AI Planning

17. Comparing Heuristic Functions

h^{foo} vs. h^{bar}: What's the Difference Anyway?

Álvaro Torralba, Cosmina Croitoru

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Winter Term 2018/2019

Thanks to Prof. Jörg Hoffmann for slide sources

Agenda

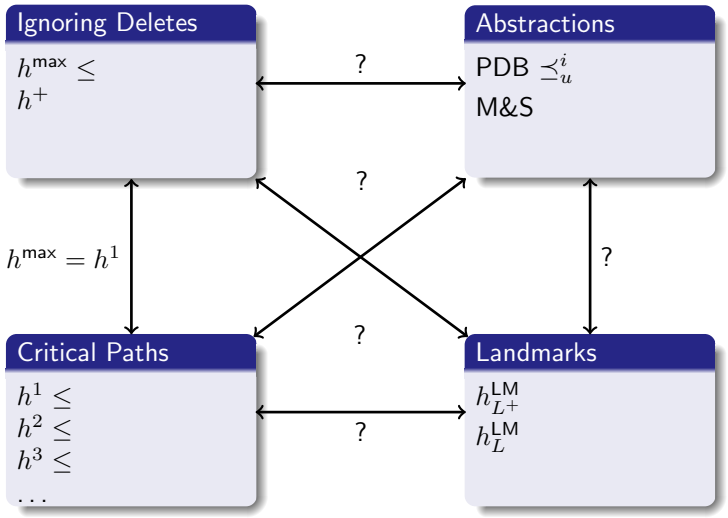
1. Introduction
2. The Compilability Framework
3. Example Proofs
4. A Walk Through the Zoo [for Reference]
5. Conclusion

"The Zoo"

All the wild (and admissible) animals we've learned about:

- h^1, h^2, h^3, ...: Critical path heuristics. h^m estimates remaining cost for s by the most costly size-m subgoals.
- h^+: Optimal delete relaxation heuristic. Estimates remaining cost for s by the cost of a cheapest plan for s in the delete-relaxed task.
- $h^{\max}$: The max heuristic. Estimates remaining cost for s by the most costly individual subgoal.
- $\{h^\alpha\}$: Abstraction heuristics, parameterized by abstraction mapping α. Estimate remaining cost for s by the remaining cost of $\alpha(s)$ within the abstract state space Θ^α. Classes of α we looked at are pattern databases (PDB) and merge-and-shrink abstractions (M&S).
- $\{h^{\text{LM}}_L\}$: Elementary landmark heuristics, parameterized by action set L. Estimate remaining cost for s by the cost of the cheapest action in L if L is a disjunctive action landmark for s, and by 0 otherwise.
- Elementary delete relaxation landmark heuristics $\{h^{\text{LM}}_{L^+}\}$: Cost of cheapest action in L^+ if L^+ is a *delete relaxation landmark*; 0 otherwise.

What Do We Know About "The Zoo"?



What Do We *Want* to Know About "The Zoo"?

Motivation for comparing heuristics: It's a big zoo: h^1, h^2, h^3, ..., h^+, $h^{\max}$, $\{h^\alpha \mid \alpha \text{ PDB}\}$, $\{h^\alpha \mid \alpha \text{ M\&S}\}$, $\{h^{\text{LM}}_L\}$, $\{h^{\text{LM}}_{L^+}\}$.

- Given any one planning task, which one should we use? Are perhaps some heuristics dominated by others?
- Are all these differences meaningful anyway? Or can we "simulate" some framework using another framework?

Restriction: We consider admissible heuristics only.

- Of lower bounds h and h', the bigger one is better.
- There is no similarly clear method to decide which one of two inadmissible heuristics (i.e., heuristics that are neither lower- nor upper-bounding) is "better".

Our Agenda for This Chapter

2. **The Compilability Framework:** We introduce, and discuss in detail, what it means for one family of heuristics to be able to "simulate" another family of heuristics.
3. **Example Proofs:** We give some concrete compilability and uncompilability proofs between particular families of heuristics as introduced in the previous chapters. In particular, we detail the aforementioned LM-cut heuristic, which was originally conceived as part of such a proof argument.
4. **A Walk Through the Zoo:** We briefly summarize the entire set of compilability results known about our "Zoo" of admissible heuristics, at this time.

Domination Between Heuristics

→ The only traditional means of comparing heuristic functions is by dominance:

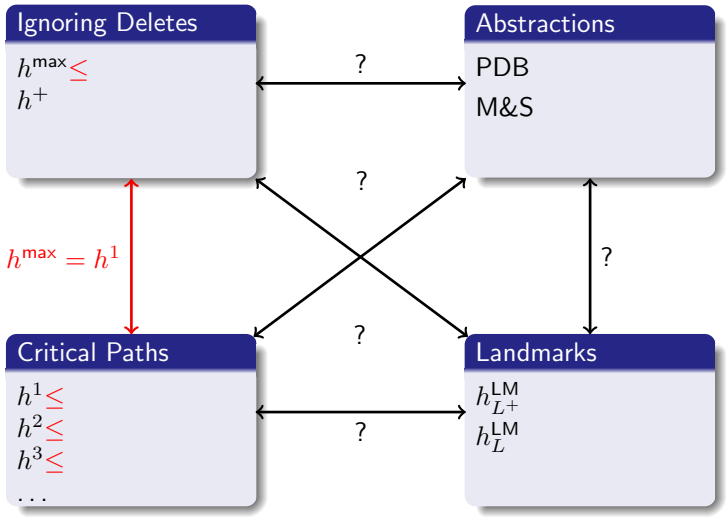
Reminder: → Chapter 7

Definition (Domination). *Let Π be a planning task, and let h and h' be admissible heuristics for Π. We say that h' dominates h if $h \leq h'$, i.e., for all states s in Π we have $h(s) \leq h'(s)$.*

→ This is a very limited framework because it only allows us to compare single heuristics, as opposed to classes of heuristics. See next slide.

What About "The Zoo"?

(Red: Conclusions from standard "domination" concept.)



Domination Between *Classes* of Heuristics?

→ To compare classes of heuristics, which question should we ask?

Ⓐ **For every $h \in H$, do all $h' \in H'$ dominate h?**

→ In "The Zoo": Not useful because we'll in most cases be able to choose a trivial $h' \in H'$. For example, (A) does not hold for H =PDB and H'=M&S because we can choose h' as a single-state abstraction.

Ⓑ **For every $h \in H$, does there exist $h' \in H'$ that dominates h?**

→ In "The Zoo": Not that bad. Captures relation between PDB and M&S. Some inter-family relations are also identifiable this way.

