Complete Local Search: Boosting Hill-Climbing through Online Relaxation Refinement

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

Abstract

Several known heuristic functions can capture the input at different levels of precision, and support *relaxation-refinement* operations guaranteeing to *converge* to exact information in a finite number of steps. A natural idea is to use such refinement online, during search, yet this has barely been addressed. We do so here for local search, where relaxation refinement is particularly appealing: *escape local minima not by search, but by removing them from the search surface*. Thanks to convergence, such an escape is always possible. We design a family of hill-climbing algorithms along these lines. We show that these are *complete, even when using helpful actions pruning*. Using them with the partial delete relaxation heuristic $h^{C\text{FF}}$, the best-performing variant outclasses FF's enforced hill-climbing, outperforms FF, outperforms dual-queue greedy best-first search with h^{FF}, and in 6 IPC domains outperforms both LAMA and Mercury.

Introduction

Many heuristic functions can capture the input at different levels of precision. Abstractions (e. g. (Clarke, Grumberg, and Long 1994; Culberson and Schaeffer 1998; Edelkamp 2001)) span the entire range between the exact heuristic h^* and the null heuristic $h = 0$. Critical-path heuristics (e. g. (Haslum and Geffner 2000; Haslum 2006; Hoffmann and Fickert 2015)) yield more information through treating larger sets C of conjunctions as being atomic. Partial delete relaxation heuristics (e. g. (Keyder, Hoffmann, and Haslum 2014; Domshlak, Hoffmann, and Katz 2015)) allow to interpolate between the delete relaxation and exact planning.

All these methods have powerful parameters allowing to choose the trade-off between computational effort and precision. One wide-spread means to make that choice in practice are *refinement* operations, starting from a null or simple relaxation, *converging* to an exact relaxation given sufficient time & memory (Clarke et al. 2003; Haslum et al. 2007; Seipp and Helmert 2013; Helmert et al. 2014; Keyder, Hoffmann, and Haslum 2014; Steinmetz and Hoffmann 2017).

The traditional approach is to refine the relaxation offline, prior to search, up to some computational limit. Yet the most pertinent information for targeted refinement – the difficulties actually encountered during search – becomes known online only. *Online relaxation refinement* appears to be the right answer, but has barely been addressed. Seipp (2012) touches Cartesian-abstraction online refinement, yet only briefly at the end of his M.Sc. thesis (with disappointing results). Steinmetz and Hoffmann (2017) refine a critical-path dead-end detector instead of a distance estimator. Wilt and Ruml's (2013) bidirectional search uses the backwards part as a growing perimeter to improve the forward-part heuristic function, which can be viewed as a form of relaxation refinement. Certainly, this is not all there is to be done.

That said, online relaxation refinement is one form of online heuristic-function learning, which has been explored in some depth for different forms of learning. The most traditional form is per-state value updating, as in transposition tables (Akagi, Kishimoto, and Fukunaga 2010), LRTA* (Korf 1990), and real-time dynamic programming (Barto, Bradtke, and Singh 1995; Bonet and Geffner 2003). This lacks the generalization inherent in refining a heuristic function. Various works learn or refine the combination of a given ensemble of heuristic functions (Felner, Korf, and Hanan 2004; Fink 2007; Katz and Domshlak 2010; Karpas, Katz, and Markovitch 2011; Domshlak, Karpas, and Markovitch 2012). This does not refine the actual relaxations, i. e. the information sources underlying the heuristics. Online training of machine-learned (ML) distance predictors has been realized through bootstrapping and local surface-error correction (Humphrey, Bramanti-Gregor, and Davis 1995; Arfaee, Zilles, and Holte 2011; Thayer, Dionne, and Ruml 2011). This lacks a comparable convergence guarantee: in a finite number of steps, for arbitrary inputs, without having to design/choose a sufficient feature representation.

A common perception in favor of offline heuristic-function refinement/learning presumably is that a frequently changing heuristic yields volatile search guidance. For global searches like A* or greedy best-first search (GBFS), do we need to reevaluate the entire open list upon each refinement step? Possibilities are to instead restart, or spawn parallel search processes (Arfaee, Zilles, and Holte 2011). The alternative we consider here is to use *local search* instead. This is particularly appealing as the local view removes the need for global reevaluation. *When stuck in a local minimum, instead of trying to escape it by search, refine the relaxation to remove it from the search space surface*. Observe that convergence guarantees this to work: after sufficient refinement, the local minimum will be gone. Indeed, as we show, this renders local search *complete*.

We design a family of online-refinement hill-climbing algorithms. Like FF's *enforced hill-climbing* (EHC) (Hoffmann and Nebel 2001), we consider lookahead searches on each state during a hill-climbing run. In difference to EHC, we limit the lookahead horizon by a parameter k. This trades off the amount of search vs. refinement used to escape local minima: we invoke refinement if search does not find a better state within the horizon. In particular, for $k = 1$, this yields a standard hill-climbing algorithm, but invoking refinement when no strictly better immediate successor exists.

We identify suitable convergence properties as eventually (i) detecting all dead-end states, and (ii) forcing the relaxed solutions underlying the heuristic function to be real solutions on solvable states. We show that, given these properties and using restarts on dead-end states, our hill-climbing algorithms are complete. This remains so *even when using helpful actions pruning*, i. e., when restricting the search to those actions appearing in the relaxed solutions.

We test our algorithms with the partial delete relaxation heuristic $h^{C\text{FF}}$, that combines the delete relaxation with critical-path heuristics, considering a set C of conjunctions to be atomic (Haslum 2012; Keyder, Hoffmann, and Haslum 2012; 2014; Hoffmann and Fickert 2015; Fickert, Hoffmann, and Steinmetz 2016). Refinement operations here consist in adding new conjunctions into C. This is known to provide the necessary properties (i) and (ii). In our experiments on IPC benchmarks, the best-performing hill-climbing variant vastly outclasses EHC, outperforms FF, outperforms dual-queue greedy best-first search with h^{FF}, and in 6 domains outperforms both LAMA and Mercury. For reference, we also experiment with variants of online-refinement GBFS.

Background

Framework and Basic Concepts

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 $\mathcal{F}$.

A *state* s is a subset of facts $s \subseteq \mathcal{F}$. An 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* π for s is a sequence of iteratively applicable actions leading from s to a state s_G s.t. $s_G \supseteq \mathcal{G}$. π is *optimal* if its length is minimal among all plans for s (we consider unit action costs for simplicity). A plan for Π is a plan for the initial state $\mathcal{I}$.

