AI Planning

8. Critical Path Heuristics

It's a Long Way to the Goal, But How Long Exactly?
Part I: *Honing In On the Most Critical Subgoals*

Álvaro Torralba, Cosmina Croitoru

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Winter Term 2018/2019

Thanks to Prof. Jörg Hoffmann for slide sources

Agenda

We Need Heuristic Functions!

→ Critical path heuristics are a method to relax planning tasks, and thus automatically compute heuristic functions h.

We cover the 4 different methods currently known:

- Critical path heuristics: → **This Chapter**
- Delete relaxation: → **Chapters 9 and 10**
- Abstractions: → **Chapters 11-13**
- Landmarks: → **Chapter 14**
- LP Heuristics: → **Chapter 16**

→ Each of these have advantages and disadvantages. (We will do a formal comparison in **Chapter 17**.)

Critical Path Heuristics: Basic Idea

Our Agenda for This Chapter

2. **Critical Path Heuristics:** Introduces and illustrates the formal definition.
3. **Dynamic Programming Computation:** The straightforward method to compute critical path heuristics.
4. **Graphplan Representation:** A slightly less straigtforward method to compute critical path heuristics. I mention this here only because, historically, it was there first, and its terminology is all over the planning literature.
5. **What about FDR Planning?** The above uses STRIPS as this is a little easier to discuss in the examples. In this section, we point out on 1 slide that (almost) everything remains exactly the same for FDR.

A Regression-Based Characterization of h^*

Definition (r^*). *Let $\Pi = (P, A, c, I, G)$ be a STRIPS planning task. The perfect regression heuristic r^* for Π is the function $r^*(s) := r^*(s, G)$ where $r^*(s, g)$ is the point-wise greatest function*[1] *that satisfies $r^*(s, g) =$*

$$\begin{cases} 0 & g \subseteq s \\ min_{a \in A, regr(g,a) \neq \perp} c(a) + r^*(s, regr(g, a)) & \textit{otherwise} \end{cases}$$

(Reminder **Chapter 6**: $regr(g, a) \neq \perp$ if $add_a \cap g \neq \emptyset$ and $del_a \cap g = \emptyset$; then, $regr(g, a) = (g \setminus add_a) \cup pre_a$.)

→ The cost of achieving a subgoal g is 0 if it is true in s; else, it is the minimum of using any action a to achieve g.

Proposition. *Let $\Pi = (P, A, c, I, G)$ be a STRIPS planning task. Then $r^* = h^*$.* (Proof omitted.)

[1] "point-wise greatest" is needed here, and in the following, only to correctly handle 0-cost actions. We might bother you with an **Exercise** on this.

Critical Path Heuristics: h^1

Definition (h^1). *Let $\Pi = (P, A, c, I, G)$ be a STRIPS planning task. The critical path heuristic h^1 for Π is the function $h^1(s) := h^1(s, G)$ where $h^1(s, g)$ is the point-wise greatest function that satisfies $h^1(s, g) =$*

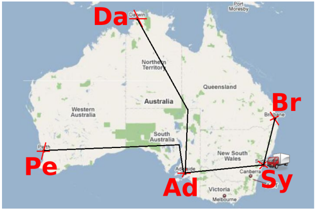
$$\begin{cases} 0 & g \subseteq s \\ min_{a \in A, regr(g,a) \neq \bot} c(a) + h^1(s, regr(g, a)) & |g| = 1 \\ \max_{g' \in g} h^1(s, \{g'\}) & |g| > 1 \end{cases}$$

→ For singleton subgoals g, use regression as in r^*. For subgoal sets g, use the cost of the most costly singleton subgoal $g' \in g$.

→ "Path" $= g_1 \xrightarrow{a_1} g_2 \dots g_{n-1} \xrightarrow{a_{n-1}} g_n$ where $g_1 \subseteq s$, $g_n \subseteq G$, $g_i \neq g_j$, and $g_i \subseteq regr(g_{i+1}, a_i)$. $|g_i| = 1$ here, $|g_i| \leq m$ for h^m (up next).

