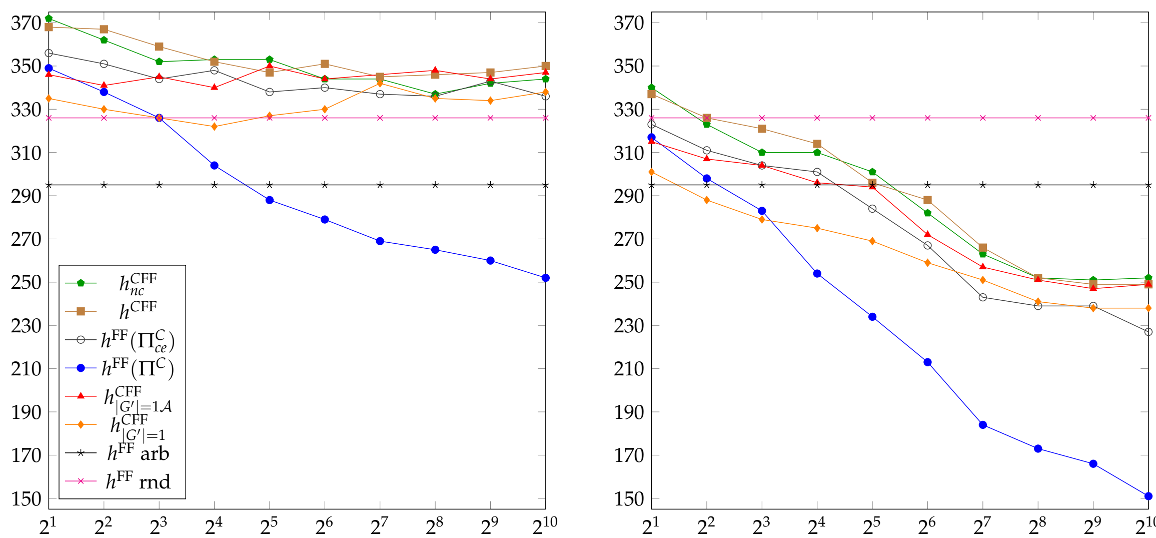
Denote $E = \{e_1, \dots, e_n\}$. We construct a planning task $\Pi = (\mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G})$ as follows. $\mathcal{F} := E \cup \{p_0, p_1, p_2\} \cup \{g_1, \dots, g_n\}$, $\mathcal{I} := \{p_0\}$, $\mathcal{G} := \{g_1, \dots, g_n\}$. The action set $\mathcal{A}$ contains $a[p_1]$ with precondition p_0, add p_1, and empty delete, as well as $a[p_2]$ with precondition p_1, add p_2, and empty delete. The action set furthermore contains an action $a[e_i]$ for every $e_i \in E$, with $pre(a[e_i]) = \{p_0\}$, $add(a[e_i]) = \{e_i\}$, and $del(a[e_i]) = \{p_2\} \cup \{e_j \mid \text{ex. } b \in B : \{e_i, e_j\} \subseteq b\}$. Finally, the action set contains the actions $a[g_1], \dots, a[g_n]$ where $pre(a[g_i]) = \{e_i, p_2\}$, $add(a[g_i]) = \{g_i\}$, and the delete is empty. We set $C := \{\{p\} \mid p \in \mathcal{F}\} \cup B \cup \{\{e_i, p_2\} \mid e_i \in E\}$.

We think of h^C now in terms of a (C-)relaxed planning graph (RPG), where layer t corresponds to the conjunctions g with $h^C(g) \leq t$. None of the conjunctions $b \in B$ can be achieved, as there exists no action through which b can be regressed. However, all the facts $e_i \in E$ can be achieved in isolation. Consider layer 1 of the RPG. The key property we exploit below is that *(*) any subset $E' = \{e_1, \dots, e_k\} \subseteq E$ is feasible at layer 1, i. e. $h^C(E') \leq 1$, iff there does not exist $b \in B$ s.t. $b \subseteq E'$.* From right to left, if $b \subseteq E'$ then E' is infeasible simply because $h^C(b) = \infty$. Vice versa, say there does not exist $b \in B$ s.t. $b \subseteq E'$. Then $c' \subseteq E', c' \in C$ is just the set of singleton conjunctions $\{e_i\}$, and we get $h^C(E') = 1$ as each $\{e_i\}$ is achieved by a single action.

