Ranking Conjunctions for Partial Delete Relaxation Heuristics in Planning

Maximilian Fickert and **Jörg Hoffmann**
Saarland University
Saarland Informatics Campus
Saarbrücken, Germany
{fickert,hoffmann}@cs.uni-saarland.de

Abstract

Heuristic search is one of the most successful approaches to classical planning, finding solution paths in large state spaces. A major focus has been the development of domain-independent heuristic functions. One recent method are partial delete relaxation heuristics, improving over the standard delete relaxation heuristic through imposing a set C of conjunctions to be treated as atomic. Practical methods for selecting C are based on counter-example guided abstraction refinement, where iteratively a relaxed plan is checked for conflicts and new atomic conjunctions are introduced to address these. However, in each refinement step, the choice of possible new conjunctions is huge. The literature so far offers merely one simple strategy to make that choice. Here we fill that gap, considering a sizable space of basic ranking strategies as well as combinations thereof. We furthermore devise ranking strategies for conjunction-forgetting, where the ranking pertains to the current conjunctions and thus statistics over their usefulness can be maintained. Our experiments show that ranking strategies do make a large difference in performance, and that our new strategies can be useful.

Introduction

Classical planning demands to find solution paths in large state spaces, compactly represented in terms of a set of state variables, an initial state, a set of actions, and a goal condition. Planning tools handle any input planning task represented in such a language, and the search for a solution is required to be automatic, without any further human input. Heuristic search is one of the most successful approaches towards this endeavor (e. g. (Bonet and Geffner 2001; Hoffmann and Nebel 2001; Gerevini, Saetti, and Serina 2003; Richter and Westphal 2010)). A major focus here has been on the development of domain-independent heuristic functions: *How to automatically relax the planning input and produce goal estimates based on relaxed solutions?*

A highly successful answer to this question is based on the *delete relaxation*, which ignores negative action effects (effectively pretending that state variables accumulate, rather than switch between, their values). This preserves chains of prerequisite-effect and yields highly informative goal distance estimates in many planning domains (Hoffmann 2005). At the same time, the delete relaxation has major weaknesses in terms of disregarding the need for variables (like a vehicle in transportation) to move to-and-fro (after driving from A to B, under the delete relaxation the vehicle is at both A and B), and in terms of disregarding resource consumption (a single drop of fuel suffices, under the delete relaxation, to travel the world). *Partial delete relaxation* methods address these weaknesses systematically, ultimately allowing to interpolate between the delete relaxation and full, un-relaxed, planning. Known methods are *red-black relaxation* (Katz, Hoffmann, and Domshlak 2013; Domshlak, Hoffmann, and Katz 2015), and *explicit conjunctions* imposing a set C of atomic conjunctions (Keyder, Hoffmann, and Haslum 2012; 2014). We continue the latter line of work here. We focus on *satisficing* planning, where no optimality guarantee needs to be given; in particular, the heuristic functions we consider here are not admissible.

The delete relaxation can, at an abstract level, be viewed as achieving each fact, in the goal condition as well as regressed subgoals, in separation (ignoring negative interactions across subgoal facts). The explicit conjunctions method generalizes this by considering an arbitrary set C of fact *conjunctions* as atomic, forcing the relaxed plan to achieve each subgoal from C. The initial works (Haslum 2012; Keyder, Hoffmann, and Haslum 2012; 2014) compiled C into a standard planning syntax. Later works, establishing the most recent explicit-conjunctions heuristic $h^{C\text{FF}}$, observed that a compilation is not needed. This has computational advantages through not suffering from potentially large compilation size (Hoffmann and Fickert 2015; Fickert, Hoffmann, and Steinmetz 2016). All these works find the set C through *offline* counter-example guided abstraction refinement: before search begins, iteratively a relaxed plan for the initial state is checked for conflicts and new atomic conjunctions are introduced to address these. The process stops when a size bound is reached. Search is then invoked using the same set C throughout. Recent work has shown that it can be of advantage to learn C *online* during search instead, from the conflicts – local minima in a hill-climbing search – encountered (Fickert and Hoffmann 2017). But the core refinement step remains unchanged.

Here, we address the choice of conjunctions *within* each refinement step. The choice of possible new conjunctions is huge, up to millions. For computational reasons, most meth-

ods select just one out of these many possible conjunctions. (Precisely: this is true for all except Haslum's initial method (Haslum 2012), which in difference to all later methods considered optimal planning.) The literature so far offers merely one simple ranking strategy, preferring conjunctions incurring a low computational overhead. This is despite the fact that, even with that ranking in place, often a large choice of ("best") possible conjunctions persists; and it is despite the fact that it was observed early on (Keyder, Hoffmann, and Haslum 2012) that the choice of conjunctions is important, with even small changes having a large impact on results.

Here we fill that gap. We consider a sizable space of basic ranking strategies, targeted not only at minimizing overhead but also at capturing conjunction "importance". We systematically explore the performance of these strategies, as well as that of combined ranking strategies with lexicographic tie-breaking. We furthermore devise ranking strategies for conjunction-forgetting, where the ranking pertains to the current conjunctions and thus statistics over their usefulness can be maintained. We finally devise a new variant of the conflict extraction preceding the conjunction ranking, which gets rid of a major bottleneck in several standard planning benchmark domains. In terms of overall peformance, we find that different ranking strategies do make a large difference, and that our new strategies can be useful. In particular, our best performing search configuration beats the previous best explicit-conjunctions search, and therewith beats the state of the art on several domains.

Preliminaries

We introduce our basic notations, and the standard concepts forming the background to our work.

Planning Framework

We use the STRIPS framework for classical planning (Fikes and Nilsson 1971), the canonical simplest planning language where all state variables are Boolean. We use the standard notation where the state variables are notated as propositional-logic *facts*.