We denote the set of all states by $\mathcal{S}$. A *heuristic function*, short *heuristic*, is a function $h : \mathcal{S} \mapsto \mathbb{N}_0 \cup \{\infty\}$ mapping states to natural numbers, or to ∞ to indicate that the state is unsolvable (a *dead-end*). We assume that $h(s) = 0$ iff $s \supseteq \mathcal{G}$; and that $h(s) = \infty$ only if s is a dead-end, i. e., if the heuristic indicates s to be a dead-end then this is indeed so. These assumptions are satisfied for most if not all heuristics in planning. The *perfect heuristic* h^* maps any state s to the length of an optimal plan for s, or to ∞ if s is a dead-end.

Heuristic functions h are generally based on *relaxations*, and can typically be phrased in terms of returning the length of a relaxed solution $\pi[h](s)$. The *helpful actions* (aka *preferred operators*) associated with h then are those actions in $\pi[h](s)$ that are applicable to s. We denote that action set by $H(s)$. A search uses *helpful actions pruning* if, when expanding a state s, only the actions in $H(s)$ are considered.

We say that a planning algorithm is *sound* if, whenever it returns an action sequence π, then π is a plan. The algorithm is *complete* if it terminates in finite time, finds a plan in case Π is solvable, and proves unsolvability otherwise. All algorithms we consider here are sound, but only some of them are complete. In particular, $H(s)$ typically does not guarantee to contain an action starting a plan for s, so helpful actions pruning is a source of incompleteness.

The $h^{C\text{FF}}$ Heuristic & its Refinement Operation

Our design of online-refinement search algorithms is independent of the heuristic used. But in our experiments we use $h^{C\text{FF}}$, and also the convergence properties we identify are inspired by $h^{C\text{FF}}$. So we next give a brief summary of $h^{C\text{FF}}$ and its refinement operation. For details please see Fickert et al.'s (2016) respectively Keyder et al.'s (2012) work.

The $h^{C\text{FF}}$ heuristic combines the *critical-path* heuristic h^C (Haslum and Geffner 2000; Haslum 2006; Hoffmann and Fickert 2015) with the *delete relaxation* heuristic h^{FF} (Hoffmann and Nebel 2001). The critical-path relaxation underlying h^C assumes that, to achieve a set G of facts, it suffices to achieve the most costly *atomic conjunction* c (fact set) contained in G. The set C of atomic conjunctions is a parameter. (Originally, in h^m, C was set to all conjunctions up to size m; later on, this was generalized to arbitrary C.)

The delete relaxation assumes that all delete lists are empty (Bonet and Geffner 2001). The ideal, but hard to compute, heuristic h^+ returns the length of an optimal relaxed solution, or ∞ if no relaxed solution exists. Its approximation h^{FF} returns ∞ in the same case (deciding relaxed plan existence is easy), and otherweise returns the length of some, not necessarily optimal, relaxed solution $\pi[h^{\text{FF}}]$.

The combination of h^C and h^{FF} results from perceiving the delete relaxation as being like h^1, but requiring to achieve *every* fact p, rather than only the single most costly p, in a subgoal G. The relaxation then amounts to considering each $p \in G$ in isolation, ignoring interferences across these p. The generalization to arbitrary sets C of atomic conjunctions applies that same principle, but to the conjunctions $c \subseteq G, c \in C$ instead of the single facts $p \in G$. The idealized heuristic h^{C+} returns the length of an optimal plan in this extended relaxation. Its approximation $h^{C\text{FF}}$ returns the length of a not necessarily optimal such plan $\pi[h^{C\text{FF}}]$.

If C contains exactly the singleton conjunctions, then $h^{C+} = h^+$. However, h^{C+} *converges* to h^*, i. e., the value of h^{C+} can only grow as we add more conjunctions into C, and there always exists C such that $h^{C+} = h^*$. In this sense, h^{C+} incorporates a *partial delete relaxation*, allowing to interpolate all the way between h^+ and h^*. Yet how to actually find a good set C in practice, i. e., one that yields an accurate but not too expensive heuristic $h^{C\text{FF}}$?

All known answers are based on *refinement* steps. Such a step proceeds as follows, given a state s where $h^{C\text{FF}}(s) \neq \infty$: extract a partially relaxed plan, $\pi[h^{C\text{FF}}](s)$; if $\pi[h^{C\text{FF}}](s)$ actually is a real plan for s, then do nothing;

otherwise, identify one reason why $\pi[h^{C\text{FF}}](s)$ fails to be a real plan, and generate atomic conjunctions addressing that reason. Haslum's (2012) variant of this technique guarantees progress in a strong sense, namely that adding the generated atomic conjunctions into C excludes $\pi[h^{C\text{FF}}](s)$ from the space of partially relaxed plans for s. Keyder et al.'s (2012; 2014) variant of refinement generates, in each step, a single atomic conjunction c only, and guarantees a weaker form of progress, namely that the generated c is new, $c \notin C$. Here we use Keyder et al.'s method, which tends to construct smaller C and thus incur less overhead. Henceforth, when we say refinement, we mean Keyder et al.'s variant.

Converging Heuristic Functions

As pointed out, h^{C+} converges to h^*. This is the strongest convergence property possible – after a finite number of refinement steps, the heuristic will be perfect. We now identify a weaker convergence property that still suffices for our hill-climbing algorithms to be complete as advertised. That convergence property is satisfied by $h^{C\text{FF}}$. We start with $h^{C\text{FF}}$ to make things concrete, then derive our general definition.

The reason why $h^{C\text{FF}}$ does not converge to h^* is overestimation: $h^{C\text{FF}}$ is not an admissible heuristic function. It does, however, (i) converge to perfect information regarding dead-end states (as both h^{C+} and $h^{C\text{FF}}$ return ∞ in the same cases); and (ii) its relaxed solutions $\pi[h^{C\text{FF}}]$ converge to real plans on solvable states. This follows directly from prior results. To state it precisely, given a planning task with fact set $\mathcal{F}$, denote by $C_* := \mathcal{P}(\mathcal{F})$ the maximal set of conjunctions, considering *all* conjunctions to be atomic. We have:

Proposition 1 *Let $\Pi = (\mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G})$ be a planning task, and let s be a state. Then there exists C such that (i) in case s is unsolvable, we have $h^{C\text{FF}}(s) = \infty$; and (ii) in case s is solvable, $\pi[h^{C\text{FF}}](s)$ is a plan for s. In particular, both (i) and (ii) hold for $C = C_*$.*

Proof: We prove (i) and (ii) for $C_* = \mathcal{P}(\mathcal{F})$.

For (i): By Fickert et al.'s (2016) Corollary 1, $h^{C\text{FF}}(s) = \infty$ iff $h^{C+}(s) = \infty$, and there exists C s.t. $h^{C+}(s) = h^*(s)$. As $h^{C_*+} \geq h^{C+}$ for any C, this shows the claim.