At RPG layer 1, we can apply $a[p_2]$. As each e_i is already present, we get each of the conjunctions $\{e_1, p_2\}, \dots, \{e_n, p_2\}$ at layer 2. With this, the $a[g_i]$ actions become feasible, so that the goal is reached at layer 3.

Consider now relaxed plan extraction. To get the goal, we must select all $a[g_i]$ actions. Say all those are selected in sequence. Then we get the subgoal $\{\{e_1, p_2\}, \dots, \{e_n, p_2\}\}$ at layer 2 (plus the subsumed singleton conjunctions, which we omit for readability). The only action through which these can be regressed is $a[p_2]$: recall that the $a[e_i]$ actions delete p_2. But what is the maximal subset $G' := \{\{e_{i_1}, p_2\}, \dots, \{e_{i_k}, p_2\}\} \subseteq \{\{e_1, p_2\}, \dots, \{e_n, p_2\}\}$ that we can choose to support?

Any such subset yields the new generated subgoal $\{e_{i_1}, \ldots, e_{i_k}, p_1\}$ at RPG layer 1. Here p_1 is achieved by $a[p_1]$ which does not interact with anything so is not critical: $h^C(\{e_{i_1}, \ldots, e_{i_k}, p_1\}) = h^C(\{e_{i_1}, \ldots, e_{i_k}\})$. Denote $E' := \{e_{i_1}, \ldots, e_{i_k}\}$. Then the action occurrence $(a[p_2], G')$ is C-feasible at RPG layer 1 iff E' is feasible at RPG layer 1. By (*), the latter is the case iff there does not exist $b \in B$ s.t. $b \subseteq E'$. But then, consider $E \setminus E'$. By construction, this is a hitting set iff E' is feasible: if $E \setminus E'$ is a hitting set then no b can be fully contained in E', and if no b is fully contained in E' then $E \setminus E'$ must hit every b. Setting $K := n - L$, we thus get that there exists a C-feasible G' with $|G'| \geq K$ iff there exists a feasible E' with $|E'| \geq n - L$ iff there exists a hitting set of size $\leq n - (n - L) = L$. This concludes the proof. ■



(a) Search-only coverage (b) Inclusive coverage

Figure 7: Total coverage.

Appendix B. Coverage when Including C-Learning into the Time Limit

We give the same coverage plots as in Section 5.4, but imposing a 30-minute limit on C-learning and search together ("inclusive" in Figures 7, 8, and 9). For convenience, we also include the search-only plots from Section 5.4.

References

Alcázar, V., Borrajo, D., Fernández, S., & Fuentetaja, R. (2013). Revisiting regression in planning. In Rossi, F. (Ed.), *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI'13)*, pp. 2254–2260. AAAI Press/IJCAI.

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. AAAI Press.