Ⓒ **For every partitioned sum $\sum_{i=1}^{n} h_i[c_i]$ of $h_i \in H$, does there exist a partitioned sum $\sum_{i=1}^{n} h'_i[c_i]$ of $h'_i \in H'$ that dominates $\sum_{i=1}^{n} h_i[c_i]$?**

→ Way to go! After all, no reason to ignore additivity here.

Some Simple Terminology

Start/finish: *We will be considering compilations from a "start class" H into a "finish class" H'.*

Reminder: → Chapter 15

$h[c_i]$ is h computed on Π with cost function replaced by c_i.

Closedness:

- *A class H of heuristics is closed if $h \in H$ implies $h[c_i] \in H$ for any cost function c_i.*
- We consider only closed H for the rest of this Chapter, allowing us to do partitionings *within* classes of heuristics.
- All of our families (Critical Paths, PDB, etc.) are closed: All our heuristics can be applied to any cost function.

The Compilability Framework

Definition (Compilability Between Heuristics). *Let H and H' be closed classes of admissible heuristic functions. We say that H can be compiled into H', written $H \preceq H'$, if there exists an algorithm with:*

(i) **Input:** *Task Π, state s, heuristic $h \in H$ for Π.*

(ii) **Output:** *Heuristics $h'_1, \ldots, h'_k \in H'$ for Π, and cost partitioning $c'_1, \ldots, c'_k$ for Π, such that $h(s) \leq \sum_{i=1}^{k} h'_i[c'_i](s)$.*

[$\leq h^*(s)$: partitioned sum admissible, cf. Chapter 15]

(iii) **Runtime:** *Polynomial in the size of the input.*

If we can fix $k = 1$, we say that H can be compiled into individual H', and write $\preceq^i$. If we can fix $h'_1, \ldots, h'_k$ and $c'_1, \ldots, c'_k$ for all states s in Π, we say that H can be universally compiled into H', and write $\preceq_u$.

→ H can be compiled into H' if, given a state s and $h \in H$, we can efficiently obtain an at least as good lower bound for s by a partitioned sum of $h'_i \in H'$.

→ If we need only one h', then the compilation is individual. If we can fix one combination of h' for all states of Π, then the compilation is universal.

"Heuristic h" as an Algorithm Input/Output?

"Data Structure h": We overload h to denote (a) the heuristic function itself as usual, and (b) some structure allowing to compute h in time polynomial in the size of that structure.

- For abstraction heuristics with abstraction mapping α, the "data structure h" is the abstract state space Θ^α.
- For h^m and $h^{\max}$, the "data structure h" is the task Π itself.
- For $h^{\mathsf{LM}}_{L^+}$, the "data structure h" is the action set L^+ and the task Π itself.
- For h^* and h^{LM}_{L}, "data structure h" is the original state space, for h^+ it is the state space of the delete-relaxed problem.
  → Exponentially large in $\|\Pi\|$ as these h are hard to compute.

→ Computational efficiency is important when examining how to compile heuristics into each other. Our compilation framework inputs, and outputs, representations allowing to compute h efficiently.

Questionnaire

Question!

Say $H = \{h^*\}$. Does there exist a family H' of heuristic functions into which H can be compiled?

(A): Yes (B): No

→ Yes, for example $H' := \{h^*\}$. Note that this is computable in time polynomial *in the size of the input* which contains h^* (cf. previous slide).

→ H can be compiled into H' only if either (1) the heuristics $h'_i \in H'$ cannot be computed in time polynomial in $\|\Pi\|$, or (2) the number k of such heuristics produced is not polynomial in $\|\Pi\|$. Otherwise, we could compute h^* in time polynomial in $\|\Pi\|$.

Namely, if $\{h^*\}$ were compilable into a family H' of heuristic functions, with both (1) and (2) being false, then for any task Π and state s we could efficiently find $h'_i \in H'$, and a cost partitioning c'_i, such that $h^*(s) \leq \sum_{i=1}^{k} h'_i[c'_i](s)$. By admissibility, we also have $\sum_{i=1}^{k} h'_i[c'_i](s) \leq h^*(s)$, hence $h^*(s) = \sum_{i=1}^{k} h'_i[c'_i](s)$. So we could then compute $h^*(s)$ in polynomial time, and in consequence in particular decide plan existence in polynomial time, getting **P=PSPACE** (and thus in particular **P=NP**).

The Compilability Framework: Remarks

→ H can be compiled into H' if, given a state s and $h \in H$, we can efficiently obtain an at least as good lower bound for s by a partitioned sum of $h'_i \in H'$.

→ If we need only one h', then the compilation is individual. If we can fix one combination of h' for all states of Π, then the compilation is universal.

- The per-state compilation is needed because good cost partitionings are state-dependent (cf. **Chapter 15**).
- "Efficient" means that we can find the "structure h'_i" in polynomial time. Without this restriction, the compilation would be useless (e.g., everything could be compiled into h^*).
- The special case $\preceq^i_u$ is (B) on slide 11. Examples:

  PDB $\preceq^i_u$ M&S: Given a PDB heuristic h, we can in time polynomial in the PDB's size construct an M&S heuristic h' dominating h (cf. **Chapter 13**).

  We do *not* have $h^{\mathsf{LM}}_{L^+} \preceq^i_u h^{\mathsf{LM}}_{L}$: Because "data structure $h^{\mathsf{LM}}_{L^+}$" has size polynomial in $\|\Pi\|$ while "data structure h^{LM}_{L}" has size exponential in $\|\Pi\|$. (In other words: $h^{\mathsf{LM}}_{L^+}$ but not h^{LM}_{L} can be computed in polynomial time.)

Compiling *Combinations* of $h \in H$

Theorem. *Let H and H' be closed classes of admissible heuristic functions, where $H \preceq H'$. Then there exists an algorithm with:*

- **Input:** *Task Π, state s, $h_1, \ldots, h_n \in H$, cost partitioning $c_1, \ldots, c_n$.*
- **Output:** *Heuristics $h'_1, \ldots, h'_k \in H'$ for Π, and cost partitioning $c'_1, \ldots, c'_k$, such that $\sum_{i=1}^{n} h_i[c_i](s) \leq \sum_{j=1}^{k} h'_j[c'_j](s)$.*
- **Runtime:** *Polynomial in the size of the input.*

Proof. Denote Π with cost function c_i by Π_i. Because H is closed, $h_i[c_i] \in H$, so with $H \preceq H'$ we efficiently get heuristics $h'_{i,1}, \ldots, h'_{i,k_i} \in H'$ for Π_i, and a cost partitioning $c'_{i,1}, \ldots, c'_{i,k_i}$ for Π_i, so that $h_i[c_i](s) \leq \sum_{l=1}^{k_i} h'_{i,l}[c'_{i,l}](s)$. We set $h'_1, \ldots, h'_k := h'_{1,1}, \ldots, h'_{n,k_n}$ and $c'_1, \ldots, c'_k := c'_{1,1}, \ldots, c'_{n,k_n}$.