→ "Critical path" = Cheapest path through the most costly subgoals g_i.

The h^1 Heuristic in "TSP" in Australia



- P: $at(x)$ for $x \in \{Sy, Ad, Br, Pe, Ad\}$; $v(x)$ for $x \in \{Sy, Ad, Br, Pe, Ad\}$.
- A: $drive(x, y)$ where x, y have a road.

$$c(drive(x,y)) = \begin{cases} 1 & \{x,y\} = \{Sy, Br\} \\ 1.5 & \{x,y\} = \{Sy, Ad\} \\ 3.5 & \{x,y\} = \{Ad, Pe\} \\ 4 & \{x,y\} = \{Ad, Da\} \end{cases}$$

- I: $at(Sy), v(Sy)$; G: $at(Sy), v(x)$ for all x.

- $h^1(I) = h^1(I, G) = h^1(I, \{at(Sy), v(Sy), v(Ad), v(Br), v(Pe), v(Da)\}) =$
- $h^1(I, \{at(Sy)\}) = h^1(I, \{v(Sy)\}) =$
- $h^1(I, \{v(Da)\}) =$
- $h^1(I, \{at(Ad)\}) =$
- So $h^1(I, \{v(Da)\}) =$
- The critical path is?

Critical Path Heuristics: The General Case

Definition (h^m). *Let $\Pi = (P, A, c, I, G)$ be a STRIPS planning task, and let $m \in \mathbb{N}$. The critical path heuristic h^m for Π is the function $h^m(s) := h^m(s, G)$ where $h^m(s, g)$ is the point-wise greatest function that satisfies $h^m(s, g) =$*

$$\begin{cases} 0 & g \subseteq s \\ min_{a \in A, regr(g,a) \neq \perp} c(a) + h^m(s, regr(g, a)) & |g| \leq m \\ \max_{g' \subseteq g, |g'| = m} h^m(s, g') & |g| > m \end{cases}$$

→ For subgoal sets $|g| \leq m$, use regression as in r^*. For subgoal sets $|g| > m$, use the cost of the most costly m-subset g'.

→ Like h^1, basically just replace "1" with "m".

→ For fixed m, $h^m(s, g)$ can be computed in time polynomial in the size of Π. (See next section.)

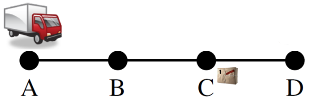
Critical Path Heuristics: Properties

Proposition (h^m is Admissible). *h^m is consistent and goal-aware, and thus also admissible and safe.*

Proof Sketch. Goal-awareness is obvious. We need to prove that $h^m(s) \leq h^m(s') + c(a)$ for all transitions $s \xrightarrow{a} s'$. Since $s \supseteq regr(s', a)$, a critical path $\vec{p}$ for $h^m(s')$ can be pre-fixed by a to obtain an upper bound on $h^m(s)$: *all* subgoals at the start of $\vec{p}$ are contained in s', and are achieved by a in s.

→ Intuition: h^m is admissible because it is always more difficult to achieve larger subgoals (so m-subsets can only be cheaper).

Questionnaire



- Initial state I: $t(A), p(C)$.
- Goal G: $t(A), p(D)$.
- Actions A: $drXY, loX, ulX$.

Question!

In this planning task, what is the value of $h^1(I)$?

(A): 2 (B): 3

(C): 4 (D): 5

Question!

In this planning task, what is the value of $h^2(I)$?

(A): 5 (B): 8

Overview

Basic idea:

"Consider all size-$\leq m$ subgoals g. Initialize $h^m(s, g)$ to 0 if $g \subseteq s$, and to ∞ otherwise.

Then keep updating the value of each g based on actions applied to the values computed so far, until the values converge."

- We start with an iterative definition of h^m that makes this approach explicit.
- We define a dynamic programming algorithm that corresponds to this iterative definition.
- We point out the relation to general fixed point mechanisms.