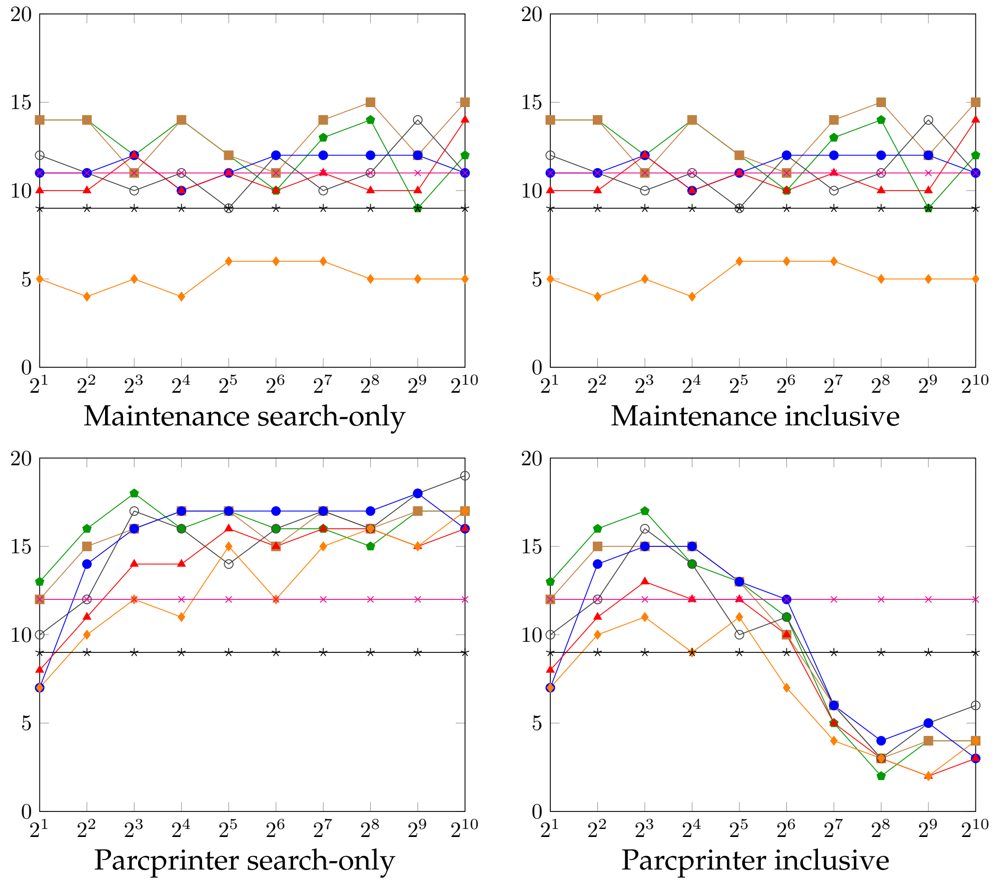


Figure 8: Coverage in individual domains.

Bonet, B., & Geffner, H. (1999). Planning as heuristic search: New results. In Biundo, S., & Fox, M. (Eds.), *Proceedings of the 5th European Conference on Planning (ECP'99)*, pp. 60–72. Springer-Verlag.

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.

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. AAAI Press.

Coles, A. J., Coles, A., Fox, M., & Long, D. (2013). A hybrid LP-RPG heuristic for modelling numeric resource flows in planning. *Journal of Artificial Intelligence Research*, *46*, 343–412.