Clearly, $\sum_{i=1}^{n} h_i[c_i](s) \leq \sum_{j=1}^{k} h'_j(s)$.

$\sum_{l=1}^{k_i} c'_{i,l}(a) \leq c_i(a)$ for all i because $c'_{i,1}, \ldots, c'_{i,k_i}$ is a cost partitioning for Π_i.
$\sum_{i=1}^{n} c_i(a) \leq c(a)$ because $c_1, \ldots, c_n$ is a cost partitioning for Π. Thus
$\sum_{j=1}^{k} c'_j(a) \leq c(a)$ so $c'_1, \ldots, c'_k$ is a cost partitioning for Π as desired.

Transitivity and Reflexivity

Corollary (Compilability is Transitive). *Let H, H', and H'' be closed classes of admissible heuristic functions, where $H \preceq H'$ and $H' \preceq H''$. Then $H \preceq H''$.*

Proof. Need: Efficient algorithm taking Π, s, and $h \in H$, and delivering $h''_1, \ldots, h''_k \in H''$ and cost partitioning $c''_1, \ldots, c''_k$ with $h(s) \leq \sum_{j=1}^{k} h''_j[c''_j](s)$.
Since $H \preceq H'$, we get $h'_1, \ldots, h'_n \in H'$ and a cost partitioning $c'_1, \ldots, c'_n \in H'$ with $h(s) \leq \sum_{i=1}^{n} h'_i[c_i](s)$. Since $H' \preceq H''$, by the theorem on the previous slide we get $h''_1, \ldots, h''_k \in H''$ and cost partitioning $c''_1, \ldots, c''_k$ with $\sum_{i=1}^{n} h'_i[c_i](s) \leq \sum_{j=1}^{k} h''_j[c''_j](s)$, which is what we needed.

Proposition (Compilability is Reflexive). *Let H be a closed class of admissible heuristic functions. Then $H \preceq H$.*

Proof. Simply set $k := 1$, $h'_1 := h$ and $c'_1 := c$.

→ We can think of "$\preceq$" like "$\leq$ for classes of heuristics".

Notation: If $H \preceq H'$ and $H' \preceq H$, we write $H \equiv H'$.

Questionnaire

Question!

Consider STRIPS planning, and $H = \{h^+\}$. Does there exist a family H' of heuristic functions into which H can be compiled?

(A): Yes (B): No

→ Yes, but same (1) and (2) as on slide 15. Otherwise, we could compute h^+ in time polynomial in $\|\Pi\|$, because on STRIPS tasks without delete lists we have $h^+ = h^*$.

Namely, if $\{h^+\}$ were compilable into a family H' of heuristic functions, with both (1) and (2) being false, then for any STRIPS task Π^+ with empty delete lists, and any state s, we could efficiently find $h'_i \in H'$, and a cost partitioning c'_i, such that $h^+(s) \leq \sum_{i=1}^k h'_i[c'_i](s) \leq h^*(s)$. Because Π^+ has empty delete lists, we have $h^+(s) = h^*(s)$. So we would get $h^+(s) = \sum_{i=1}^k h'_i[c'_i](s)$ and could compute $h^+(s)$ in polynomial time, implying that **P=NP**.

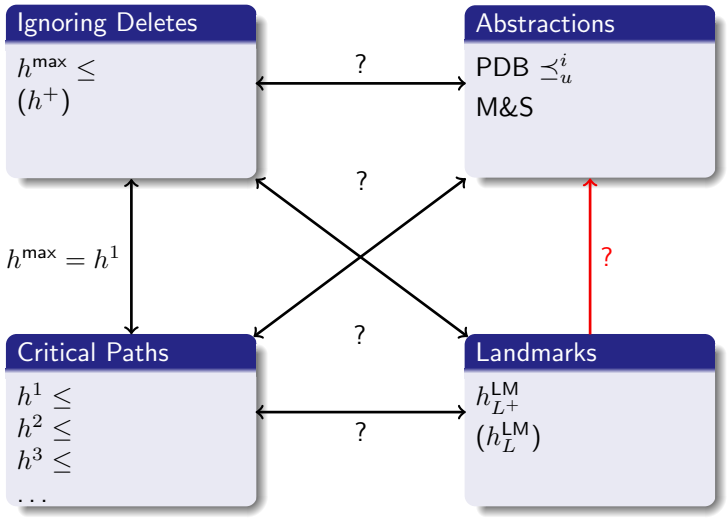
Note: We restrict to STRIPS here only to make the question easier to answer. Delete-free STRIPS planning can be simulated in FDR, so the same result holds there.

Before We Begin

- In what follows, we'll look at a few example compilability and uncompilability proofs.
- I've basically selected some simple examples for illustration, and one example because it is interesting and important in practice.
- All these examples are from [Helmert and Domshlak (2009)].

- We'll ignore from now on the heuristics that cannot be computed in time polynomial in $\|\Pi\|$: h^+ and h^{LM}_L.
- By definition, no h can be compiled into such heuristics, unless h itself already uses an exponentially large "data structure h".
- As for compiling such heuristics into other h, cf. slide 15 and slide 19.

Learning Things About "The Zoo", Part I



Example Proof: $h^{\mathsf{LM}}_{L^+}$ to PDB

Theorem ($h^{\mathsf{LM}}_{L^+} \not\preceq$PDB). *Let H be the family of elementary delete relaxation landmark heuristics $h^{\mathsf{LM}}_{L^+}$, and let H' be the family of abstraction heuristics h^α where α is a projection. Then $H \not\preceq H'$.*

Proof. Consider the unit-cost STRIPS planning task family $\Pi_n = (P_n, A_n, c, I, G)$ where $P_n = \{q_1, \ldots, q_n, g\}$, $I = \emptyset$, $G = \{g\}$, and $A_n = \{setup_i, goal_i \mid 1 \leq i \leq n\}$ where $setup_i$ adds q_i with an empty precondition at cost 1, and $goal_i$ adds g with precondition q_i at cost 0. Let $L^+ := \{setup_i \mid 1 \leq i \leq n\}$.

L^+ is a delete relaxation landmark for I, and $h^{\mathsf{LM}}_{L^+}(I) = 1$. However, for any $h' = \pi_{V'}$ where $V' \neq V$, we have $h'(I) = 0$.

For construction in time polynomial in $\|\Pi\|$ (and thus in the representation of $h^{\mathsf{LM}}_{L^+}$), the size of $\Theta^{\pi_{V'}}$ must be polynomial in n, thus $V' \neq V$ for large n. Hence $\sum_{i=1}^k h'_i(s) = 0$ for any efficiently constructible $h'_1, \ldots, h'_k \in H'$.