Iterative Definition of h^m

Definition (Iterative h^m). *Let $\Pi = (P, A, c, I, G)$ be a STRIPS planning task, and let $m \in \mathbb{N}$. The iterative h^m heuristic h^m_i is defined by $h^m_0(s, g) :=$*

$$\begin{cases} 0 & g \subseteq s \\ \infty & \textit{otherwise} \end{cases}$$

and $h^m_{i+1}(s, g) :=$

$$\begin{cases} \min[h^m_i(s, g), \min_{a \in A, regr(g,a) \neq \perp} c(a) + h^m_i(s, regr(g, a))] & |g| \leq m \\ \max_{g' \subseteq g, |g'|=m} h^m_{i+1}(s, g') & |g| > m \end{cases}$$

Proposition. *Let $\Pi = (P, A, c, I, G)$ be a STRIPS planning task. Then the series $\{h^m_i\}_{i=0,\ldots}$ converges to h^m.*

Proof Sketch: (i) Convergence: If $h^m_{i+1}(s, g) \neq h^m_i(s, g)$, then $h^m_{i+1}(s, g) < h^m_i(s, g)$; that can happen only finitely often because each decrease is due to a new path for g. (ii) If $h^m_{i+1} = h^m_i$ then h^m_i satisfies the h^m equation (direct from definition). (iii) No function greater than h^m_i at any point can satisfy the h^m equation (easy by induction over i).

Dynamic Programming

Dynamic Programming Algorithm

new table $T_0^m(g)$, for $g \subseteq P$ with $|g| \leq m$

For all $g \subseteq P$ with $|g| \leq m$: $T_0^m(g) := \begin{cases} 0 & g \subseteq s \\ \infty & \text{otherwise} \end{cases}$

fn $Cost_i(g) := \begin{cases} T_i^m(g) & |g| \leq m \\ \max_{g' \subseteq g, |g'|=m} T_i^m(g') & |g| > m \end{cases}$

fn $Next_i(g) := \min[Cost_i(g), \min_{a \in A, regr(g,a) \neq \bot} c(a) + Cost_i(regr(g,a))]$

$i := 0$

do forever:

    new table $T_{i+1}^m(g)$, for $g \subseteq P$ with $|g| \leq m$

    For all $g \subseteq P$ with $|g| \leq m$: $T_{i+1}^m(g) := Next_i(g)$

    if $T_{i+1}^m = T_i^m$ **then** stop **endif**

    $i := i + 1$

enddo

Proposition. $h_i^m(s,g) = Cost_i(g)$ *for all* i *and* g. (Proof is easy.)

$\rightarrow$ This is *very* inefficient! (Optimized for readability.) We can use "Generalized Dijkstra" instead, maintaining the frontier of cheapest m-tuples reached so far.

Just for the Record: Fixed Point Formulation

Fixed Point Algorithm – Template!

new table $T^m(g)$, for $g \subseteq P$ with $|g| \leq m$

For all $g \subseteq P$ with $|g| \leq m$: $T^m(g) := \begin{cases} 0 & g \subseteq s \\ \infty & \text{otherwise} \end{cases}$

fn $Cost(g) := \begin{cases} T^m(g) & |g| \leq m \\ \max_{g' \subseteq g, |g'|=m} T^m(g') & |g| > m \end{cases}$

fn $Next(g) := \min[Cost(g), \min_{a \in A, regr(g,a) \neq \bot} c(a) + Cost(regr(g,a))]$

while $\exists g \subseteq P, |g| \leq m : T^m(g) \neq Next(g)$ **do**:

  select one such g

  $T^m(g) := Next(g)$

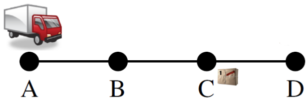
endwhile

Proposition. *Once the algorithm stops,* $h^m(s,g) = Cost(g)$ *for all* g.

Proof Sketch: Similar to that for convergence of h^m_i to h^m.

$\rightarrow$ This algorithm is not fully specified (hence "template"): How to select g s.t. $T^m(g) \neq Next(g)$? We will use dynamic programming for simplicity.