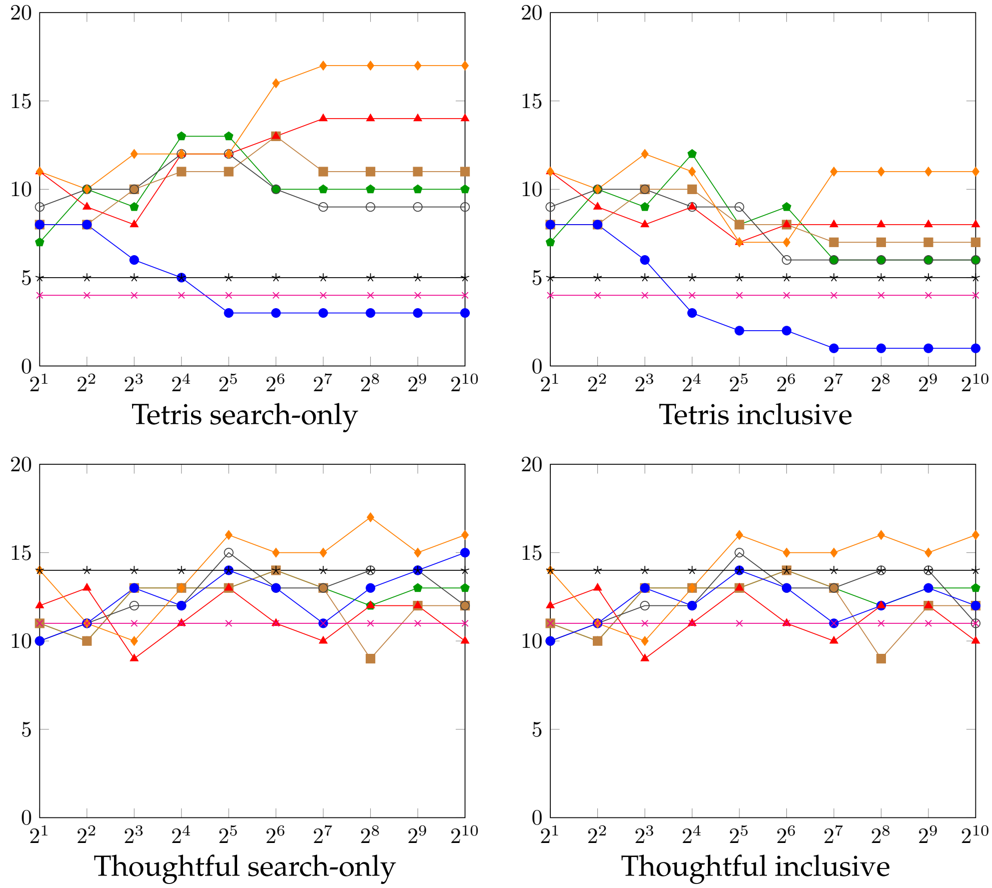


Figure 9: Coverage in individual domains.

Do, M. B., & Kambhampati, S. (2001). Sapa: A domain-independent heuristic metric temporal planner. In Cesta, A., & Borrajo, D. (Eds.), *Proceedings of the 6th European Conference on Planning (ECP'01)*, pp. 109–120. Springer-Verlag.

Domshlak, C., Hoffmann, J., & Katz, M. (2015). Red-black planning: A new systematic approach to partial delete relaxation. *Artificial Intelligence*, *221*, 73–114.

Fox, M., & Long, D. (2001). Hybrid STAN: Identifying and managing combinatorial optimisation sub-problems in planning. In Nebel, B. (Ed.), *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI-01)*, pp. 445–450, Seattle, Washington, USA. Morgan Kaufmann.

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. (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. 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. AAAI Press.

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, Menlo Park.

Helmert, M. (2004). A planning heuristic based on causal graph analysis. In Koenig, S., Zilberstein, S., & Koehler, J. (Eds.), *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS'04)*, pp. 161–170, Whistler, Canada. Morgan Kaufmann.

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

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. 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. (2011). Analyzing search topology without running any search: On the connection between causal graphs and h^+. *Journal of Artificial Intelligence Research*, *41*, 155–229.

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., Steinmetz, M., & Haslum, P. (2014). What does it take to render $h^+(\pi^c)$ perfect?. In *ICAPS 2014 Workshop on Heuristics and Search for Domain-Independent Planning (HSDIP'14)*.

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 2009)*, pp. 1734–1739, 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. AAAI Press.

Keyder, E., Hoffmann, J., & Haslum, P. (2014). Improving delete relaxation heuristics through explicitly represented conjunctions. *Journal of Artificial Intelligence Research*, *50*, 487–533.

Marques-Silva, J., & Sakallah, K. (1999). GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, *48*(5), 506–521.

McDermott, D. V. (1999). Using regression-match graphs to control search in planning. *Artificial Intelligence*, *109*(1-2), 111–159.

Moskewicz, M., Madigan, C., Zhao, Y., Zhang, L., & Malik, S. (2001). Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Conference on Design Automation (DAC-01)*, Las Vegas, Nevada, USA. IEEE Computer Society.

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

Steinmetz, M., & Hoffmann, J. (2016). Towards clause-learning state space search: Learning to recognize dead-ends. In Schuurmans, D., & Wellman, M. (Eds.), *Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI'16)*. AAAI Press.

Valenzano, R. A., Sturtevant, N. R., Schaeffer, J., & Xie, F. (2014). A comparison of knowledge-based GBFS enhancements and knowledge-free exploration. In Chien, S., Do, M., Fern, A., & Ruml, W. (Eds.), *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS'14)*. AAAI Press.

van den Briel, M., Benton, J., Kambhampati, S., & Vossen, T. (2007). An LP-based heuristic for optimal planning. In Bessiere, C. (Ed.), *Proceedings of the Thirteenth International Conference on Principles and Practice of Constraint Programming (CP'07)*, Vol. 4741 of *Lecture Notes in Computer Science*, pp. 651–665. Springer-Verlag.