Example Proof: $h^{\mathsf{LM}}_{L^+}$ to M&S

Theorem ($h^{\mathsf{LM}}_{L^+} \preceq^i$ M&S). *Let H be the family of elementary delete relaxation landmark heuristics $h^{\mathsf{LM}}_{L^+}$, and let H' be the family of abstraction heuristics h^α where α is constructed by merge-and-shrink abstraction. Then $H \preceq^i H'$.*

Proof Sketch. Assume a planning task Π with cost function c, a state s, and a delete relaxation landmark L^+ for s. We construct a merge-and-shrink abstraction α so that $h^\alpha(s) \geq h^{\mathsf{LM}}_{L^+}(s)$, as follows.

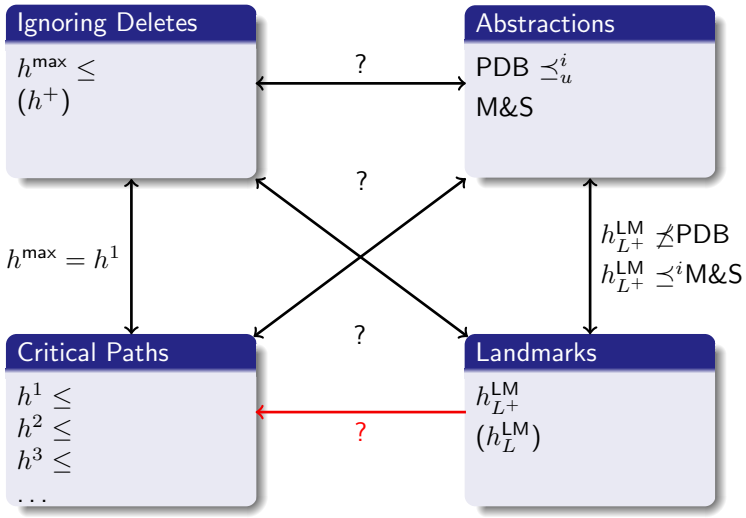
We first compute the set F^- of all facts that can be reached from s under the delete relaxation when *not* using any action from L^+.

Then we run merge-and-shrink, aggregating states so that we obtain α with an image of just two abstract states, p^- and p^+, and

$$\alpha(s) = \begin{cases} p^- & s \subseteq F^- \\ p^+ & s \not\subseteq F^- \end{cases}$$

Since L^+ is a delete relaxation LM for s, all goal states s are mapped to p^+. But I is mapped to p^-. All abstract transitions from p^- to p^+ must use an action from L^+. Hence $h^\alpha(s) \geq \min_{a \in L^+} c(a) = h^{\mathsf{LM}}_{L^+}(s)$ as desired.

Learning Things About "The Zoo", Part II



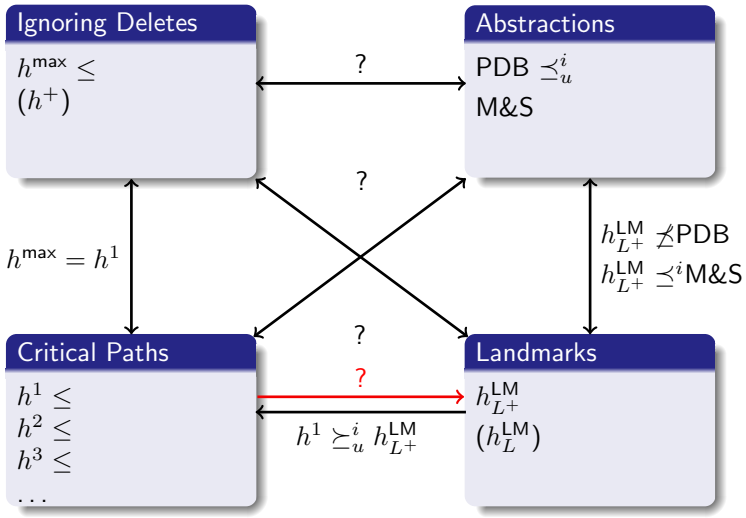
Example Proof: $h^{\mathsf{LM}}_{L^+}$ to h^1

Theorem ($h^{\mathsf{LM}}_{L^+} \preceq^i_u h^1$). *Let H be the family of elementary delete relaxation landmark heuristics $h^{\mathsf{LM}}_{L^+}$, and let $H' = \{h^1\}$ be the family that contains only h^1. Then $H \preceq^i_u H'$.*

Proof. Assume a planning task Π with action set A and cost function c, and $L^+ \subseteq A$. We show that, for all states s, $h^{\mathsf{LM}}_{L^+}(s) \leq h^1(s)$. Let s be any state. If L^+ is not a delete relaxation landmark for s, then $h^{\mathsf{LM}}_{L^+}(s) = 0$ and trivially $h^{\mathsf{LM}}_{L^+}(s) \leq h^1(s)$. Similar if $h^1(s) = \infty$. Assume that neither is the case.

Because L^+ is a delete relaxation landmark for s, there exists no relaxed plan for s using actions $A \setminus L^+$. As h^1 returns ∞ iff no relaxed plan exists (cf. **Chapter 9**), $h^1(s)$ returns ∞ when not allowing to use any action from L^+. But $h^1(s) \neq \infty$, so the critical path for $h^1(s)$ must make use of at least one of the actions in L^+. Hence $h^1(s) \geq \min_{a \in L^+} c(a) = h^{\mathsf{LM}}_{L^+}(s)$ as desired.

Learning Things About "The Zoo", Part III



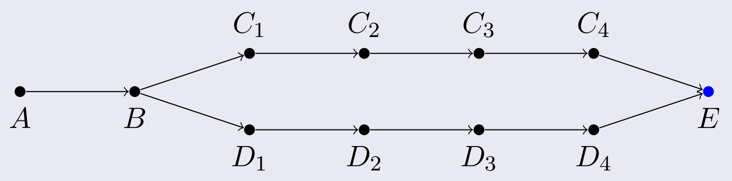
Example Proof: h^1 to $h^{\mathrm{LM}}_{L^+}$

Theorem ($h^1 \preceq h^{\mathbf{LM}}_{L^+}$). *Let $H = \{h^1\}$ be the family that contains only h^1, and let H' be the family of elementary delete relaxation landmark heuristics $h^{\mathrm{LM}}_{L^+}$. Then $H \preceq H'$.*

Proof. LM-cut!

Reminder: (Rough Intuition!) → **Chapter 14**

Get L as a *cut* between the initial state and the "0-cost goal zone"; reduce the cost of each action in L by $\min_{a \in L} c(a)$; iterate.