Example: $m = 1$ in "Logistics"



- Facts P: $t(x)\ x \in \{A, B, C, D\}$; $p(x)\ x \in \{A, B, C, D, T\}$.
- Initial state I: $\{t(A),\ p(C)\}$.
- Goal G: $\{t(A),\ p(D)\}$.
- Actions A (unit costs): $drive(x, y)$, $load(x)$, $unload(x)$.
  E.g.: $load(x)$: pre $t(x), p(x)$; add $p(T)$; del $p(x)$.

Content of Tables T_i^1:

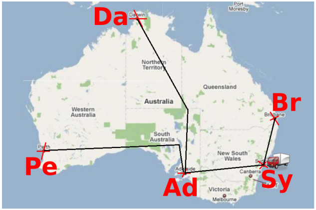
i	$t(A)$	$t(B)$	$t(C)$	$t(D)$	$p(T)$	$p(A)$	$p(B)$	$p(C)$	$p(D)$

$\rightarrow$ So $h^1(I) = 4$. (Cf. slide 13)

Note: This table computation always first finds the *shortest* path to achieve a subgoal g. Hence, with unit action costs, the value of g is fixed once it becomes $< \infty$, and equals the i where that happens. With non-unit action costs, neither is true.

Example: $m = 2$ in "Dompteur"

Example: $m = 1$ in "TSP" in Australia



- P: $at(x)$ for $x \in \{Sy, Ad, Br, Pe, Ad\}$; $v(x)$ for $x \in \{Sy, Ad, Br, Pe, Ad\}$.
- A: $drive(x, y)$ where x, y have a road.

$$c(drive(x,y)) = \begin{cases} 1 & \{x,y\} = \{Sy, Br\} \\ 1.5 & \{x,y\} = \{Sy, Ad\} \\ 3.5 & \{x,y\} = \{Ad, Pe\} \\ 4 & \{x,y\} = \{Ad, Da\} \end{cases}$$

- I: $at(Sy), v(Sy)$; G: $at(Sy), v(x)$ for all x.

Content of Tables T_i^1:

i	$at(Sy)$	$at(Ad)$	$at(Br)$	$at(Pe)$	$at(Da)$	$v(Sy)$	$v(Ad)$	$v(Br)$	$v(Pe)$	$v(Da)$

$\rightarrow$ So what is $h^1(I)$?

Example: $m = 2$ in Very Simple "TSP" in Australia

- Facts P: $at(Sy), at(Br), v(Sy), v(Br)$.
- Initial state I: $at(Sy), v(Sy)$.
- Goal G: $at(Sy), v(Sy), v(Br)$.
- Actions A: $drive(Sy, Br)$, $drive(Br, Sy)$; both cost 1.
  $drive(Sy, Br)$:
  pre $at(Sy)$; add $at(Br), v(Br)$; del $at(Sy)$.
  $drive(Br, Sy)$:
  pre $at(Br)$; add $at(Sy), v(Sy)$; del $at(Br)$.

Content of Tables T_i^2:

i	$at(Sy)$	$at(Br)$	$v(Sy)$	$v(Br)$	$at(Sy), at(Br)$	$at(Sy), v(Sy)$	$at(Sy), v(Br)$	$at(Br), v(Sy)$	$at(Br), v(Br)$	$v(Sy), v(Br)$

$\rightarrow$ So $h^2(I) = 2$, in contrast to $h^1(I) = 1$.

NOTE reg $at(Sy), v(Br))$ **in step** 1**:** Each of $at(Sy)$ and $v(Br)$ is reached, but not both together: $drive(Sy, Br)$ deletes $at(Sy)$ so we can't regress this subgoal over that action; $drive(Br, Sy)$ yields the regressed subgoal $\{at(Br), v(Br)\}$ whose value at iteration 0 is ∞.