A planning *task* is defined as a tuple $\Pi = (\mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G})$, where $\mathcal{F}$ is a set of *facts*, $\mathcal{A}$ is a set of *actions*, $\mathcal{I} \subseteq \mathcal{F}$ is the *initial state*, and $\mathcal{G} \subseteq \mathcal{F}$ is the set of *goal* facts. An action $a \in \mathcal{A}$ is composed of its *preconditions* $pre(a)$, *add list* $add(a)$, and *delete list* $del(a)$.

A *state* $s \subseteq \mathcal{F}$ is a set of facts. An action a is *applicable* to s if $pre(a) \subseteq s$. Applying a to s results in the state $(s \cup add(a)) \setminus del(a)$. A sequence of actions leading from a state s to a goal state $s_G \subseteq \mathcal{G}$ is called a *plan* for s. A plan for a task Π is a plan for its initial state I. A plan is called *optimal* if it has minimal length among all plans (we assume action costs to be one for simplicity).

The set of all states is denoted by $\mathcal{S}$. A *heuristic function* (short *heuristic*) is a function $h : \mathcal{S} \mapsto \mathbb{N}_0 \cup \{\infty\}$ that estimates the goal distance for states $s \in \mathcal{S}$, where ∞ indicates that there is no plan for s (in this case s is called a *dead end*). The *perfect heuristic* h^* evaluates to the length of an optimal plan for any state s, or to ∞ if no plan for s exists.

Standard Heuristic Functions

The family of *critical path heuristics* h^m (Haslum and Geffner 2000) is based on the assumption that in order to achieve any set of facts g, it suffices to achieve their most costly m-sized subset. The h^C heuristic generalizes this to an arbitrary set of *conjunctions* (fact sets) C, such that the cost to achieve a fact set g is estimated by its most costly subset $g' \subseteq g, g' \in C$ (Hoffmann and Fickert 2015).

Throughout this paper we will assume C to contain all singleton conjunctions $c = \{f\}, f \in \mathcal{F}$. We will use the shorthand "X^C" to mean all conjunctions in a fact set X, i.e. $\{c \subseteq X \mid c \in C\}$.

We now define h^C formally. This is done via the concept of regression. A fact set g is *regressable* over an action a if $add(a) \cap g \neq \emptyset$ and $del(a) \cap g = \emptyset$. In that case, the *regression* of g over a is defined as $R(g, a) = (g \setminus add(a)) \cup pre(a)$, otherwise we write $R(g, a) = \bot$.

Given a state s and a set of conjunctions C, h^C is defined as $h^C = h(s, \mathcal{G})$ where h is the function that satisfies

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

We can compute h^C using a forward exploration until a fixed point is reached. First, $h^C(s, c)$ is set to 0 for all $c \in s^C$. Now we can iteratively set the h^C value for conjunctions where h^C is yet undefined, if there is an action a such that h^C is already assigned for all conjunctions $c' \in R(c, a)^C$. In that case, $h^C(s, c)$ is set to the maximum cost of the conjunctions c', plus 1 for having to apply a. The final $h^C(s)$ value is the maximum cost of the goal conjunctions $c \in \mathcal{G}^C$.

This algorithm can be efficiently implemented using counters that keep track on the number of precondition conjunctions that need to be achieved before a conjunction can be reached through a specific action. Specifically, counters $count(a, c)$ are attached to (action, conjunction) pairs (a, c) where $R(c, a) \neq \bot$. The counters are initially set to $|R(c, a)^C|$, and are decremented whenever a conjunction in $R(c, a)^C$ is reached for the first time. When a counter reaches zero, all conjunctions that are necessary to achieve c via a are reached and c can be used in the next iteration (Fickert, Hoffmann, and Steinmetz 2016).

The *delete relaxation* simplifies the task by assuming all delete lists to be empty. A corresponding plan is called a *relaxed plan*. The *perfect delete relaxation heuristic* h^+ evaluates to the length of an optimal delete-relaxed plan for any state s, or to ∞ if no such plan for s exists. The *delete relaxation* heuristic h^{FF} (Hoffmann and Nebel 2001) also estimates states by the length of a delete-relaxed plan, but does not require the plans to be optimal. In contrast to h^+, h^{FF} can be computed in polynomial time which makes it suitable to use as a heuristic.

For h^{FF} relaxed plans are usually generated using a *best supporter function* bs, which assigns an action to each fact that is deemed to be its cheapest achiever. A relaxed plan can then be extracted by starting from the goal facts, adding their best supporters to the relaxed plan, and propagating

their preconditions until all open facts are either already true in the current state or achieved by an action in the relaxed plan. In practice, h^{FF} usually uses h^{add} as the best supporter function, which corresponds to h^1 but sums over facts instead of taking the maximum.

$h^{C\text{FF}}$ and its Refinement Operation

The partial delete relaxation heuristic $h^{C\text{FF}}$ (Fickert, Hoffmann, and Steinmetz 2016) increases the accuracy of h^{FF} by forcing the relaxed plans to respect a given set of conjunctions C. It arises from the combination of h^{FF} and h^C: the transition from h^1 to h^{FF} can be seen as having to achieve *all* facts of a subgoal G instead of only the most costly fact $p \in G$. Similarly, $h^{C\text{FF}}$ has to achieve all subgoals $g \in G^C$ instead of only the most costly one for h^C. The corresponding perfect delete relaxation heuristic respecting a set of conjunctions C is denoted by h^{C+}. If C is the set of all singleton conjunctions, then $h^{C+} = h^+$. However, h^{C+} eventually converges to h^* when adding more conjunctions to C, thus allowing to interpolate between h^+ and h^* with the choice of C.