For (ii): As s is solvable, $h^*(s) \neq \infty$, so $h^{C_*+}(s) \neq \infty$ and $h^{C_*\text{FF}}(s) \neq \infty$. Thus we can run C-refinement on s. Assume that $\pi[h^{C\text{FF}}](s)$ is not a plan for s. Then, by Keyder et al.'s (2014) Lemma 3, C-refinement on s generates an atomic conjunction $c \notin C_*$, in contradiction. ∎

Observe that continued C-refinement will eventually result in C_*, unless we terminate it earlier on: if we keep refining C, each step adding a new conjunction, then eventually C will contain all conjunctions. Therefore, Proposition 1 shows that, starting from any arbitrary C, $h^{C\text{FF}}$ will eventually (i) detect all dead-ends and (ii) solve all solvable states. This is precisely the convergence property we will need.

Given a heuristic h, we capture a *refinement operation* at an abstract level as a mapping ρ from h to a modified heuristic $\rho[h]$, where ρ is again applicable to $\rho[h]$ (and ρ possibly but not necessarily requires, as $h^{C\text{FF}}$ does, also an input state s where $h(s) \neq \infty$ and $\pi[h]$ is not a plan for s). We can then define convergence as follows:

Definition 1 (Converging Heuristic) *Let Π be a planning task. Let h be a heuristic function based on relaxed solutions $\pi[h]$, associated with a refinement operation ρ. We say that h is* converging *with ρ if there exists $N \in \mathbb{N}_0$ s.t. (i) for all $s \in \mathcal{S}$ where $h^*(s) = \infty$, $\rho^N[h](s) = \infty$; and (ii) for all $s \in \mathcal{S}$ where $h^*(s) < \infty$, $\pi[\rho^N[h]](s)$ is a plan for s.*

As argued, $h^{C\text{FF}}$ is converging with Keyder et al.'s refinement method. The same is true, in principle, of arbitrary abstraction heuristics, so long as the refinement step allows to obtain h^* (i. e., to have the abstract state space equal the real state space in the limit). Similarly, critical-path heuristics are converging with any refinement operation that adds at least one new atomic conjunction in each step. Thus all of these heuristic functions qualify for the search algorithms and completeness results in the next section. (Though it remains to be investigated of course whether heuristics other than $h^{C\text{FF}}$ are practically userful in this role.)

Online-Refinement Hill-Climbing

We introduce a family of local search algorithms with online relaxation refinement. Essentially, the algorithms are hybrids between standard *hill-climbing (HC)* and *enforced hill-climbing (EHC)* as introduced in FF (Hoffmann and Nebel 2001), enriched by restarting and relaxation refinement. By HC, we mean a simple gradient descent, a loop of action-selection steps each considering a current state s, stopping when $h(s) = 0$ and otherwise selecting an immediate successor with best (lowest) h value. EHC modifies this through a complete lookahead within each action selection step, running a breadth-first search for a state s' with strictly better heuristic value $h(s') < h(s)$. We obtain hybrids between HC and EHC by fixing the lookahead horizon k. This is well suited to online refinement as the limited horizon provides us with a crisp definition of when h is not "good enough": *we refine h if there is no better state within the horizon*.

Observe that k trades off search vs. refinement. For $k = 1$, the base search is plain HC and refinement does all the "hard work"; for $k = \infty$ the base search is EHC and refinement is only triggered if the entire search space below s was unsuccessfully exhausted; for intermediate values of k, e. g. $k = 2$, simple situations are handled by a quick search, while tougher ones call on refinement.

Towards designing our algorithm family, that we call *Refinement-HC*, it will be convenient to first design an intermediate family *Episode-EHC*, which is like EHC but running multiple *episodes*, with restarting in between episodes, to handle *failure*.[1] An EHC failure occurs if the search space in a lookahead phase becomes empty, because either the current state s is a dead-end, or helpful actions pruning is too aggressive. The original EHC just gives up in this case (FF switches to GBFS as a simple meta-heuristic). Episode-EHC instead provides the option to start another, randomized, episode of EHC. Figure 1 provides pseudo-code for

[1]Episode-EHC is related to Coles et al.'s (2007) stochastic EHC, in that both use restarts in some form. But Coles et al. investigate stochastic local search, while for us Episode-EHC is merely an intermediate construction for online-refinement search.

procedure *Episode-EHC:*
$\mathcal{C}_{de} := \emptyset$; /* cross-episode dead-end cache */
$s_0 := \mathcal{I}$; $\pi_0 := \langle\rangle$; /* first episode, start state and plan */
while TRUE **do** /* loop over EHC episodes */
  $s := s_0$; $\pi := \pi_0$;
  while TRUE **do** /* one EHC episode */
    if $h(s) = 0$ **then return** π; **endif**;
    [Breadth-First Search for state s' s.t. $h(s') < h(s)$];
    if *[Failure Trigger]* **then** /* EHC episode failed */
      if $\neg$*Helpful* **then** store s in $\mathcal{C}_{de}$; **endif**;
      [Failure Handling];
    else
      [Refinement Phase]; /* ONLY in Refinement-HC */
      $\pi := \pi\circ$ path from s to s'; $s := s'$;
    endif;
  endwhile;
endwhile;

[Breadth-First Search]:
Standard BrFS, handling search states t as follows:
  when generating t: prune t if $t \in \mathcal{C}_{de}$ or $h(t) = \infty$
  when expanding t: randomize the order of t's successor states
    if *Helpful*, apply $H(t)$ actions only
  when computing $h(t)$:
    if $\pi[h](t)$ is a plan for t **return** $\pi\circ$ path from s to $t \circ \pi[h](t)$
    if $h(t) = \infty$ set $\mathcal{C}_{de} := \mathcal{C}_{de} \cup \{t\}$

[Failure Trigger]: search space exhausted, no s' can be found

[Failure Handling]: /* only *Giveup* in original EHC */
if $s = \mathcal{I}$ **then**
  if $s \in \mathcal{C}_{de} \vee h(s) = \infty \vee \neg$*Helpful* **then return** "unsolvable";
  else return "don't know"; **endif**;
else
  case:
    Giveup: {**return** "don't know";};
    Restart: {$s_0 := \mathcal{I}$; $\pi_0 := \langle\rangle$; **break**;};
    Backjump: {$s_0 :=$ last state $t \neq s$ along π where $h(t) \neq \infty$,
      or I if no such t exists;
      $\pi_0 :=$ prefix of π up to s_0; **break**;};
  endcase;
endif;

procedure *Refinement-HC:*
[Refinement Phase]:
if *[Refinement Trigger]* **then** *[Refinement Handling]*; **endif**;
if *[Stagnation Trigger]* **then** *[Stagnation Handling]*; **endif**;

[Breadth-First Search]:
Like in Episode-EHC, with depth limit k;
$h_{\min} :=$ minimum h-value of search states $t \neq s$;

[Failure Trigger]: /* no better state at any depth below s */
$h(s) \leq h_{\min}$ and no search states cut off due to the depth limit