Runtime

Proposition. *Let $\Pi = (P, A, c, I, G)$ be a STRIPS planning task, and let $m \in \mathbb{N}$ be fixed. Then the dynamic programming algorithm runs in time polynomial in the size of Π.*

Proof Sketch. With fixed m, the number of size-m fact sets is polynomial in the size of Π, so obviously each iteration of the algorithm runs in time polynomial in that size. The fixed point is reached at the latest at $i + 1 = |P|^m + 1$, as each path has length at most $|P|^m$.

$\rightarrow$ For any fixed m, h^m can be computed in polynomial time.

Remarks:

- In practice, only $m = 1, 2$ are used; higher values of m are infeasible.
- However! Instead of considering all "atomic subgoals" of size $\leq m$, one can select an arbitrary set C of atomic subgoals!

  $\rightarrow$ h^C, currently investigated in FAI, great results in learning to recognize dead-ends [Steinmetz and Hoffmann (2016)].

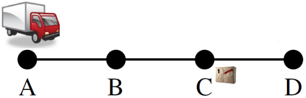
Graphplan Representation: The Case $m = 1$

1-Planning Graphs

$F_0 := s$; $i := 0$
while $G \not\subseteq F_i$ **do**
    $A_i := \{a \in A \mid pre_a \subseteq F_i\}$
    $F_{i+1} := F_i \cup \bigcup_{a \in A_i} add_a$
    if $F_{i+1} = F_i$ **then** stop **endif**
    $i := i + 1$
endwhile

Rings a bell?

1-Planning Graph for "Logistics"



- Initial state I: $t(A), p(C)$.
- Goal G: $t(A), p(D)$.
- Actions A: $dr(X, Y), lo(X), ul(X)$.

Content of Fact Sets F_i:

i	$t(A)$	$t(B)$	$t(C)$	$t(D)$	$p(T)$	$p(A)$	$p(B)$	$p(C)$	$p(D)$

→ Rings a bell?

1-Planning Graphs vs. h^1

Definition. *Let* $\Pi = (P, A, c, I, G)$ *be a STRIPS planning task. The* *1-planning graph heuristic* h^1_{PG} *for* Π *is the function* $h^1_{\mathsf{PG}}(s) := \min\{i \mid s \subseteq F_i\}$, *where* F_i *are the fact sets computed by a* 1*-planning graph, and the minimum over an empty set is* ∞.

Proposition. *Let* $\Pi = (P, A, c, I, G)$ *be a STRIPS planning task with unit costs. Then* $h^1_{\mathsf{PG}} = h^1$.

Proof Sketch: Induction over the value i of $h^1(s)$. Trivial for base case $i = 0$. For the step case, assume that $h^1_{\mathsf{PG}}(s) = h^1(s)$ for all s where $h^1(s) \leq i$, and show the same property for all s with $h^1(s) \leq i + 1$. $h^1_{\mathsf{PG}}(s) < i + 1$ directly contradicts the assumption. To show $h^1_{\mathsf{PG}}(s) \leq i + 1$, it suffices to observe that $h^1(pre_a) \leq i$ implies $h^1_{\mathsf{PG}}(pre_a) \leq i$ by assumption.

→ Intuition: A 1-planning graph is like our dynamic programming algorithm for $m = 1$, except that it represents not all facts but only those that have been reached (value $\neq \infty$), and instead of a fact-value table it only remembers that set.

Graphplan Representation: The General Case

m-Planning Graphs

$F_0 := s$; $M_0 := \emptyset$; $i := 0$

fn $Reached_i(g) := \begin{cases} True & g \subseteq F_i, \nexists g' \in M_i : g' \subseteq g \\ False & \text{otherwise} \end{cases}$

while not $Reached_i(G)$ **do**

$A_i := \{a \in A \mid Reached_i(pre_a)\}$

$F_{i+1} := F_i \cup \bigcup_{a \in A_i} add_a$

$M_{i+1} := \{g \subseteq P \mid |g| \leq m, \forall a \in A_i : \text{not } Reached_i(regr(g, a))\}$

if $F_{i+1} = F_i$ and $M_{i+1} = M_i$ **then** stop **endif**

$i := i + 1$

endwhile

→ Intuition: All m-subsets g of F_i are reachable within i steps, except for those g listed in M_i (the "mutexes").