The partially relaxed plans $\pi[h^{C\text{FF}}]$ generated by $h^{C\text{FF}}$ consist of (action, supported conjunctions) pairs which we call *action occurrences*. For an action occurrence (a, G), the set of supported conjunctions G indicates which conjunctions are achieved. The preconditions for an action occurrence are $pre((a, G)) = (\bigcup_{c \in G} R(c, a))^C$. In practice, it suffices to consider only the *non-dominated* conjunctions as preconditions, i.e. those that are not a subset of a different conjunction that is also a precondition.

Similar to h^{FF}, partially relaxed plans for $h^{C\text{FF}}$ are also computed using a best supporter function based on h^C. In practice, we use $h^{C\text{add}}$, which is the equivalent extension for h^C as h^{add} is for h^1.

C-Refinement

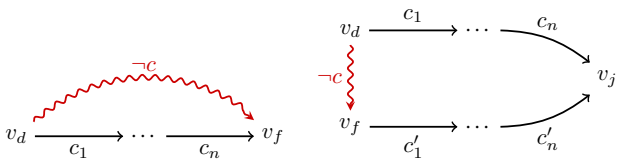
We use Keyder et al.'s refinement algorithm (Keyder, Hoffmann, and Haslum 2014) to generate a suiteable set of conjunctions for $h^{C\text{FF}}$. Given the current partially relaxed plan, the algorithm generates a set of candidate conjunctions $C_{cand} \cap C = \emptyset$ if the plan fails to execute in the real world (with delete semantics). A single candidate $c \in C_{cand}$ is selected to be added to C. The algorithm was originally defined for $h^{\text{FF}}(\Pi^C)$, which is the compilation equivalent of $h^{C\text{FF}}$, but can be straightforwardly adapted to $h^{C\text{FF}}$. We summarize the algorithm in the following.

First, the relaxed plan is put into a data structure called *Best Supporter Graph (BSG)*:

Definition 1 (Best Supporter Graph) *Given a state s for a planning task Π, a set of action occurrences $\pi[h^{C\text{FF}}](s)$ that can be sequenced to form a partially relaxed plan for s, and the corresponding best supporter function bs, the best supporter graph is a directed acyclic graph $\phi = \langle V, E \rangle$, where $V = \pi[h^{C\text{FF}}](s) \cup \{(a_{GOAL}, \emptyset)\}$, $E = \{\langle v, v' \rangle \mid \exists c \in pre(v') \wedge v = bs(c)\}$. Each vertex is labelled with the action occurrence it represents, and each edge is labelled with the set of (non-dominated) precondition conjunctions $\{c \mid c \in pre(v') \wedge v = bs(c)\}$.*

The BSG models the dependencies between action occurrences: if there is an edge from a vertex v to another vertex v', the action occurrence represented by v achieves a conjunction that is required as a precondition for the action occurrence represented by v'. There is an additional vertex for the goal which is represented by an artificial action a_{GOAL} with $pre(a_{GOAL}) = \mathcal{G}$.

If the partially relaxed plan fails to execute in the original planning task, there must be an action (the *deleter*) that deletes a precondition of another action (or the goal) occurring later in the plan (the *failed action*). The deleter and failed action can either be ordered sequentially or have a common descendant in the BSG, as shown in Figure 1.



(a) sequential conflict (b) parallel conflict

Figure 1: Conflict types in relaxed plans.

In the sequential case, there is a path from the deleter d to the failed action f. Let L be the set of labels on the path from d to f. Possible new conjunctions addressing this conflict are $\{c_L \cup c \mid c_L \in L\}$. In the parallel case, there is no path from d to f but they have a common descendant j. Let L_d be the set of edge labels on the path from d to j and L_f be the edge labels on the path from f to j. Possible new conjunctions are $\{c_{L_d} \cup c_{L_f} \mid c_{L_d} \in L_d, c_{L_f} \in L_f \cup \{c\}\}$. As only one conjunction will be selected to be added to C, it suffices to consider one candidate for a single conflict to avoid redundancy. The considered candidate conjunction is $c_n \cup c$ in the sequential case and $c_n \cup c'_n$ in the parallel case.

Candidate Ranking Strategies

The refinement algorithm returns a set of candidate conjunctions C_{cand} that are not part of C yet. While it is possible to add all the candidate conjunctions to C at once, in practice it is better to only add a single conjunction to minimize the computational overhead and avoid redundant conjunctions addressing the same conflicts (Keyder, Hoffmann, and Haslum 2014). However, to select a suitable candidate, the conjunctions must be ranked according to some criteria. We now discuss a range of ranking strategies. These strategies can be combined using lexicographic tie-breaking, which we denote as $\langle s_1, s_2, \ldots \rangle$ (s_1 is applied first, then s_2 is used to break the remaining ties etc.). If there are still multiple candidates remaining after applying all ranking strategies, the remaining ties will be broken arbitrarily.

Keyder et al. introduced the following ranking strategies:

- *min-distance*: rank the candidates based on the number of vertices between the deleter and the failed action in the BSG for sequential conflicts (e.g. if the deleter and failed action are ordered immediately after each other, this value is 0). For parallel conflicts this is set to 1.

- *min-counters*: rank the candidates based on the number of counters that are introduced when adding it to C. For a candidate conjunction c, this is computed as $|\{a \in A \mid R(c,a) \neq \bot\}|$.[1]

The intuition of the *min-distance* strategy is that preferring conflicts with minimal distance between the deleter and the failed action tends to yield more relevant conflicts. Additionally, this also allows for a faster implementation of the conflict extraction algorithm, because conflicts with lower distance are computationally easier to find, and if those are preferred anyway the algorithm can terminate earlier.

The two strategies are combined as ⟨*min-distance*, *min-counters*⟩. From the conflicts with minimal distance the one that introduces the minimal number of new counters is chosen in order to minimize the computational overhead.