[Failure Handling]:
refine h on s, i. e., replace h with $\rho[h]$;
if $s = \mathcal{I}$ **then**
  if $s \in \mathcal{C}_{de} \vee h(s) = \infty$ **then return** "unsolvable"; **endif**;
else [like in Episode-EHC Failure Handling] **endif**;

[Refinement Trigger]: $h(s) \leq h_{\min}$ /* no better state found */

[Refinement Handling]: /* increase $h(s)$ beyond previous $h_{\min}$ */
while $h(s) \leq h_{\min}$ **do**
  if $\pi[h](s)$ is a plan for s **then return** $\pi \circ \pi[h](s)$; **endif**;
  refine h on s, i. e., replace h with $\rho[h]$
endwhile;
continue;

[Stagnation Trigger]: /* still no better state after refinement */
refinement trigger applies twice in a row on the same s

[Stagnation Handling]:
case:
  StagContinue: {};
  StagRestart: {$s_0 := \mathcal{I}$; $\pi_0 := \langle\rangle$; **break**;};
  StagBackjump: {$s_0 := s$;
    while $s_0 \neq \mathcal{I}$ **do**
      $s_0 :=$ the predecessor of s_0 along π;
      [Breadth-First Search
        for state s'_0 s.t. $h(s'_0) < h(s_0)$];
      if such s'_0 was found **then break**; **endif**;
    endwhile;
    $\pi_0 :=$ prefix of π up to s_0;
    break;};
endcase;

Figure 1: Episode-EHC, extending EHC with failure handling (left); modifications of Episode-EHC for Refinement-HC (right). We assume a heuristic h based on relaxed solutions $\pi[h]$, with helpful actions H; Refinement-HC also assumes a refinement operation ρ. *Helpful*, *Giveup*, *Restart*, *Backjump*, *StagContinue*, *StagRestart*, *StagBackjump*, and $k \in \mathbb{N}$ are parameters.

both, Episode-EHC and Refinement-HC. Consider first the left-hand side to understand the details of Episode-EHC.

The "*[X]*" notation in Figure 1 identifies *macros* (not sub-procedures), moved outside the main loop for readability; other cursive words, like *Helpful*, are configuration options. Consider first the main loop of Episode-EHC. It wraps the main loop of EHC into an outside loop over EHC episodes. Each episode starts from a state s_0 and corresponding plan prefix π_0 leading from $\mathcal{I}$ to s_0, set initially to $\mathcal{I}$ and $\langle\rangle$, and set later on by failure handling. The cross-episode dead-end cache is new relative to the original formulation of EHC. We include it to point out that, without helpful actions pruning, such a cache suffices to ensure cross-episode progress and, therewith, completeness (see Proposition 2 below).

The lookahead phase for s' with $h(s') < h(s)$, *[Breadth-First Search]* in Figure 1, is exactly the same as in EHC, except that it employs the dead-end cache; randomizes the expansion order during breadth-first search to achieve different behavior across episodes; and terminates early in case a relaxed plan turns out to be a real plan for some state. (The latter is required for completeness, with relaxation refinement, even under a depth limit k.)

Failure handling is triggered when the lookahead phase exhausted its search space without finding the desired state s'. If failure occurs on the initial state, then there is no point in running another EHC episode, and we terminate indicating whether or not a proof of unsolvability was found. On states other than the initial state, the failure handling encompasses three options, namely to give up (return "don't know"), to restart, or to backjump. Restart starts the new episode at the initial state, while a backjump starts the new episode at the most recent predecessor t of s where $h(t) \neq \infty$. Both are implemented through setting s_0 and π_0 accordingly, before **break**ing out of the EHC-episode loop to give back control to the outside loop, starting a new episode.

With helpful actions pruning, Episode-EHC is incomplete simply because H may prune the solutions. Without helpful actions pruning, matters are more interesting. The Giveup

configuration – original EHC – is complete only for input tasks where all dead-ends are recognized by h, i. e., $h(t) = \infty$ whenever t is a dead-end (Hoffmann 2003). However:

Proposition 2 *Episode-EHC without helpful actions pruning is complete when not using Giveup.*

Proof: Without helpful actions pruning, every EHC episode adds at least one new state into $\mathcal{C}_{de}$. So after at most N episodes, where N is the number of dead-end states, $\mathcal{C}_{de}$ contains all dead-ends. Hence EHC episode $N+1$ will either fail directly as $\mathcal{I} \in \mathcal{C}_{de}$, or will not fail and find a plan. ■

Refinement-HC introduces two major changes relative to Episode-EHC: a lookahead horizon $k \in \mathbb{N}$, and relaxation refinement. It assumes that h comes with a refinement operation ρ, and caters for the case where ρ is applicable only on states s where $h(s) < \infty$ and $\pi[h](s)$ is not a real plan.

The pseudo-code, Figure 1 (right), reuses the previous macros, with some changes. In breadth-first search, apart from the search depth limit k, the only change consists in remembering, for later use, the minimal heuristic value $h_{\min}$ encountered below s. The failure trigger is now different, as the lookahead bound means we no longer actually exhaust the search space below s. We therefore trigger a failure only if, in addition to there being no suitable state s' within depth k, the search (and its growing dead-end detection facilities) proved that there can be no such state at *any* depth, namely if the depth limit did not actually prune anything. In failure handling, we first refine h to ensure that at least one refinement step occurs in every episode; and we do not necessarily stop on the initial state, as further episodes will yield further refinements and hence further progress. The remainder of the failure handling is identical to that in Episode-EHC.

Next, consider the new elements of Refinement-HC. The main loop adds a refinement phase, consisting of refinement and stagnation steps. The refinement trigger is $h(s) \leq h_{\min}$, i. e., no better state is found within lookahead k. We are hence located on a local minimum or plateau escaping which by search is difficult (requires lookahead $> k$). We refine the relaxation to remove that difficulty. The refinement handling does so by iteratively refining h until $h(s) > h_{\min}$ (terminating early if $\pi[h](s)$ is a plan so no refinement is possible, nor needed). Note here that $h_{\min}$ in the comparison $h(s) > h_{\min}$ is static, comparing the new refined $h(s)$ against the minimal value found below s with the old unrefined h. This is because validating every intermediate h version through a search would be too costly. Such a validation is instead done at the end of *[Refinement Handling]*, through the **continue** command which hands control back to the same episode, and thus to the search below s.

Stagnation is what we call the event where that search still does not find a better state: while the refined h has increased $h(s)$, it has increased the minimum h value below s to a similar degree. One may choose to continue the alternating search/refinement on s until eventually either search is successful, or h is so refined as to cause termination (cf. Theorem 1 below). We call this stagnation handling option *StagContinue*; it forces the algorithm to escape s through refinement, which can be ineffective in practice. So we provide two alternate options starting a new episode instead. *StagRestart* is identical to *Restart* in failure handling. Similarly for *StagBackjump*, except that the state backjumped to is different: both *Backjump* and *StagBackjump* jump back to the most recent predecessor where *the reason for backjumping has disappeared*. In failure handling, the reason is being a (necessarily) failed state; in stagnation handling, the reason is unsuccessful search. Hence *StagBackjump* tests the ancestors of s, in reverse order, by search.