→ Instead of listing the reached m-subsets, represent those that are not reached (and hope that there are fewer of those).

Critical Path Heuristics in FDR

... are exactly the same!

→ All definitions, results, and proofs apply, exactly as stated, also to FDR planning tasks. (See the single exception below.)

→ Remember (cf. → **Chapter 2**): We refer to pairs (v, d) of variable and value as facts. We identify partial variable assignments with sets of facts.

The single non-verbatim-applicable statement, adapted to FDR:

Proposition (h^m is Perfect in the Limit). *There exists m s.t. $h^m = h^*$.*

Proof. Given the definition of $regr(g, a)$ for FDR (→ **Chapter 6**), it is easy to see by induction that every subgoal g contains at most one fact for each variable $v \in V$. Thus, if we set $m := |V|$, then the case $|g| > m$ will never be used, so $h^m = r^*$.

→ In FDR, it suffices to set m to the number of *variables*, as opposed to the number of *variable values* i.e. STRIPS facts, compare slide 12!

Summary

- The critical path heuristics h^m estimate the cost of reaching a subgoal g by the most costly m-subset of g.
- This is admissible because it is always more difficult to achieve larger subgoals.
- h^m can be computed using dynamic programming, i.e., initializing true m-subsets g to 0 and false ones to ∞, then applying value updates until convergence.
- This computation is polynomial in the size of the planning task, given fixed m. In practice, $m = 1, 2$ are used; $m > 2$ is typically infeasible.
- Planning graphs correspond to dynamic programming with unit costs, using a particular representation of reached/unreached m-subsets g.

Historical Remarks

- The first critical path heuristic was introduced in the Graphplan system [Blum and Furst (1997)], which uses h^2 computed by a 2-planning graph.[2]
- 1-planning graphs are commonly referred to as relaxed planning graphs. This is because they're identical to Graphplan's 2-planning graphs when ignoring the delete lists [Hoffmann and Nebel (2001)].
- Graphplan spawned a huge amount of follow-up work [e.g., Kambhampati *et al.* (1997); Koehler *et al.* (1997); Koehler (1998); Kambhampati (2000)]; in particular, it was my personal "kindergarden planner".
- Nowadays, h^m is not in wide use anymore; its most prominent application right now is in modified forms that allow to use arbitrary sets of atomic subgoals (see slide 36), or to compute improved delete-relaxation heuristics ($\rightarrow$ **Chapter 10**).

[2] Actually, Graphplan does parallel planning (a simplistic form of temporal planning), and uses a version of 2-planning graphs reflecting this. I omit the details since parallel planning is not relevant in practice.

A Technical Remark

Reminder: Search Space for Progression

- start$() = I$
- succ$(s) = \{(a, s') \mid \Theta_\Pi$ has the transition $s \xrightarrow{a} s'\}$

$\rightarrow$ Need to compute $h^m(s) = h^m(s, G) \Rightarrow$ one call of dynamic programming for every different search state s!

Reminder: Search Space for Regression

- start$() = G$
- succ$(g) = \{(a, g') \mid g' = regr(g, a)\}$

$\rightarrow$ Need to compute $h^m(I, g) = \max_{g' \subseteq g, |g'|=m} h^m(I, g') \Rightarrow$ a single call of dynamic programming, for $s = I$ before search begins!

$\rightarrow$ For $m = 1$, it is feasible to use progression and recompute the cost of the (singleton) subgoals in every search state s. For $m = 2$ already, this is completely infeasible; all systems using h^2 do regression search, where all subgoals can be evaluated relative to the dynamic programming outcome for I.

Reading

- *Admissible Heuristics for Optimal Planning* [Haslum and Geffner (2000)].

  Available at:

  `http://www.dtic.upf.edu/~hgeffner/html/reports/admissible.ps`

  Content: The original paper defining the h^m heuristic function, and comparing it to the techniques previously used in Graphplan.