However, the overhead is not the only important metric when considering which conjunctions to add to C. Some conjunctions might yield a more informative heuristic than others. Hence, we also consider strategies with very different approaches:

- *random*: rank the candidates randomly.
- *arbitrary*: rank the candidates arbitrarily (i.e. they stay in the order as generated).
- *min-size* and *max-size*: rank the candidates by their size.
- *min-counters-estimate*: estimate the number of additional counters by $\sum_{f \in c} |\{a \in A \mid R(\{f\}, a) \neq \bot\}|$ for a candidate conjunction c.
- *max-occurrences*: the number of conflicts for which the candidate conjunction was added to the set of candidates.
- *min-influence* and *max-influence*: rank the conjunctions based on the number of counters where the conjunction will appear as a precondition when added to C.
- *min-deleter-alternatives* and *max-deleter-alternatives*: rank the conjunctions based on how many of the deleter's supported conjunctions could be achieved equally well by different actions. Specifically, we calculate this as the percentage of supported conjunctions of the deleter that have other best supporters that do not delete the missing precondition of the failed action.

The strategies range from trivial methods (*random* and *arbitrary*) over simple features of the candidate conjunctions (e.g. *min-size*) to more complex strategies. First we briefly discuss the ideas behind the non-trivial strategies.

The *min-counters-estimate* is an attempt to reduce the computational overhead of the *min-counters* strategy: *min-counters* requires computing $R(c,a)$ for all candidates $c \in C_{cand}$ and actions $a \in A$, whereas for *min-counters-estimate* we only need the regressions for all unit conjunctions, which are already available as the unit conjunctions are always contained in C.

[1]For the compilation-based $h^{\text{FF}}(\Pi^C)$, this corresponds to the number of action representatives introduced in the compiled task Π^C.

The *max-occurrences* strategy arises from the observation that different conflicts may yield the same candidate conjunctions. These candidates can fix multiple conflicts at once when added to C.

The *influence*-based strategies estimate how many existing counters will be affected by a new conjunction.

Finally, the strategies considering the *deleter-alternatives* attempt to predict the changes to the partially relaxed plans when a conjunction is added. If there are no alternative actions to achieve the conjunctions supported by the deleter, the structure of the next relaxed plan will probably change more compared to the deleter just being exchanged for a different action.

Some of the strategies require additional computation or memory. For example, the *min-counters* strategy requires computing $R(c,a)$ for all candidate conjunctions $c \in C_{cand}$ and actions $a \in \mathcal{A}$, and for the *max-occurrences* strategy we have to keep track of the number of times each conjunction is added to the set of candidates. Our implementation avoids this overhead whenever it is not needed. For example, if the *min-counters* strategy only appears as the second tie-breaker, it is only evaluated for the remaining conjunctions. Similarly, if *max-occurrences* is not used as a ranking strategy, we do not need to count the number of times the same candidate is added to C_{cand}.

Online Ranking Strategies

In addition to ranking candidates that are not yet in C, we can also devise rankings for conjunctions that are already used by the heuristic. For these online ranking strategies, we can take features and statistics into account that can only be measured during search. This way, we may get more precise information about the conjunctions, which can be used to remove conjunctions from C again if they turn out not to be useful during search. We will evaluate these strategies by periodically replacing existing conjunctions in C that are deemed worst by the online ranking strategies by new ones. We consider the following strategies:

- *random*: rank the conjunctions randomly.
- *oldest*: rank the conjunctions by how long they are contained in C.
- *max-counters*: rank the conjunctions by the number of attached counters.
- *min-rp-frequency*: rank the conjunctions by how often they appear as a supported conjunction of any action occurrence in a partially relaxed plan (as a percentage of the number of generated relaxed plans since this conjunction was added to C).
- *min-$h^{C\text{add}}$-increase*: rank the conjunctions according to how often their $h^{C\text{add}}$ value increased over that of its dominated conjunctions (as a percentage of the number of evaluations since this conjunction was added to C).

For a more complex strategy, we first define the *effectiveness* of a conjunction as follows:

Definition 2 *A conjunction c is called* effective *in a state s if $\forall c' \subset c, c' \in C : h^{C\text{add}}(s,c') < h^{C\text{add}}(s,c)$ and either*

1. *$c \subseteq G$ and if $h^{C\text{add}}(s,c) = \infty$, then $\forall c' \subseteq G, c' \in C, c' \neq c : h^{C\text{add}}(s,c') \neq \infty$ or*
2. *there exists a counter $\text{count}_{c',a}$ attached to a conjunction c' and action a with $c \subseteq R(c',a)$ and either*
   - *$h^{C\text{add}}(s,c) < \infty$ and a is a best supporter of c' or*
   - *$h^{C\text{add}}(s,c) = \infty, h^{C\text{add}}(s,c') = \infty$, and $\forall c'' \in R(c',a)^C, c'' \neq c : h^{C\text{add}}(s,c'') \neq \infty$*

A conjunction c is considered effective in a state s if its $h^{C\text{add}}$ value increases over that of its subconjunctions $c' \subset c$, and the conjunction contributes in some way to the overall $h^{C\text{add}}$ value. The contribution is either by being part of the goal, or by being required for another conjunction. If c is unreachable ($h^{C\text{add}}(s,c) = \infty$), it must make either the goal or some other conjunction unreachable.

- *min-effective*: replace the conjunction that is effective the least often (as a percentage of the number of evaluations since this conjunction was added to C).

Since *min-counters* is already a successful ranking strategy for refinement, it appears logical that during search, replacing conjunctions with many attached counters by new ones with fewer counters can be beneficial.

The *min-rp-frequency* strategy considers how often conjunctions appear in the partially relaxed plans, assuming that conjunctions that are used in the relaxed plans often are more useful than others.