Regarding completeness, obviously the Giveup configuration is out of the question. Remarkably though, this is the *only* incomplete configuration option:

Theorem 1 *Given a heuristic h converging with ρ, Refinement-HC is complete when not using Giveup.*

Proof: Observe that every episode refines h at least once, and that this sequence of refinements stops only if either (a) a plan is found, or (b) $\mathcal{I} \in \mathcal{C}_{de} \vee h(\mathcal{I}) = \infty$.

Say the input task Π is unsolvable. Then (a) never happens, and termination on (b) is an unsolvability proof as desired. Unless termination on (b) happens earlier, h will eventually converge, at which point (Definition 1 (i)) $h(s) = \infty$ for all unsolvable s, hence for all reachable ones. In both the Restart and Backjump configurations, we go back to $\mathcal{I}$, find that $h(\mathcal{I}) = \infty$, and terminate on (b).

Say now that Π is solvable. Then (b) never happens, and (a) is the desired termination. Unless that termination happens earlier, h will eventually converge, at which point $h(s) = \infty$ for all unsolvable s, and (Definition 1 (ii)) $\pi[h](s)$ is a plan for all solvable s. In both the Restart and Backjump configurations, we go back to a solvable state s, and find that $\pi[h](s)$ is a plan for s, so terminate on (a). ■

Recall that this theorem applies to an entire algorithm family, namely all 12 configurations of Refinement-HC not giving up in case of episode failure, times all values of k. In particular, for $k = 1$ with *Restart*, the search itself is just plain hill-climbing, restarting whenever no better successor state exists – yet made complete by relaxation refinement.

Completeness on unsolvable tasks relies only on convergence property (i). For completeness on solvable tasks, convergence to h^* would suffice to always find a better state within any horizon; (ii) is a weaker property that suffices thanks to early termination when $\pi[h](s)$ is a plan for s.

Online-Refinement GBFS

We also experimented with some variants of *greedy best-first search (GBFS)*, the most popular search algorithm in satisficing planning. Preempting our overall conclusion, online relaxation refinement in GBFS is beneficial, but not as much as in hill-climbing; further studies are needed to understand its behavior in more detail. We therefore provide, in what follows, only a brief summary of the methods we tried.

We assume that the reader is familiar with standard GBFS, whose global open list is a priority queue preferring states with small h values. For online relaxation refinement in this setting, we maintain the set S_0 of visited states s where $h(s) = h_0$, h_0 being the minimum h value encountered so far with the current h. We trigger refinement when search has proved that S_0 has no better depth-$\leq k$ descendant. We

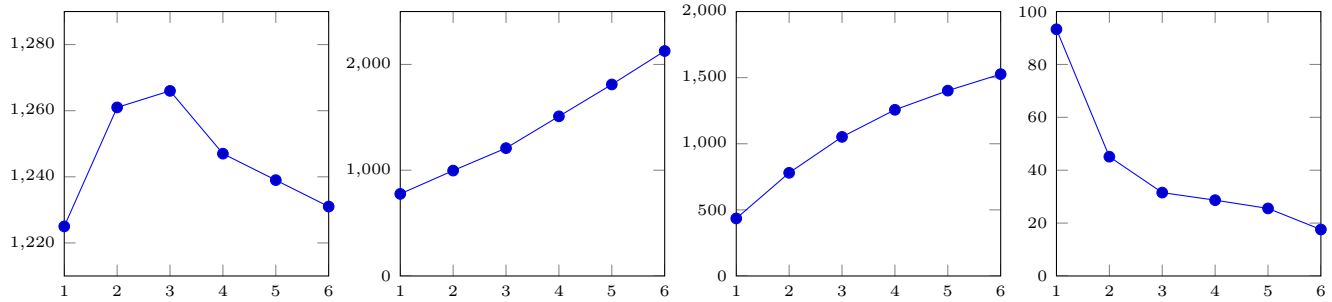

Figure 2: Performance of Refinement-HC using *Restart* and *StagContinue*, for scaling k. From left to right: overall coverage, geometric mean of the number of evaluations for commonly solved instances, geometric mean of the number of evaluated states per second, and geometric mean of the number of refinement phases for problems where refinement was done at least once.

apply a single refinement step, learning a single new conjunction only. We also experimented with other variants refining more aggressively, e. g. until at least one $s \in S_0$ does have a better depth-$\leq k$ descendant; but these variants were, generally, inferior due to the computational overhead.

We remark that, to obtain completeness under helpful actions pruning, i. e., when limiting state expansions to helpful actions only, restarting or node reopening is required. For example, the only solution may be pruned by $H(\mathcal{I})$ already, so if $\mathcal{I}$ is not revisited then relaxation refinement will never correct that mistake. In other words, the full power of online relaxation refinement unfolds only if one allows to revisit previous states in the light of these refinements.

Experiments

We implemented the described algorithms in Fast Downward (FD) (Helmert 2006). The experiments were run on machines with Intel Xeon E5-2660 processors with a clock rate of 2.2 GHz. The time and memory limits were set to 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 EHC with a depth bound of 2 using standard h^{FF} solves all instances. This results in a total of 1465 benchmark instances. All experiments with HC variants use helpful actions pruning, all experiments with GBFS use FD's lazy dual-queue search with preferred operators. Both h^{FF} and $h^{C\text{FF}}$ extract the relaxed plans using h^{add} (Bonet and Geffner 2001), with random tie-breaking, which was overall the best configuration in prior work (Fickert, Hoffmann, and Steinmetz 2016) and also in our context. The results we show are averaged over three random seeds. For the coverage values, an instance counts as solved if it was solved with at least two random seeds.

Given the overall observations regarding GBFS stated earlier, we concentrate mainly on Refinement-HC. We start by analyzing the impact of its configuration parameters. We then compare a subset of Refinement-HC configurations, as well as selected GBFS variants, to baselines and the state of the art. We conclude with a direct assessment of the "quality" of atomic conjunctions learned online vs. offline.

Refinement-HC Configurations

We first examine the depth bound parameter k, controlling the trade-off between learning (small k) and search (large k). We let k range between 1 and 6. We fix the other algorithm parameters to a canonical configuration of Refinement-HC, using *Restart* upon failure, and using *StagContinue* in stagnation handling. Consider Figure 2.

As expected, for smaller values of k the heuristic is more accurate as many conjunctions are learned. With increasing k, the accuracy of the heuristic decreases and more evaluations are required to find a solution. However, the computational overhead for relaxation refinement decreases as well. The sweet spot of this trade-off is at $k = 3$, with a peak coverage of 1266 solved instances.