Introduction Compilability Example Proofs The Zoo Conclusion References

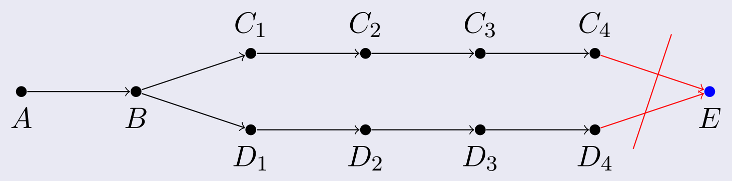
Example Proof: h^1 to $h^{\mathrm{LM}}_{L^+}$

Theorem ($h^1 \preceq h^{\mathbf{LM}}_{L^+}$). Let $H = \{h^1\}$ be the family that contains only h^1, and let H' be the family of elementary delete relaxation landmark heuristics $h^{\mathrm{LM}}_{L^+}$. Then $H \preceq H'$.

Proof. LM-cut!

Reminder: (Rough Intuition!) → Chapter 14

Get L as a cut between the initial state and the "0-cost goal zone"; reduce the cost of each action in L by $\min_{a \in L} c(a)$; iterate.



Álvaro Torralba, Cosmina Croitoru AI Planning Chapter 17: Comparing Heuristic Functions 28/41

Introduction Compilability Example Proofs The Zoo Conclusion References

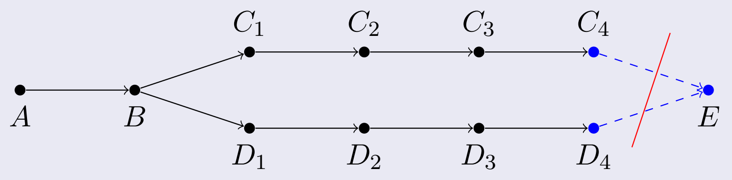
Example Proof: h^1 to $h^{\mathrm{LM}}_{L^+}$

Theorem ($h^1 \preceq h^{\mathbf{LM}}_{L^+}$). Let $H = \{h^1\}$ be the family that contains only h^1, and let H' be the family of elementary delete relaxation landmark heuristics $h^{\mathrm{LM}}_{L^+}$. Then $H \preceq H'$.

Proof. LM-cut!

Reminder: (Rough Intuition!) → Chapter 14

Get L as a cut between the initial state and the "0-cost goal zone"; reduce the cost of each action in L by $\min_{a \in L} c(a)$; iterate.



Álvaro Torralba, Cosmina Croitoru AI Planning Chapter 17: Comparing Heuristic Functions 28/41

Introduction Compilability Example Proofs The Zoo Conclusion References

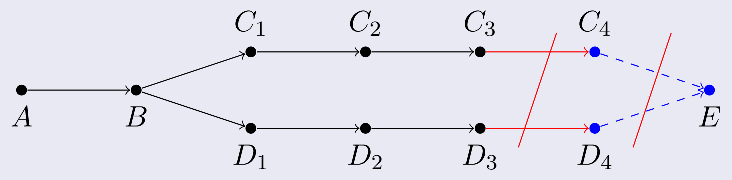
Example Proof: h^1 to $h^{\mathrm{LM}}_{L^+}$

Theorem ($h^1 \preceq h^{\mathbf{LM}}_{L^+}$). Let $H = \{h^1\}$ be the family that contains only h^1, and let H' be the family of elementary delete relaxation landmark heuristics $h^{\mathrm{LM}}_{L^+}$. Then $H \preceq H'$.

Proof. LM-cut!

Reminder: (Rough Intuition!) → Chapter 14

Get L as a cut between the initial state and the "0-cost goal zone"; reduce the cost of each action in L by $\min_{a \in L} c(a)$; iterate.



Álvaro Torralba, Cosmina Croitoru AI Planning Chapter 17: Comparing Heuristic Functions 28/41

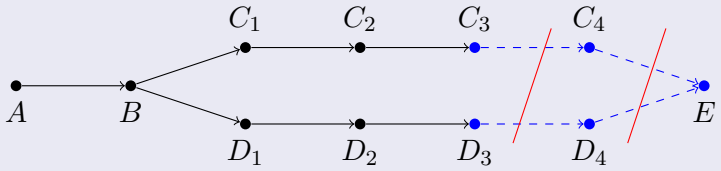
Example Proof: h^1 to $h^{\mathsf{LM}}_{L^+}$

Theorem ($h^1 \preceq h^{\mathsf{LM}}_{L^+}$). *Let $H = \{h^1\}$ be the family that contains only h^1, and let H' be the family of elementary delete relaxation landmark heuristics $h^{\mathsf{LM}}_{L^+}$. Then $H \preceq H'$.*

Proof. LM-cut!

Reminder: (Rough Intuition!) → Chapter 14

Get L as a *cut* between the initial state and the "0-cost goal zone"; reduce the cost of each action in L by $\min_{a \in L} c(a)$; iterate.



Introduction Compilability Example Proofs The Zoo Conclusion References

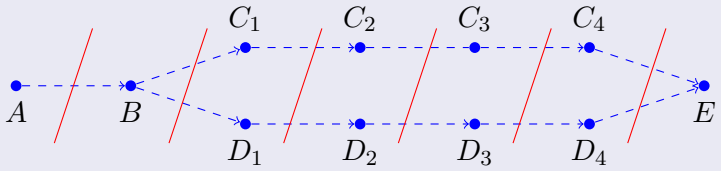
Example Proof: h^1 to $h^{\mathsf{LM}}_{L^+}$

Theorem ($h^1 \preceq h^{\mathsf{LM}}_{L^+}$). *Let $H = \{h^1\}$ be the family that contains only h^1, and let H' be the family of elementary delete relaxation landmark heuristics $h^{\mathsf{LM}}_{L^+}$. Then $H \preceq H'$.*

Proof. LM-cut!

Reminder: (Rough Intuition!) → Chapter 14

Get L as a *cut* between the initial state and the "0-cost goal zone"; reduce the cost of each action in L by $\min_{a \in L} c(a)$; iterate.



Álvaro Torralba, Cosmina Croitoru AI Planning Chapter 17: Comparing Heuristic Functions 28/41

LM-cut Algorithm in Detail

Assume: $I = \{i\}$; $G = \{g\}$; for all a we have $|pre_a| \geq 1$. (Without loss of generality: this can be achieved with simple transformations.)

Assume: $h^1(s) \neq \infty$; else, return $L_1^+ = \emptyset$, so that $h^{\mathsf{LM}}_{L_1^+} = \infty$ (cf. Chapter 14).

Input: Planning task Π, state s.
Output: Delete relaxation landmarks L_i^+ for s along with a cost partitioning c_i.