- $h^m(P) = h^1(P^m)$: *Alternative Characterisations of the Generalisation from* $h^{\max}$ *to* h^m [Haslum (2009)].

  Available at: `http://users.cecs.anu.edu.au/~patrik/publik/pm4p2.pdf`

  Content: Shows how to characterize h^m in terms of h^1 in a compiled planning task that explicitly represents size-m conjunctions.

  Relevance here: this contains the only published account of the iterative h^m_i characterization of h^m. Relevance more generally: yields an alternative computation of h^m. This is not per se useful, but variants thereof have been shown to allow the computation of powerful partial-delete-relaxation heuristics (→ **Chapter 10**).

Reading, ctd.

- *Explicit Conjunctions w/o Compilation: Computing $h^{FF}(\Pi^C)$ in Polynomial Time* [Hoffmann and Fickert (2015)].

  Available at:

  http://fai.cs.uni-saarland.de/hoffmann/papers/icaps15b.pdf

  Content: Introduces the h^C heuristic (cf. slide 23), which allows to select an arbitrary set C of atomic subgoals, and thus strictly generalizes h^m.

  This is only a side note in the paper though, the actual concern is with defining and computing partial-delete-relaxation heuristics on top of h^C.

Reading, ctd.

- *Towards Clause-Learning State Space Search: Learning to Recognize Dead-Ends* [Steinmetz and Hoffmann (2016)].

  Available at:

  `http://fai.cs.uni-saarland.de/hoffmann/papers/aaai16.pdf`

  Content: Specifies how to "learn" the atomic subgoals C based on states s where the search already knows that $h^*(s) = \infty$, yet where $h^C(s) \neq \infty$. The learning process adds new conjunctions into C, in a manner guaranteeing that $h^C(s) = \infty$ afterwards.

  Doing this systematically in a depth-first search, we obtain a framework that approaches the elegance of clause learning in SAT, finding and analyzing conflicts to learn knowledge that generalizes to other search branches.

References I

Avrim L. Blum and Merrick L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90(1–2):279–298, 1997.

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

Patrik Haslum. $h^m(P) = h^1(P^m)$: Alternative characterisations of the generalisation from $h^{\max}$ to h^m. In Alfonso Gerevini, Adele Howe, Amedeo Cesta, and Ioannis Refanidis, editors, *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS'09)*, pages 354–357. AAAI Press, 2009.

Jörg Hoffmann and Maximilian Fickert. Explicit conjunctions w/o compilation: Computing $h^{\mathsf{FF}}(\Pi^C)$ in polynomial time. In Ronen Brafman, Carmel Domshlak, Patrik Haslum, and Shlomo Zilberstein, editors, *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS'15)*. AAAI Press, 2015.

References II

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

Subbarao Kambhampati, Eric Parker, and Eric Lambrecht. Understanding and extending Graphplan. In S. Steel and R. Alami, editors, *Proceedings of the 4th European Conference on Planning (ECP'97)*, pages 260–272. Springer-Verlag, 1997.

Subbarao Kambhampati. Planning graph as a (dynamic) CSP: Exploiting EBL, DDB and other CSP search techniques in graphplan. *Journal of Artificial Intelligence Research*, 12:1–34, 2000.

Jana Koehler, Bernhard Nebel, Jörg Hoffmann, and Yannis Dimopoulos. Extending planning graphs to an ADL subset. In S. Steel and R. Alami, editors, *Proceedings of the 4th European Conference on Planning (ECP'97)*, pages 273–285. Springer-Verlag, 1997.

Jana Koehler. Planning under resource constraints. In H. Prade, editor, *Proceedings of the 13th European Conference on Artificial Intelligence (ECAI'98)*, pages 489–493, Brighton, UK, August 1998. Wiley.

References III

Marcel Steinmetz and Jörg Hoffmann. Towards clause-learning state space search: Learning to recognize dead-ends. In Dale Schuurmans and Michael Wellman, editors, *Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI'16)*. AAAI Press, February 2016.