There are some outlier domains that don't follow this pattern, where instead coverage consistently increases or decreases as a function of increasing k. The most extreme case where relaxation refinement is detrimental is the Barman domain, where coverage increases from 5 to 40 with increasing k. At the other extreme end, the domains that stand out are Transport and Tetris where, with increasing k, coverage drops from 70 to 41 respectively from 19 to 8.

We next examine Refinement-HC's other configuration parameters, for the two best-performing values of k, i. e., $k \in \{2, 3\}$. We omit the *Giveup* setting as *Restart* and *Backjump* are consistently better. Table 1 gives an overview.

k	Failure	Stagnation	Cov.	Eval.	Ref.
2	Restart	StagContinue	1261	579	63
		StagRestart	1084	868	90
		StagBackjump	1227	597	69
	Backjump	StagContinue	1209	549	102
		StagRestart	1091	846	116
		StagBackjump	1209	541	101
3	Restart	StagContinue	**1266**	684	32
		StagRestart	1156	877	36
		StagBackjump	1254	689	33
	Backjump	StagContinue	1212	659	59
		StagRestart	1165	820	57
		StagBackjump	1218	635	58

Table 1: Overview of the effects of Refinement-HC algorithm parameters: overall coverage ("Cov."), geometric mean of number of evaluations over commonly solved instances ("Eval."), and the geometric mean of the number of refinement phases for benchmark instances where refinement was done at least once ("Ref.").

The overall best configuration is the canonical one as already used in Figure 2, using *Restart* failure handling and *StagContinue* stagnation handling, with $k = 3$. For the stag-

nation case, restarting proves to be an overreaction, both alternate options performing significantly better.

The comparison between *Restart* and *Backjump* failure handling is interesting in that, while *Restart* typically has better overall coverage, the two methods are quite complementary on a per-domain basis. Table 2 illuminates this.

Backjump		**Restart**	
Domain	**ΔCov.**	**Domain**	**ΔCov.**
Barman	+1	Airport	+20
CityCar	+10	Cavediving	+7
Parking	+6	ChildSnack	+3
Storage	+3	Freecell	+1
Tetris	+4	Nomystery	+1
Transport	+4	Pathways	+7
Tidybot	+1	Pegsol	+33
Trucks	+1	Pipes-NoTank	+ 1
		Sokoban	+2
		Thoughtful	+8
		Visitall	+1
Sum	+30	**Sum**	+84

Table 2: Coverage comparison of the Refinement-HC *Backjump* vs. *Restart* failure handling options. Domains where *Backjump* has higher coverage shown on the left, domains where *Restart* has higher coverage shown on the right. Other parameters fixed to *StagContinue* and $k = 3$.

The superiority of *Restart* in terms of overall coverage is mainly due to Airport and Pegsol. In terms of the number of domains with superior performance – arguably a more reliable measure – the comparison is closer, with 7 respectively 11 domains where *Backjump* respectively *Restart* is better.

Intuitively, *Backjump* is preferrable in domains where bad choices are easy to fix, whereas *Restart* is preferrable in domains where mistakes are often made early on and are easier to correct by starting from scratch. It is presumably possible to adjust this behavior correctly in a per-domain training phase; and we hypothesize that it is even possible to predict the adequate behavior during search, from search statistics and/or features of the planning task at hand. Assuming a perfect per-domain adjustment, overall coverage could be further increased by 30, for a hypothetical total of 1296.

Baselines and State of the Art

We now turn to the comparison of our techniques with competing planning algorithms. Table 3 shows coverage results. We compare our best-performing configurations (HC "onl." and GBFS "onl." in the table) to the most closely related heuristic functions, to FF (Hoffmann and Nebel 2001) as the canonical fix for EHC's incompleteness, as well as to the state of the art in satisficing planning. The latter is represented here by LAMA (Richter and Westphal 2010) and Mercury (Domshlak, Hoffmann, and Katz 2015). As the most closely related heuristic functions, we run h^{FF}, as well as $h^{C\text{FF}}$ with conjunctions generated offline, using Keyder et al.'s (2014) C-learning method with a growth bound of 1.5 and a timeout of 15 minutes (the most competitive parameter setting in our experiments). In the comparison for HC using these heuristic functions, we run Episode-EHC using *Restart* failure handling, the most closely related configuration of HC without online relaxation refinement.

Search	HC			GBFS					
Heuristic	h^{FF}	$h^{C\text{FF}}$		h^{FF}	$h^{C\text{FF}}$		FF	LAMA	Mercury
C**-Learning**	–	offl.	onl.	–	offl.	onl.			
Airport 50	24	38	**43**	35	37	38	35	32	32
Barman 40	36	29	36	26	4	4	36	39	**40**
Blocks 35	**35**	**35**	**35**	**35**	**35**	**35**	**35**	**35**	**35**
Cavediving 20	0	0	**7**	**7**	**7**	**7**	**7**	**7**	**7**
ChildSnack 20	**7**	**7**	3	0	2	0	0	5	0
CityCar 20	0	0	10	6	7	**16**	1	5	5
Depots 22	18	**22**	**22**	20	21	**22**	19	20	21
DriverLog 20	11	17	**20**	**20**	**20**	**20**	**20**	**20**	**20**
Elevators 50	**50**	47	**50**	**50**	**50**	**50**	**50**	**50**	**50**
Floortile 40	6	12	**40**	10	13	**40**	8	8	8
Freecell 80	**80**	**80**	73	79	79	79	**80**	79	**80**
GED 20	**20**	18	**20**	**20**	**20**	**20**	**20**	**20**	**20**
Grid 5	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
Hiking 20	0	1	**20**	**20**	**20**	**20**	18	18	**20**
Logistics 63	60	**63**	60	**63**	**63**	**63**	60	**63**	**63**
Maintenanc 20	**17**	**17**	**17**	10	10	**17**	11	0	7
Mprime 35	**35**	**35**	**35**	**35**	**35**	**35**	**35**	**35**	**35**
Mystery 19	16	**19**	**19**	18	**19**	**19**	17	**19**	**19**
Nomystery 20	13	12	13	9	5	**15**	8	11	14
Openstacks 40	**40**	10	**40**	**40**	34	**40**	**40**	**40**	**40**
Parcprinter 50	49	49	**50**	35	43	**50**	38	49	**50**
Parking 40	20	19	27	36	29	34	19	**40**	**40**
Pathways 30	**30**	**30**	**30**	21	23	21	23	23	**30**
Pegsol 50	21	33	**50**	**50**	**50**	48	**50**	**50**	**50**
Pipes-NT 50	42	41	**45**	41	42	**45**	36	43	44
Pipes-T 50	37	40	**43**	38	40	**43**	38	42	42
PSR 50	0	11	**50**	**50**	**50**	**50**	**50**	**50**	**50**
Rovers 40	**40**	39	**40**	**40**	**40**	**40**	**40**	**40**	**40**
Satellite 36	**36**	**36**	**36**	**36**	35	**36**	**36**	**36**	**36**
Scanalyzer 50	**50**	**50**	**50**	46	**50**	46	**50**	**50**	**50**
Sokoban 50	0	0	5	46	42	27	**48**	**48**	42
Storage 30	8	11	**27**	21	21	22	20	19	19
Tetris 20	1	2	15	18	17	6	14	13	**19**
Thoughtful 20	**19**	**19**	**19**	9	12	13	15	16	13
Tidybot 20	15	17	16	17	17	**18**	17	17	15
TPP 30	28	28	**30**	**30**	**30**	**30**	28	**30**	**30**
Transport 70	31	32	60	47	44	54	31	61	**70**
Trucks 30	22	19	16	18	16	**25**	14	15	19
VisitAll 40	7	5	19	25	19	18	7	**40**	**40**
Woodwork 50	5	**50**	**50**	**50**	**50**	**50**	36	**50**	**50**
Zenotravel 20	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**
Sum 1465	954	1018	1266	1202	1176	1241	1135	1263	**1290**