$i := 1$
loop do
  Compute $h^1[c](s)$; **if** $h^1(s) = 0$ **then** stop
  For each a, select $p_a \in pre_a$ with maximal $h^1(s, \{p\})$
  Build a graph L whose nodes are the facts, and
    with a labeled arc $p_a \xrightarrow{a} q$ whenever $q \in \mathit{eff}_a$
  $L_i^+ :=$ the labels of a cut in L between i and
    the part of L from which g can be reached with 0 cost
  $c_i(a) := \min_{a \in L_i^+} c(a)$ for the actions in L_i^+, and $c_i(a) := 0$ elsewhere
  For $a \in L_i^+$, subtract $\min_{a \in L_i^+} c(a)$ from $c(a)$
  $i := i + 1$

LM-cut Algorithm: Detailed Example

- Facts i, g, A, B, C.
- $I = \{i\}$; $G = \{g\}$.
- Actions (unit cost):
  $carA$ pre i eff A;
  $carB$ pre i eff B;
  $carC$ pre i eff C;
  $combineFilms$ pre A, B, C eff g.

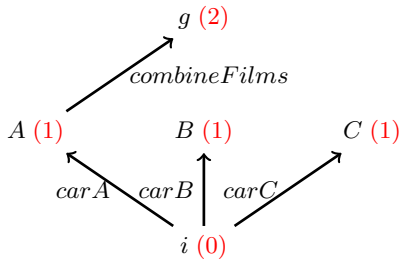
g (2)

A (1) B (1) C (1) Compute $h^1(s)$; **if** $h^1(s) = 0$ **then** stop

i (0)

LM-cut Algorithm: Detailed Example

- Facts i, g, A, B, C.
- $I = \{i\}$; $G = \{g\}$.
- Actions (unit cost):
  $carA$ pre i eff A;
  $carB$ pre i eff B;
  $carC$ pre i eff C;
  $combineFilms$ pre A, B, C eff g.

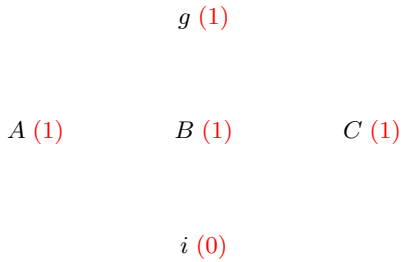


For each a, select $p_a \in pre_a$ with maximal $h^1(s, \{p\})$

Build a graph whose nodes are the facts, and with a labeled arc $p_a \xrightarrow{a} q$ whenever $q \in eff_a$

LM-cut Algorithm: Detailed Example

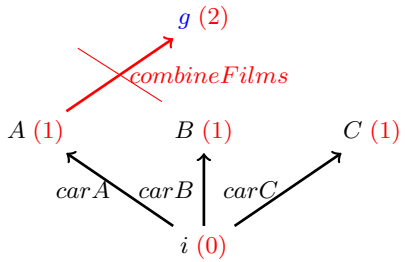
- Facts i, g, A, B, C.
- $I = \{i\}$; $G = \{g\}$.
- Actions (unit cost):
  $carA$ pre i eff A;
  $carB$ pre i eff B;
  $carC$ pre i eff C;
  $combineFilms$ pre A, B, C eff g.



Compute $h^1(s)$; **if** $h^1(s) = 0$ **then** stop

LM-cut Algorithm: Detailed Example

- Facts i, g, A, B, C.
- $I = \{i\}$; $G = \{g\}$.
- Actions (unit cost):
  $carA$ pre i eff A;
  $carB$ pre i eff B;
  $carC$ pre i eff C;
  $combineFilms$ pre A, B, C eff g.



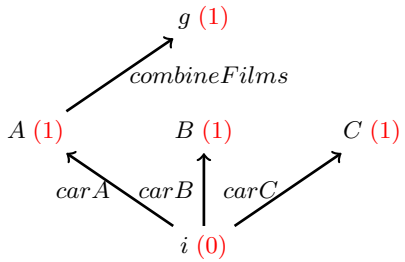
$L_1^+ := \{combineFilms\}$

$c_1(combineFilms) := 1$, and $c_1(a) := 0$ elsewhere

$c(combineFilms) := 0$

LM-cut Algorithm: Detailed Example

- Facts i, g, A, B, C.
- $I = \{i\}$; $G = \{g\}$.
- Actions (unit cost):
  $carA$ pre i eff A;
  $carB$ pre i eff B;
  $carC$ pre i eff C;
  $combineFilms$ pre A, B, C eff g.

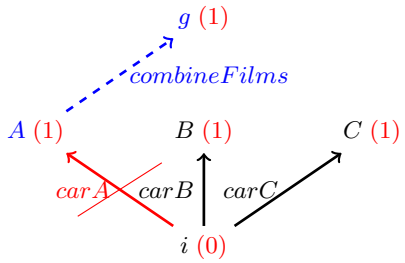


For each a, select $p_a \in pre_a$ with maximal $h^1(s, \{p\})$

Build a graph whose nodes are the facts, and with a labeled arc $p_a \xrightarrow{a} q$ whenever $q \in eff_a$

LM-cut Algorithm: Detailed Example

- Facts i, g, A, B, C.
- $I = \{i\}$; $G = \{g\}$.
- Actions (unit cost):
  $carA$ pre i eff A;
  $carB$ pre i eff B;
  $carC$ pre i eff C;
  $combineFilms$ pre A, B, C eff g.



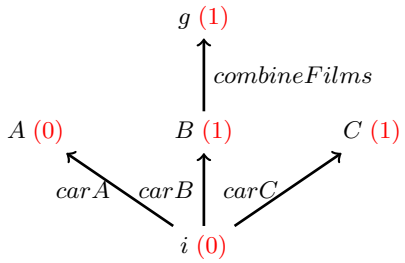
$L_2^+ := \{carA\}$

$c_2(carA) := 1$, and $c_2(a) := 0$ elsewhere

$c(carA) := 0$

LM-cut Algorithm: Detailed Example

- Facts i, g, A, B, C.
- $I = \{i\}$; $G = \{g\}$.
- Actions (unit cost):
  $carA$ pre i eff A;
  $carB$ pre i eff B;
  $carC$ pre i eff C;
  $combineFilms$ pre A, B, C eff g.

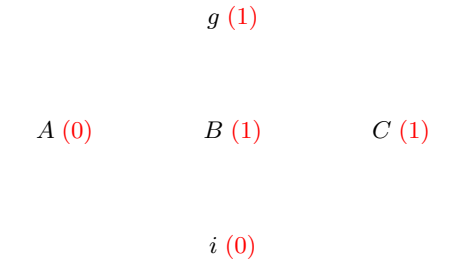


For each a, select $p_a \in pre_a$ with maximal $h^1(s, \{p\})$

Build a graph whose nodes are the facts, and with a labeled arc $p_a \xrightarrow{a} q$ whenever $q \in eff_a$

LM-cut Algorithm: Detailed Example

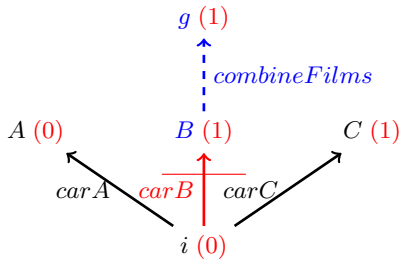
- Facts i, g, A, B, C.
- $I = \{i\}$; $G = \{g\}$.
- Actions (unit cost):
  $carA$ pre i eff A;
  $carB$ pre i eff B;
  $carC$ pre i eff C;
  $combineFilms$ pre A, B, C eff g.