The *min-$h^{C\text{add}}$-increase* and *min-effective* strategies represent the attempt to capture the importance of a conjunction. The former is rather simple, and just considers if the conjunction tends to be more difficult to achieve than its subconjunctions. The latter additionally requires the conjunction to be used for something: either as part of the goal; or in a precondition of a counter attached to another conjunction, and the other conjunction is reached through that counter.

Conflict Extraction Algorithm

We finally designed a variant of the conflict extraction step, which precedes the selection of candidate conjunctions.

Keyder et al.'s implementation of the refinement algorithm searches conflicts in the BSG and proceeds as follows. First, only sequential conflicts with zero distance between the deleter and failed action in the BSG are collected. Only if no conflicts are found, the algorithm proceeds with the computationally much more difficult procedures to collect all other sequential conflicts and parallel conflicts. As such, this implementation is well optimized for their lexicographic tie-breaking, where candidates with minimal distance between the deleter and failed action are preferred.

Collecting conflicts from the BSG has the drawback that it can lead to a very high number of conflicts (thus taking a lot of time), and many of the conflicts may not actually appear in the ordered relaxed plan that is returned by the heuristic. We take a slightly different approach and extract the conflicts directly from the sequenced relaxed plan instead, only using the BSG to identify the conflict type and generate the candidate conjunction accordingly. In the experiments section we will show that this is beneficial, both for the run-time of the refinement algorithm and for the quality of the generated conjunctions.

Experiments

The described techniques are implemented in Fast Downward (FD) (Helmert 2006). We evaluate different conjunction selection strategies in two search algorithms: enforced hill-climbing with online-refinement (Online-EHC) (Fickert and Hoffmann 2017), and greedy best-first search (GBFS) where conjunctions can be learned before and during search. The former uses helpful actions pruning, the latter uses FD's dual queue with preferred operators and lazy evaluation. The best supporter function for $h^{C\text{FF}}$ is $h^{C\text{add}}$ with random tie breaking (the best-performing configuration in prior work). All results are averaged over three different random seeds. A problem instance counts as solved if it is solved with at least two different random seeds by the same configuration.

The experiments were run on machines with Intel Xeon E5-2650 processors with a clock rate of 2.3 GHz with time and memory limits of 30 minutes and 4 GB respectively. The experiments were run on the domains from the satisficing tracks of all IPCs, excluding those (namely Gripper, Miconic, Movie, Openstacks'06, and Openstacks'08) where Online-EHC can solve all problems without learning any conjunctions. This results in a total of 1465 benchmark instances from 41 domains.

Online-EHC performs successive breadth-first explorations from some state s until a state s' with better heuristic value is found. Then the next exploration starts from that state etc. until a goal state is reached. If no such s' with better heuristic value can be found within a specific lookahead bound (we use the default value of 3 in our experiments), new conjunctions are added in iterative refinement steps in s. The refinement process repeats until $h^{C\text{FF}}(s)$ increases enough for s not to be a local minimum anymore.

We will use the statistic of how many conjunctions have to be added to C during a run of Online-EHC as an indicator for the quality of the generated conjunctions. If fewer conjunctions are sufficient to guide search to the goal, the conjunctions must have a better impact on the heuristic than if more refinement steps were needed. As a measurement of the computational overhead that is incurred by the set of conjunctions, we use the increase in the number of counters in the h^C implementation.

GBFS with offline learning is configured to generate conjunctions until the number of counters increases by a factor of 1.5, or a timeout of 900 seconds is reached (this is the overall most competitive configuration from (Fickert, Hoffmann, and Steinmetz 2016)). After C is generated, search is started using the resulting $h^{C\text{FF}}$ heuristic.

Unless noted otherwise, the default lexicographical tie-breaking for candidate conjunctions is $\langle$*min-distance*, *min-counters*$\rangle$, i.e. the same one as used in prior work.

The experiments are structured as follows. We start out by assessing the impact of our changes to the conflict extraction algorithm. Then we evaluate the different ranking strategies; first considering the candidate ranking strategies and combinations of them before we turn to the online ranking strategies.

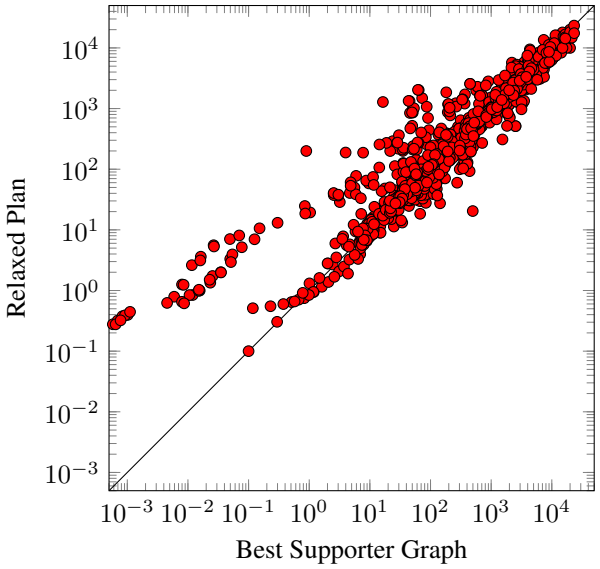


Figure 2: Refinement iterations per second using the original conflict extraction algorithm ("Best Supporter Graph") compared to our modified algorithm ("Relaxed Plan").

Conflict Extraction Algorithm

The main difference in our modifications to Keyder et al.'s conflict extraction algorithm is that we collect the conflicts directly from the extracted relaxed plan instead of considering all conflicts that appear in the BSG. We ran both conflict extraction algorithms in both GBFS with offline learning and Online-EHC. We will compare the run-times of the algorithms in GBFS since they are less influenced by other effects than in Online-EHC (e.g. learning in different states). In Online-EHC we will compare the overall performance and quality of the generated conjunctions.