Table 3: Coverage. Comparison of our best configuration of Refinement-HC (*Restart*, *StagContinue*, $k = 3$) and online-learning GBFS to the state of the art.

Consider first the comparison of HC algorithms, in the left part of the table. Online relaxation refinement yields a major boost here, improving overall coverage by $+244$ compared to offline $h^{C\text{FF}}$, and by $+312$ compared to h^{FF}. It yields strictly better coverage than both competing methods in 18 domains, has equally good coverage as the best competing method in 18 domains, and has worse coverage than at least one competing method in only 5 domains. We consider this a vivid demonstration that escaping local minima and plateaus through refining the relaxation, rather than through search alone, can be beneficial.

This strong picture persists even in the comparison against the GBFS algorithms, in the middle part of Table 3, though not to quite such an extent. Online heuristic function refinement in EHC beats GBFS with h^{FF} and offline $h^{C\text{FF}}$, improving overall coverage by $+64$ respectively $+90$; it is superior to GBFS with h^{FF} in 17 domains, equal in 16, worse in 8; it is superior to GBFS with offline $h^{C\text{FF}}$in 17

domains, equal in 18, worse in 6. Slightly weaker performance improvements result from using online relaxation refinement in GBFS itself. Observe that, while Airport, Floortile, Parcprinter, and Thoughtful are the strongest domains for offline $h^{C\text{FF}}$ refinement as per Keyder et al. (2012; 2014), i. e., for GBFS with vs. without such refinement, it is outclassed in all these domains by online $h^{C\text{FF}}$ refinement.

Overall, online $h^{C\text{FF}}$ refinement is a clear win for satisficing planning over both, the baseline h^{FF} and the previous offline $h^{C\text{FF}}$ refinement variants.

Consider finally the right part of Table 3, giving data for FF, which switches to GBFS if EHC fails, as well as LAMA and Mercury, which combine heuristic functions based on different principles (h^{FF} and landmarks, respectively a red-black plan heuristic and landmarks).

Refinement-HC outperforms FF, with $+131$ in overall coverage, and superior coverage in 22 domains, equal coverage in 6 domains, smaller coverage in only 3 domains.

Unsurprisingly, the highly engineered planners LAMA and Mercury are more difficult to beat. In overall coverage, Refinement-HC marginally beats LAMA $+3$, but loses to Mercury -24. Nevertheless, a per-domain comparison clearly shows the advantages of HC with online $h^{C\text{FF}}$ refinement. Compared to LAMA, our method is superior in 15 domains, equal in 17 domains, and inferior in 9 domains, so this comparison is clearly in favor of us. Compared to Mercury, our method is superior in 11 domains, equal in 20 domains, and inferior in 10 domains, so this comparison is roughly on par. In 9 domains, our method outperforms both, LAMA and Mercury. The coverage advantages are small in Depots and the Pipesworld domains, but are substantial in the other 6 domains: Airport $+11$, CityCar $+5$, Floortile $+32$, Maintenance $+17$ respectively $+10$, Storage $+8$, and Thoughtful $+3$ respectively $+6$.

The advantages in Maintenance and Thoughtful are due to HC rather than our online-refinement methods, as the other HC-based planners perform equally well in these domains. The HC boost thanks to these methods is, however, likely to make HC much more relevant again to state-of-the-art competitive planner configuration/portfolios.

Online vs. Offline Conjunctions Quality

Intuitively, as online learning is performed at a set of especially relevant states, rather than just at the initial state, the "quality" of the learned conjunctions should be better. This cannot soundly be concluded from the above though, as online vs. offline C-learning differ not only in *which* conjunctions are learned, but also in *when* and *how many*. A cleaner assessment is to compare conjunction sets C_1 and C_2 whose size is equal, $|C_1| = |C_2|$, and that are fixed throughout the search. We say that C_1 has better quality than C_2 if it provides better search guidance with $h^{C\text{FF}}$. As a canonical proxy for the latter, we use the search space size of GBFS.

To realize this kind of assessment, we save the set of conjunctions C^{on} generated online during a Refinement-HC run (of the configuration from Table 3). Subsequently, we obtain a corresponding offline set C^{off} by generating the same number of conjunctions through iterated refinement steps on the initial state. Figure 3 shows a per-instance comparison of the resulting GBFS search space sizes.

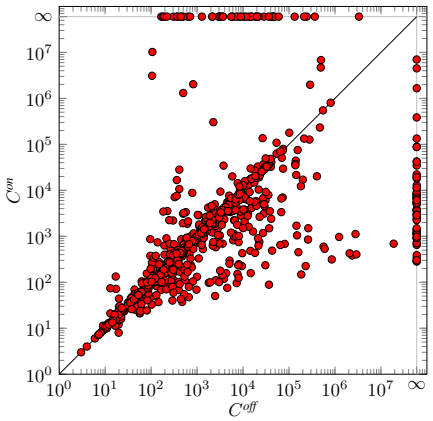

Figure 3: Comparing conjunctions learned online during Refinement-HC (C^{on}), vs. offline through refinement steps on the initial state (C^{off}). We show the number of state evaluations in GBFS with $h^{C\text{FF}}$ (∞ means unsolved).

Despite a lot of variance, C^{on} does tend to be in the advantage. Over instances solved commonly with both C^{on} and C^{off}, the geometric mean of the number of state evaluations is 738 for C^{on}, and 1138 for C^{off}. Moreover, the advantage of C^{on} is manifested in almost all domains, the only exceptions being Parcprinter and TPP.