Compute $h^1(s)$; **if** $h^1(s) = 0$ **then** stop

LM-cut Algorithm: Detailed Example

- Facts i, g, A, B, C.
- $I = \{i\}$; $G = \{g\}$.
- Actions (unit cost):
  $carA$ pre i eff A;
  $carB$ pre i eff B;
  $carC$ pre i eff C;
  $combineFilms$ pre A, B, C eff g.



$L_3^+ := \{carB\}$

$c_3(carB) := 1$, and $c_3(a) := 0$ elsewhere

$c(carB) := 0$

LM-cut Algorithm: Detailed Example

- Facts i, g, A, B, C.
- $I = \{i\}$; $G = \{g\}$.
- Actions (unit cost):
  $carA$ pre i eff A;
  $carB$ pre i eff B;
  $carC$ pre i eff C;
  $combineFilms$ pre A, B, C eff g.

g (1)

A (0) B (0) C (1)

i (0)

Compute $h^1(s)$; **if** $h^1(s) = 0$ **then** stop

LM-cut Algorithm: Detailed Example

- Facts i, g, A, B, C.
- $I = \{i\}$; $G = \{g\}$.
- Actions (unit cost):
  $carA$ pre i eff A;
  $carB$ pre i eff B;
  $carC$ pre i eff C;
  $combineFilms$ pre A, B, C eff g.

g (1)

$combineFilms$

A (0) B (0) C (1)

$carA$ $carB$ $carC$

i (0)

For each a, select $p_a \in pre_a$ with maximal $h^1(s, \{p\})$

Build a graph whose nodes are the facts, and with a labeled arc $p_a \xrightarrow{a} q$ whenever $q \in eff_a$

LM-cut Algorithm: Detailed Example

- Facts i, g, A, B, C.
- $I = \{i\}$; $G = \{g\}$.
- Actions (unit cost):
  $carA$ pre i eff A;
  $carB$ pre i eff B;
  $carC$ pre i eff C;
  $combineFilms$ pre A, B, C eff g.

g (1)

$combineFilms$

A (0) B (0) C (1)

$carA$ $carB$ $carC$

i (0)

$L_4^+ := \{carC\}$

$c_4(carC) := 1$, and $c_4(a) := 0$ elsewhere

$c(carC) := 0$

LM-cut Algorithm: Detailed Example

- Facts i, g, A, B, C.
- $I = \{i\}$; $G = \{g\}$.
- Actions (unit cost):
  $carA$ pre i eff A;
  $carB$ pre i eff B;
  $carC$ pre i eff C;
  $combineFilms$ pre A, B, C eff g.

g (0)

A (0) B (0) C (0)

i (0)

Compute $h^1(s)$; **if** $h^1(s) = 0$ **then** stop

LM-cut Algorithm: Detailed Example

- Facts i, g, A, B, C.
- $I = \{i\}$; $G = \{g\}$.
- Actions (unit cost):
  $carA$ pre i eff A;
  $carB$ pre i eff B;
  $carC$ pre i eff C;
  $combineFilms$ pre A, B, C eff g.

g (0)

A (0) B (0) C (0)

i (0)

Output:

- $L_1^+ = \{combineFilms\}$; $c_1(combineFilms) = 1$.
- $L_2^+ = \{carA\}$; $c_2(carA) = 1$.
- $L_3^+ = \{carB\}$; $c_3(carB) = 1$.
- $L_4^+ = \{carC\}$; $c_4(carC) = 1$.
- Heuristic $\sum_{i=1}^{4} h^{\mathsf{LM}}_{L_i^+}[c_i](I) = 4 > 2 = h^1(I)$.

LM-cut Algorithm: Why Does This Work?

→ We still need to prove the slide 26 Theorem:

Proof. Assume that $h^1(s) \neq \infty$ (else we return the empty landmark).

Say LM-cut stops after k iterations. We need to prove that (a) $\sum_{i=1}^{k} h^{\mathsf{LM}}_{L_i^+}[c_i](s) \geq h^1(s)$, and (b) LM-cut runs in time polynomial in $\|\Pi\|$.

If $h^1(s) = 0$, both (a) and (b) hold trivially. Else, consider the first iteration of the LM-cut algorithm, characterized by the sum $h(s) + h'(s)$, where $h(s) = h^{\mathsf{LM}}_{L_1^+}(s)$ with cost function c_1, and $h'(s)$ is $h^1(s)$ with cost function $c - c_1$. We prove that $h^1(s) \leq h(s) + h'(s)$. Applying this recursively to the second iteration and so forth proves (a); as the number of 0-cost actions increases strictly in each iteration, and as each iteration takes time polynomial in $\|\Pi\|$, we have (b).

As $h(s) = h^{\mathsf{LM}}_{L_1^+}(s) = \min_{a \in L_i^+} c(a)$, proving $h^1(s) \leq h(s) + h'(s)$ comes down to proving that reducing action costs c by c_1 does not decrease $h^1(s)$ by more than $\min_{a \in L_i^+} c(a)$. Observe that $h^1(s)$ corresponds exactly to cheapest paths in the graph G. Such paths enter the "0-cost goal zone" (blue on previous slide) exactly once, hence subtracting c_1 (the step into the "0-cost goal zone") reduces their cost by at most $\min_{a \in L_i^+} c(a)$ (the cost of that step). This concludes the proof.

Questionnaire

Observe: We designed LM-cut to find a partitioned-sum LM heuristic that dominates h^1. The very first thing that LM-cut does is "Compute $h^1(s)$".

Question!

Does this mean that LM-cut is completely useless?