Figure 2 compares the number of refinement operations of the original algorithm (labelled by "Best Supporter Graph") to our modified version ("Relaxed Plan"). Extracting the conflicts from the relaxed plan takes less time on most instances, in particular on instances where the refinement algorithm is generally slow, and a faster conflict extraction algorithm is especially desirable. Overall, our modified algorithm offers a 20-25 percent speed increase. The domains where our modified algorithm provides the biggest speedup are VisitAll (3x), Parcprinter (3.5x), Pathways (5.3x), Maintenance (5.9x), and Openstacks (50.5x). On the largest Openstacks instances, it is two orders of magnitude faster. The domains where the original algorithm has the biggest advantage are GED (1.5x faster), Barman (1.6x), and Sokoban (1.6x).

While we expected our algorithm to be faster because of a reduced number of conflicts, the number of conflicts generated by our modified algorithm is on average *more than three times greater* than using the original algorithm. In fact, the only domains where our modified algorithm yields fewer conflicts are Tetris (-11%), CityCar (-32%), Pathways (-73%), Maintenance (-92%), and Openstacks (-96%). Our algorithm can only find more conflicts if the original algorithm terminates early because of zero-distance conflicts. Since this does occur often and we expect the original algorithm to be superior in these cases, the difference in computation time must be significantly more impactful in the remaining cases where no zero-distance conflicts are present.

Ref. Algorithm	BSG	RP	BSG	RP
Domain	**Coverage**		**Conjunctions**	
Airport 50	40	**45**	144.5	**36.6**
Barman 40	24	**38**	4386.9	**2867.9**
ChildSnack 20	**3**	2	**1304.5**	1454.7
CityCar 20	**12**	10	**1843.0**	2028.2
Freecell 80	**74**	72	**19.3**	20.1
Nomystery 20	13	**14**	37.0	**35.7**
Parking 40	19	**29**	247.3	**69.0**
Pipes-NT 50	44	**45**	**13.1**	14.0
Sokoban 50	2	**6**	**1771.2**	2152.7
Storage 30	26	**27**	40.6	**37.6**
Tetris 20	15	**16**	354.3	**300.8**
Tidybot 20	14	**15**	384.5	**144.5**
Transport 70	**61**	60	**75.3**	76.7
Trucks 30	**18**	16	**58.4**	105.0
VisitAll 40	16	**18**	1119.4	**1058.1**
Others 885	**859**	**859**	30.4	**28.2**
Overall 1465	1240	**1272**	197.1	**159.8**

Table 1: Comparison of the original conflict extraction algorithm ("BSG") to our modified version ("RP") in Online-EHC. The table shows the coverage and number of conjunctions added to C during search for commonly solved instances. Domains with identical coverage are grouped to "Others".

We now evaluate the conflict extraction modifications in the Online-EHC. Table 1 shows an overview. The overall coverage increases significantly, with only slight losses of up to -2 in five domains. This can be attributed to the increased quality of the conjunctions, as shown by the number of added conjunctions for commonly solved instances (right half of the table). Since the original conflict extraction algorithm also includes conflicts that do not occur in the ordered relaxed plan computed by the heuristic, many superfluous conjunctions may be added that do not significantly improve the informativeness of the heuristic. Here the run-times of the algorithms only have a small impact on the overall search time. In fact, the difference in the ratio of learning time compared to the overall search time stays within 5% for all domains except GED, where the modified algorithm uses on average 23% of the overall search time for learning compared to 36% for the original algorithm.

In the remainder of this paper we use our version of the conflict extraction algorithm.

Candidate Ranking Strategies

We evaluate the different candidate ranking methods in Online-EHC, starting with the basic strategies before considering their combinations. In the following experiments

remaining ties are broken randomly.

Strategy	Cov.	Conj.	Overh.	Ties
⟨*min-distance*⟩	**1245**	**18.0**	**2.42**	6.07
⟨*min-counters*⟩	1188	28.8	2.50	0.37
⟨*arbitrary*⟩	1072	39.8	3.85	0
⟨*random*⟩	1069	37.6	3.90	0
⟨*min-size*⟩	1148	28.8	2.95	7.71
⟨*max-size*⟩	1049	46.4	4.38	4.43
⟨*min-counters-est.*⟩	1176	32.5	2.63	0.39
⟨*max-occurrences*⟩	1099	44.4	3.96	0.13
⟨*min-influence*⟩	1165	22.6	2.82	0.59
⟨*max-influence*⟩	1070	40.9	3.56	1.21
⟨*min-del-alt*⟩	1094	35.8	3.76	13.90
⟨*max-del-alt*⟩	1093	39.8	3.77	5.25

Table 2: Overview of the results with the basic candidate ranking strategies in Online-EHC. Remaining ties are broken randomly. The table shows the overall coverage ("Cov."), geometric mean of the number of added conjunctions for commonly solved instances ("Conj."), computational overhead as the geometric mean of the growth in the number of counters ("Overh."), and geometric mean of the remaining number of ties after the ranking strategies are applied ("Ties").

Table 2 shows an overview. The choice of the strategy has a huge effect on the overall coverage, which ranges from 1049 for the worst to 1245 for the best performing strategy. Overall, *min-distance* clearly outperforms the other strategies and requires Online-EHC to learn fewer conjunctions compared to any other ranking strategy. In almost every single domain it reaches at least close to the highest coverage among all ranking strategies. Interestingly, the *max-size* strategy results in the least informative conjunctions while generating the largest computational overhead. The *min-counters-estimate* strategy performs very similar but slightly weaker than the *min-counters*. However, the intended advantage by saving some computation time does not really have an effect as the evaluation of the ranking strategies takes only a very small share of the overall search time.

In some domains the results stay somewhat consistent between the different ranking strategies, in others the results can fluctuate wildly. We highlight the results for some selected domains and strategies in Table 3. The domains with the highest variance in coverage are Barman, Floortile, GED, and Parking. Parking is the domain where *min-distance* is beaten by the highest margin as it loses by -5 compared to *min-counters*.

