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: \( h^1 \)

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

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

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

→ “Critical path” = Cheapest path through the most costly subgoals \( g_i \).

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 \( \Pi \) is the function \( r^*(s, G) = \)

\[
\begin{cases}
0 & g \subseteq s \\
\min_{a \in A, \text{regr}(g, a) \neq \bot} c(a) + r^*(s, \text{regr}(g, a)) & \text{otherwise}
\end{cases}
\]

(\text{Reminder Chapter 6:} \( \text{regr}(g, a) \neq \bot \) if \( \text{add}_a \cap g \neq \emptyset \) and \( \text{del}_a \cap g = \emptyset \); then, \( \text{regr}(g, a) = (g \setminus \text{add}_a) \cup \text{pre}_a \).)

→ 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.

The \( h^1 \) Heuristic in “TSP” in Australia

- \( P \): \( \text{at}(x) \) for \( x \in \{\text{Sy}, \text{Ad}, \text{Br}, \text{Pe}, \text{Ad}\} ; \text{v}(x) \) for \( x \in \{\text{Sy}, \text{Ad}, \text{Br}, \text{Pe}, \text{Ad}\} \).
- \( A \): \( \text{drive}(x, y) \) where \( x, y \) have a road.
- \( c(\text{drive}(x, y)) = \)
  \[
  \begin{cases}
  1 & \{x, y\} = \{\text{Sy}, \text{Br}\} \\
  1.5 & \{x, y\} = \{\text{Sy}, \text{Ad}\} \\
  3.5 & \{x, y\} = \{\text{Ad}, \text{Pe}\} \\
  4 & \{x, y\} = \{\text{Ad}, \text{Br}\}
  \end{cases}
  \]
- \( I \): \( \text{at}(\text{Sy}), \text{v}(\text{Sy}); \text{at}(\text{Sy}), \text{v}(\text{x}) \) for all \( x \).

- \( h^1(I = h^1(I, G) = h^1(I, \{\text{at}(\text{Sy}), \text{v}(\text{Sy}), \text{v}(\text{Ad}), \text{v}(\text{Br}), \text{v}(\text{Pe}), \text{v}(\text{Da})\}) = \)
  \[
  \begin{cases}
  h^1(I = \{\text{at}(\text{Sy})\}) = h^1(I, \{\text{at}(\text{Sy})\}) = \\
  h^1(I, \{\text{at}(\text{Ad})\}) = \\
  h^1(I, \{\text{at}(\text{Br})\}) = \\
  h^1(I, \{\text{at}(\text{Pe})\}) = \\
  h^1(I, \{\text{at}(\text{Da})\}) = \\
  \text{So } h^1(I, \{\text{v}(\text{Da})\}) = \\
  \text{The critical path is?}
  \end{cases}
  \]
Introduction

Critical Path Heuristics

Dynamic Programming

Graphplan

FDR

Conclusion

References

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 \( \Pi \) is the function \( h^m(s, g) \) that satisfies \( h^m(s, g) \) is the point-wise greatest function that satisfies

\[
\begin{align*}
0 & \quad g \subseteq s \\
\min_{a \in A, \text{regr}(g, a) \neq \bot} c(a) + h^m(s, \text{regr}(g, a)) & \quad |g| \leq m \\
\max_{g' \subseteq g, |g'| = m} h^m(s, g') & \quad |g| > m
\end{align*}
\]

→ 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 \( \Pi \). (See next section.)

Álvaro Torralba, Cosmina Croitoru

AI Planning

Chapter 8: Critical Path Heuristics

11/40

Overview

Basic idea:

“Consider all size-\( \leq m \) subgoals \( g \). Initialize \( h^m(s, g) \) to 0 if \( g \subseteq s \), and to \( \infty \) 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.

Álvaro Torralba, Cosmina Croitoru

AI Planning

Chapter 8: Critical Path Heuristics

15/40
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 is defined by \( h^0_i(s, g) := \)

\[
\begin{cases} 
0 & g \subseteq s \\
\infty & \text{otherwise}
\end{cases}
\]

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

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

**Proposition.** Let \( \Pi = (P, A, c, I, G) \) be a STRIPS planning task. Then the series \( \{h^m_i\}_{i=0}^\infty \) 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 \( h^m_{i+1} = h^m \), then \( h^m \) satisfies the \( h^m \) equation (direct from definition). (iii) No function greater than \( h^m \) at any point can satisfy the \( h^m \) equation (easy by induction over \( i \)).

---

**Fixed Point Algorithm – Template!**

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

For all \( g \subseteq P \) with \( |g| \leq m \): \( T^m(g) := \)

\[
\begin{cases} 
0 & g \subseteq s \\
\infty & \text{otherwise}
\end{cases}
\]

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

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

while \( \exists g \subseteq P, |g| \leq m \)

\( T^m(g) \neq \text{Next}(g) \) do:

select one such \( g \)

\( T^m(g) := \text{Next}(g) \)

**Proposition.** Once the algorithm stops, \( h^m(s, g) = \text{Cost}(g) \) for all \( g \).

**Proof Sketch:** Similar to that for convergence of \( h^m \) to \( h^m \).

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

---

**Example: \( m = 1 \) in “Logistics”**

- Facts \( P \): \( t(x) \in \{A, B, C, D\} \), \( p(x) \in \{A, B, C, D\} \).
- Initial state \( I \): \( \{t(A), p(C)\} \).
- Goal \( G \): \( \{t(A), p(D)\} \).
- Actions \( A \) (unit costs): \( \text{drive}(x,y), \text{load}(x), \text{unload}(x) \).

E.g.: \( \text{load}(x) \): pre \( t(x), p(x) \); add \( p(T) \); del \( p(x) \).

**Content of Tables \( T^1_A \):**

\[
\begin{array}{cccccccc}
1 & t(A) & t(B) & t(C) & t(D) & p(T) & p(A) & p(B) & p(C) & p(D) \\
\end{array}
\]

\( \rightarrow \) So \( h^1(A) = 4 \). (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

→ So $h^2(I) = 2$, in contrast to $h^3(I) = 1$.

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

Example: $m = 1$ in “TSP” in Australia

→ So what is $h^3(I)$?

Runtime

Proposition. Let $II = (P, A, c, I, G)$ 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 II.

Proof Sketch. With fixed $m$, the number of size-$m$ fact sets is polynomial in the size of II, 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 $|P|^m$.

→ 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!

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

Introduction

1-Planning Graphs

1-Planning Graphs

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

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

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

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

while $G \not\subseteq F_i$ do

$A_i := \{a \in A | pre_a \subseteq F_i\}$

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

if $F_{i+1} = F_i$ then stop

$i := i + 1$

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).

Graphplan Representation: The Case $m = 1$

1-Planning Graph for “Logistics”

→ Rings a bell?
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** \((h^m\text{ is Perfect in the Limit})\). There exists an \(m\) s.t. \(h^m = h^*\).

**Proof.** Given the definition of \(\text{regr}(g, a)\) 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!

### 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 “kindergarten 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 (→ 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.

### 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 \(\infty\), 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 \(g\).
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 $h^{\text{max}}$ 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).

- **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.

Reading, ctd.

- **Explicit Conjunctions w/o Compilation: Computing $h^{FF}(\Pi_C^C)$ 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 $h^C$.

References I


Patrik Haslum. $h^m(P) = h^1(P^m)$: Alternative characterisations of the generalisation from $h^{\text{max}}$ 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.