(A): Yes (B): No

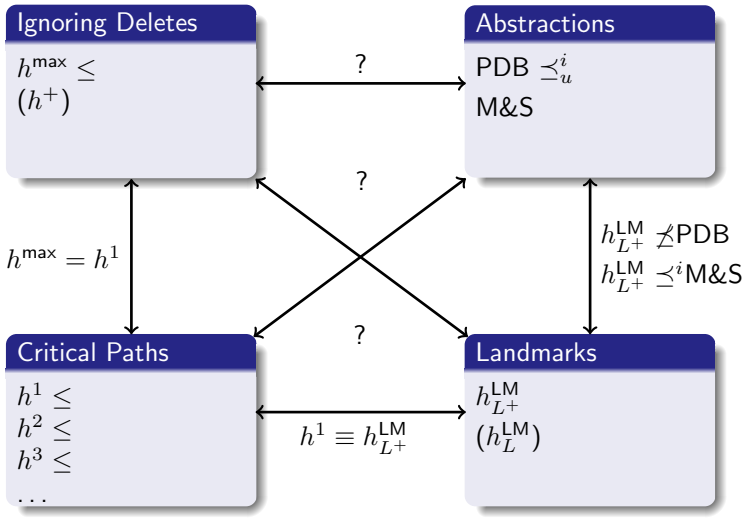
→ From a practice perspective, the answer is "NO!!!". While the proof requires only $\sum_{i=1}^{k} h^{\mathsf{LM}}_{L_i^+}[c_i](s) \geq h^1(s)$, we might have $\sum_{i=1}^{k} h^{\mathsf{LM}}_{L_i^+}[c_i](s) > h^1(s)$. Intuitively, LM-cut accounts for conjunctive requirements. E.g., the precondition $\{A, B, C\}$ on slide 30, where $\sum_{i=1}^{k} h^{\mathsf{LM}}_{L_i^+}[c_i](s) = h^*(s) = 4$ whereas $h^1(s) = 2$.

→ In practice, LM-cut almost always yields lower bounds way better than h^1!

→ From a theory perspective, compilability refers to the lower bounds we can represent using H', not how they are obtained so long as that happens in polynomial time.

→ One might consider to "disallow using h itself in the compilation of h into H'", but note – what is "h"? How would you avoid the use of equivalent/disguised heuristics?

What We Learned About "The Zoo"



Further Known Results

Corollaries of what we've seen:

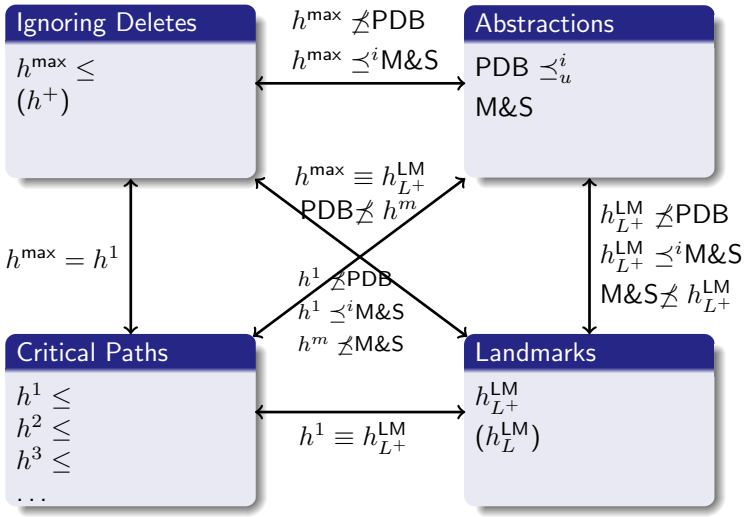
- Since $h^{\max} = h^1$ and $h^1 \equiv h^{\mathsf{LM}}_{L^+}$: $h^{\max} \equiv h^{\mathsf{LM}}_{L^+}$.
- Since $h^1 \equiv h^{\mathsf{LM}}_{L^+}$, and $h^{\mathsf{LM}}_{L^+} \not\preceq$PDB: $h^1 \not\preceq$PDB, and since $h^{\max} = h^1$: $h^{\max} \not\preceq$PDB. Further, since $h^1 \leq h^m$: $h^m \not\preceq$PDB.
- Since $h^1 \equiv h^{\mathsf{LM}}_{L^+}$, and $h^{\mathsf{LM}}_{L^+} \preceq^i$M&S: $h^1 \preceq^i$M&S, and since $h^{\max} = h^1$: $h^{\max} \preceq^i$M&S.

Additional results:

- PDB$\not\preceq h^m$ [Helmert and Domshlak (2009)].
- Since PDB$\preceq$M&S: M&S$\not\preceq h^m$, and since $h^1 \leq h^m$ and $h^1 \equiv h^{\mathsf{LM}}_{L^+}$: M&S$\not\preceq h^{\mathsf{LM}}_{L^+}$.
- $h^m \not\preceq$M&S [Helmert *et al.* (2014)].

The Full "Zoo", As We Know It

(Showing only strongest results, e.g. PDB$\not\preceq h^m$ not PDB$\not\preceq h^1$.)



Summary

- Algorithms for computing admissible heurisics can be compared by the strength of the lower bounds they can produce efficiently.
- A family H of admissible heuristics can be compiled into family H', written $H \preceq H'$, if for every state s and $h \in H$ we can efficiently construct a partitioned sum of heuristics $h' \in H'$ so that $h \leq \sum h'[c]$.
- If $H \preceq H'$, then partitioned sums over H can be compiled into partitioned sums over H' as well. Compilability is transitive and reflexive.
- Delete relaxation landmarks cannot be compiled into PDBs, but can be compiled into merge-and-shrink; they are equivalent to h^1 (and thus to $h^{\max}$ as well). All other results are negative.

Remarks

- Compilability of delete relaxation landmarks into merge-and-shrink but not PDBs is yet another indication of the superiority of merge-and-shrink.
- The most surprising result was that we can efficiently find delete relaxation landmarks compiling h^1. The implementation, LM-cut, is currently the strongest admissible heuristic available. In particular, it yields *much* better heuristic values than h^1 (i.e., it dominates h^1 in theory, and vastly outperforms it in practice).
- All other results being negative show non-domination, i.e., the respective techniques, for example critical path heuristics and merge-and-shrink, are incomparable (none dominates the other).
- (Practice is another story anyway.)

Reading

- *Landmarks, Critical Paths and Abstractions: What's the Difference Anyway?* [Helmert and Domshlak (2009)].

  Available at:

  `http://ai.cs.unibas.ch/papers/helmert-domshlak-icaps2009.pdf`

  Content: The only publication on the compilability framework, to date. Briefly introduces the framework against the (fixed) background of admissible sums via cost partitioning, and delete relaxation landmark heuristics. Provides detailed proofs for the compilations/lack thereof between landmarks and PDBs/M&S, and between landmarks and h^1. Briefly mentions non-compilability of PDBs into h^m. Runs experiments showing the excellent performance of the LM-cut heuristic, that implements the compilation of h^1 into landmarks.

References I

Malte Helmert and Carmel Domshlak. Landmarks, critical paths and abstractions: What's the difference anyway? 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 162–169. AAAI Press, 2009.

Malte Helmert, Patrik Haslum, Jörg Hoffmann, and Raz Nissim. Merge & shrink abstraction: A method for generating lower bounds in factored state spaces. *Journal of the Association for Computing Machinery*, 61(3), 2014.