We will now evaluate the ranking strategies as an additional tie breaker after the *min-distance* strategy, which performed consistently best. In all domains, there is on average still a choice among multiple equally good candidates after applying the *min-distance* strategy. However, in most domains the results are not affected much. An overview of the coverage on domains that display the biggest differences is given in Table 4, excluding some of the less interesting configurations for space reasons. The last row shows the average number of conjunctions added to C during search.

Coverage	⟨*min-dist.*⟩	⟨*min-count.*⟩	⟨*min-size*⟩	⟨*max-size*⟩	⟨*min-occ.*⟩	⟨*min-infl.*⟩	⟨*random*⟩
Airport 50	**45**	38	41	26	25	41	36
Barman 40	39	**40**	9	3	8	26	3
Floortile 40	**40**	7	5	9	32	7	4
GED 20	**20**	0	**20**	2	1	**20**	6
Nomystery 20	12	8	12	12	11	**15**	7
Parking 40	21	**26**	10	3	1	15	5
Tetris 20	13	9	**14**	5	4	11	6
Tidybot 20	**15**	**15**	14	8	8	14	12
Others 1165	1040	**1045**	1023	981	1009	1016	990
Sum 1465	**1245**	1188	1148	1049	1099	1165	1069

Table 3: Coverage results for selected ranking strategies and domains.

None of the new strategies manages to beat the state-of-the-art tie-breaking ⟨*min-distance*, *min-counters*⟩. The important observation here is that while the *min-counters* strategy only aims to reduce the computational overhead, it produces very informative conjunctions on top of that which makes it the most successful strategy.

Although it does not manifest in coverage, there are some domains where other strategies work better. In the Floortile, Hiking, and Sokoban domains, ⟨*min-distance*, *min-counters*⟩ performs worst both in terms of search time as well as number of generated conjunctions on commonly solved instances. Aside from Sokoban, which only has three commonly solved instances with the other ranking strategies and thus doesn't have the most meaningful data, the biggest improvement can be made in Hiking with a search time reduction of 62% using ⟨*min-distance*, *max-deleter-alternatives*⟩ as lexicographic tie-breaking.

Applying another tie-breaker after ⟨*min-distance*, *min-counters*⟩ becomes less interesting, as there are very few domains in which there are still reasonably many decisions to be made. Furthermore, none of them show any significant differences between the ranking strategies used as the third tie-breaker. The only meaningful observations can be made in the Parking domain, where ⟨*min-distance*, *min-counters*, *max-occurrences*⟩ performs better in terms of search time than the other options (around 15-40%). In terms of conjunction quality as measured by the number of conjunctions that need to be added in Online-EHC, it only concedes to ⟨*min-distance*, *min-counters*, *max-size*⟩, which in turn generates more computational overhead.

Online Ranking Strategies

We now consider the strategies to rank conjunctions that are already contained in C and used by the heuristic. In order to evaluate these strategies, we use GBFS with offline learning, and periodically (after every 25 evaluations) replace a single conjunctions by a new one. The conjunction to be replaced is selected according to each specific online ranking strat-

Coverage	⟨*md*, *min-count.*⟩	⟨*md*, *arbitrary*⟩	⟨*md*, *random*⟩	⟨*md*, *min-size*⟩	⟨*md*, *max-occ.*⟩	⟨*md*, *min-infl.*⟩	⟨*md*, *max-infl.*⟩
Barman 40	38	**39**	**39**	28	38	36	**39**
CityCar 20	**10**	8	8	9	7	8	5
Nomystery 20	14	12	12	14	13	**16**	13
Parking 40	**29**	21	21	22	20	23	28
Tetris 20	**16**	13	13	15	**16**	11	8
Tidybot 20	15	15	15	14	13	**16**	12
Transport 20	**60**	52	52	52	53	48	54
Others 1165	**1090**	1086	1085	1080	1073	1079	1082
Sum 1465	**1272**	1246	1245	1234	1233	1237	1241
Conjunctions	**129**	135	134	140	156	153	190

Table 4: Coverage results for lexicographic tie-breaking using the different candidate ranking strategies as an additional tie-breaker after *min-distance* (abbreviated as *md*) on selected domains. The last row shows the geometric mean of the number of added conjunctions for commonly solved instances.

egy, while the new conjunction is selected using the default candidate selection strategy, ⟨*min-distance*, *min-counters*⟩. In order to avoid removing conjunctions that have recently been added, the removed conjunction must have been part of C for at least 250 evaluations (except in the beginning before the threshold of 250 evaluations is reached for any conjunction). After replacing a conjunction, search is continued with the open and closed lists unchanged.

Strategy	Coverage	Evaluations	Overhead
random	1277	696.4	1.49
oldest	**1289**	712.2	1.51
max-counters	1256	781.3	**1.32**
min-rp-frequency	1253	**681.1**	1.58
min-$h^{C_{\text{add}}}$-increase	1199	766.7	1.48
min-effective	1213	729.3	1.54
fixed C	1177	980.6	1.45

Table 5: Overview of the results for the online ranking strategies using GBFS and replacing an existing conjunction by a new one every 25 evaluations. The table shows the overall coverage, geometric mean of the number of evaluations for commonly solved instances, and computational overhead as the geometric mean of the growth in the number of counters ("Overhead"). As a baseline, the last row shows the results if C remains fixed throughout search.

An overview of the results is shown in Table 5. The table contains an additional row for a baseline where conjunctions are never replaced, and C remains fixed throughout search.