Conclusion

Online relaxation refinement – revising the information basis of a heuristic function online during search, ultimately to convergence – has previously been all but neglected. We herein showed its pertinence for local search, removing undesirable search space surface phenomena on demand, thereby obtaining completeness even for vanilla variants of hill-climbing, and even when using incomplete helpful-actions pruning mechanisms. We furthermore showed that, instantiated with $h^{C\text{FF}}$, the approach yields a veritable performance boost for hill-climbing algorithms, bringing it on par with the state of the art in satisficing planning, substantially outperforming that state of the art in 6 IPC domains.

One important topic for future work is to be more intelligent about *which* new atomic conjunction is added in a refinement step. As Keyder et al. (2014) observed earlier, arbitrary code changes affecting that choice can have a large impact on performance. To make the choice intelligently, a promising possibility is to reward previous atomic conjunctions according to their observed importance in search, and to rank potential new conjunctions based on these rewards.

Beyond this, the big question is whether and how online relaxation refinement can be made successful for other heuristic functions and for other purposes. While Seipp's (2012) results are discouraging for Cartesian abstractions, we believe that there is hope yet, certainly for other types of abstractions, and even for Cartesian abstractions as Seipp explored only a small design space. In optimal planning in general, a pertinent question is that of the interaction between relaxation refinement and cost partitioning.

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

Akagi, Y.; Kishimoto, A.; and Fukunaga, A. 2010. On transposition tables for single-agent search and planning: Summary of results. In Felner, A., and Sturtevant, N. R., eds., *Proceedings of the 3rd Annual Symposium on Combinatorial Search (SOCS'10)*. Stone Mountain, Atlanta, GA: AAAI Press.

Arfaee, S. J.; Zilles, S.; and Holte, R. C. 2011. Learning heuristic functions for large state spaces. *Artificial Intelligence* 175(16-17):2075–2098.

Barto, A. G.; Bradtke, S. J.; and Singh, S. P. 1995. Learning to act using real-time dynamic programming. *Artificial Intelligence* 72(1-2):81–138.

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

Bonet, B., and Geffner, H. 2003. Labeled RTDP: Improving the convergence of real-time dynamic programming. In Giunchiglia, E.; Muscettola, N.; and Nau, D., eds., *Proceedings of the 13th International Conference on Automated Planning and Scheduling (ICAPS'03)*, 12–21. Trento, Italy: Morgan Kaufmann.

Clarke, E.; Grumberg, O.; Jha, S.; Lu, Y.; and Veith, H. 2003. Counterexample-guided abstraction refinement for symbolic model checking. *Journal of the Association for Computing Machinery* 50(5):752–794.

Clarke, E. M.; Grumberg, O.; and Long, D. E. 1994. Model checking and abstraction. *ACM Transactions on Programming Languages and Systems* 16(5):1512–1542.

Coles, A.; Fox, M.; and Smith, A. 2007. A new local-search algorithm for forward-chaining planning. In Boddy, M.; Fox, M.; and Thiebaux, S., eds., *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS'07)*, 89–96. Providence, Rhode Island, USA: Morgan Kaufmann.

Culberson, J. C., and Schaeffer, J. 1998. Pattern databases. *Computational Intelligence* 14(3):318–334.

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

Domshlak, C.; Karpas, E.; and Markovitch, S. 2012. Online speedup learning for optimal planning. *Journal of Artificial Intelligence Research* 44:709–755.

Edelkamp, S. 2001. Planning with pattern databases. In Cesta, A., and Borrajo, D., eds., *Proceedings of the 6th European Conference on Planning (ECP'01)*, 13–24. Springer-Verlag.

Felner, A.; Korf, R.; and Hanan, S. 2004. Additive pattern database heuristics. *Journal of Artificial Intelligence Research* 22:279–318.

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.

Fink, M. 2007. Online learning of search heuristics. In *Proceedings of the 11th International Conference on Artificial Intelligence and Statistics (AISTATS'07)*, 114–122.

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.; Botea, A.; Helmert, M.; Bonet, B.; and Koenig, S. 2007. Domain-independent construction of pattern database heuristics for cost-optimal planning. In Howe, A., and Holte, R. C., eds., *Proceedings of the 22nd National Conference of the American Association for Artificial Intelligence (AAAI'07)*, 1007–1012. Vancouver, BC, Canada: AAAI Press.

Haslum, P. 2006. Improving heuristics through relaxed search - an analysis of TP4 and HSP*a in the 2004 planning competition. *Journal of Artificial Intelligence Research* 25:233–267.

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

Helmert, M.; Haslum, P.; Hoffmann, J.; and Nissim, R. 2014. Merge & shrink abstraction: A method for generating lower bounds in factored state spaces. *Journal of the Association for Computing Machinery* 61(3).

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. 2003. *Utilizing Problem Structure in Planning: A Local Search Approach*, volume 2854 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag.

Humphrey, T.; Bramanti-Gregor, A.; and Davis, H. W. 1995. Learning while -solving problems in single agent search: Preliminary results. In *4th Congress of the Italian Association for Artificial Intelligence (AIIA'95)*, 56–66.

Karpas, E.; Katz, M.; and Markovitch, S. 2011. When optimal is just not good enough: Learning fast informative action cost-partitionings. In Bacchus, F.; Domshlak, C.; Edelkamp, S.; and Helmert, M., eds., *Proceedings of the*

21st International Conference on Automated Planning and Scheduling (ICAPS'11), 122–129. AAAI Press.

Katz, M., and Domshlak, C. 2010. Optimal admissible composition of abstraction heuristics. *Artificial Intelligence* 174(12–13):767–798.

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

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

Korf, R. E. 1990. Real-time heuristic search. *Artificial Intelligence* 42(2-3):189–211.

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

Seipp, J., and Helmert, M. 2013. Counterexample-guided Cartesian abstraction refinement. In Borrajo, D.; Fratini, S.; Kambhampati, S.; and Oddi, A., eds., *Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS'13)*, 347–351. Rome, Italy: AAAI Press.

Seipp, J. 2012. Counterexample-guided abstraction refinement for classical planning. Master's thesis, University of Freiburg, Germany.

Steinmetz, M., and Hoffmann, J. 2017. State space search nogood learning: Online refinement of critical-path dead-end detectors in planning. *Artificial Intelligence* 245:1 – 37.

Thayer, J. T.; Dionne, A. J.; and Ruml, W. 2011. Learning inadmissible heuristics during search. In Bacchus, F.; Domshlak, C.; Edelkamp, S.; and Helmert, M., eds., *Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS'11)*. AAAI Press.

Wilt, C. M., and Ruml, W. 2013. Robust bidirectional search via heuristic improvement. In desJardins, M., and Littman, M., eds., *Proceedings of the 27th AAAI Conference on Artificial Intelligence (AAAI'13)*. Bellevue, WA, USA: AAAI Press.