Periodically replacing the oldest conjunction works best overall, with an impressive overall coverage of 1289. As expected, replacing the conjunctions with the greatest number of attached counters can further reduce the overhead during search. The *min-rp-frequency* strategy results in the overall lowest number of evaluations. However, there is no domain where it achieves a higher coverage than any other strategy, though it is tied for first in 24 domains. In the Openstacks, Pathways, and VisitAll domains the fixed-C baseline has the highest coverage. The biggest difference can be observed in Openstacks, where using a fixed-C solves 37 out of 40 instances, the *min-effective* and *min-$h^{C_{\text{add}}}$-increase* strategies only achieve a coverage of 6 respectively 7, and the other strategies achieve a coverage of 25-26. Interestingly, the *random* strategy performs very well, and is only slightly worse than the *oldest* strategy overall.

All of the online replacement configurations beat the one with a fixed C by some margin, especially considering the number of evaluations. Since the heuristic steadily includes new conjunctions, it adapts itself to the part of the search space that is currently being explored. Conjunctions are often only relevant in some areas of the search space, but represent computational overhead in others without improving the informativeness of the heuristic. The *oldest* strategy tackles this problem the most effectively, as the oldest conjunctions tend to be the ones that were learned in now distant parts of the search space and are not relevant in recently generated relaxed plans.

Coverage	*oldest*	Onl.-EHC	FF	LAMA	Mercury
Airport 50	41	**45**	35	32	32
Childsnack 20	1	2	0	**5**	0
CityCar 20	**15**	10	1	5	5
Floortile 40	36	**40**	8	8	8
Maintenance 20	**17**	**17**	11	0	7
Nomystery 20	5	**14**	8	11	**14**
Openstacks 40	26	**40**	**40**	**40**	**40**
Parking 40	34	29	20	**40**	**40**
Pathways 30	22	**30**	23	23	**30**
Sokoban 50	46	6	**48**	**48**	44
Storage 30	**28**	27	20	19	19
Tetris 20	**19**	16	14	13	**19**
Thoughtful 20	14	**20**	15	16	13
Transport 70	52	60	32	61	**70**
VisitAll 40	19	18	7	**40**	**40**
Others 955	**914**	898	856	902	911
Sum 1465	1289	1272	1138	1263	**1292**

Table 6: Coverage results for the best performing online ranking strategy compared to the best performing Online-EHC configuration and the state of the art.

The best performing replacement configuration (using the *oldest* ranking strategy) even beats the overall best Online-EHC configuration and is competitive to the state of the art. A comparison to Online-EHC as well as FF (Hoffmann and Nebel 2001), LAMA (Richter and Westphal 2010), and Mer-

cury (Katz and Hoffmann 2014) is shown in Table 6. Domains where differences are small are grouped to "Others".

The biggest advantage for the online replacement configuration is in CityCar with 15 solved instances, compared to 10 for Online-EHC and 5 for the next best state-of-the-art configuration. It also compares very well to the state of the art in the Storage domain with $+8$ over the next best planner (FF), although Online-EHC also works well here. LAMA and Mercury use heuristics based on landmarks in conjunction with delete relaxation heuristics, which works especially well in the Transport and VisitAll domains.

Across almost all other domains, the online replacement configuration is consistently among the best performing configurations. In 8 domains it performs better than any of FF, LAMA, and Mercury, in 21 domains it is tied for best, and it is the worst configuration in only 4 domains.

Compared to Online-EHC, it has higher coverage in 11 domains, but also lower coverage in equally many domains. Most notably, in Sokoban, where Online-EHC performs worst, the replacement configuration has an improvement of $+40$ in coverage.

Conclusion

Research on partial delete relaxation via explicit conjunctions has so far all but ignored the intricacies of how to rank conjunctions in the abstraction-refinement step. We have filled that gap, with an extensive evaluation of strategies and strategy combinations, both for candidates that have yet to be added to C, and for already existing conjunctions. It turns out that the previous simple strategies are already very competitive. But they can be improved in particular domains, and a simple online-replacement strategy achieves state-of-the-art performance.

Of course, more things could be tried; we conjecture though, given our already very broad set of strategies, that not much more performance improvement can be obtained through varying this particular algorithm parameter. The only exception consists in more intelligent strategies for choosing the time points of conjunction replacement, as well as the number of conjunctions to be (removed or) replaced. This is an interesting direction for further work.

Acknowledgments. This work was partially supported by the German Research Foundation (DFG), under grant HO 2169/5-1, "Critically Constrained Planning via Partial Delete Relaxation". We thank the anonymous reviewers, whose comments helped significantly to improve this paper.

References

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

Bonet, B.; McCluskey, L.; Silva, J. R.; and Williams, B., eds. 2012. *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*. AAAI Press.

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

Fickert, M., and Hoffmann, J. 2017. Complete local search: Boosting hill-climbing through online heuristic-function refinement. In *Proceedings of the 27th International Conference on Automated Planning and Scheduling (ICAPS'17)*. AAAI Press.

Fickert, M.; Hoffmann, J.; and Steinmetz, M. 2016. Combining the delete relaxation with critical-path heuristics: A direct characterization. *Journal of Artificial Intelligence Research* 56(1):269–327.

Fikes, R. E., and Nilsson, N. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2:189–208.

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

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

Haslum, P. 2012. Incremental lower bounds for additive cost planning problems. In Bonet et al. (2012), 74–82.

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

Hoffmann, J., and Fickert, M. 2015. Explicit conjunctions w/o compilation: Computing $h^{\mathrm{FF}}(\Pi^C)$ in polynomial time. In Brafman, R.; Domshlak, C.; Haslum, P.; and Zilberstein, S., eds., *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS'15)*. AAAI Press.

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

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

Katz, M., and Hoffmann, J. 2014. Mercury planner: Pushing the limits of partial delete relaxation. In *IPC 2014 planner abstracts*, 43–47.

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

Keyder, E.; Hoffmann, J.; and Haslum, P. 2012. Semi-relaxed plan heuristics. In Bonet et al. (2012), 128–136.

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

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