AI Planning

7. Heuristic Search

How to Avoid Having to Look at a Gazillion States

Álvaro Torralba, Cosmina Croitoru

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Winter Term 2018/2019

Thanks to Prof. Jörg Hoffmann for slide sources

Agenda

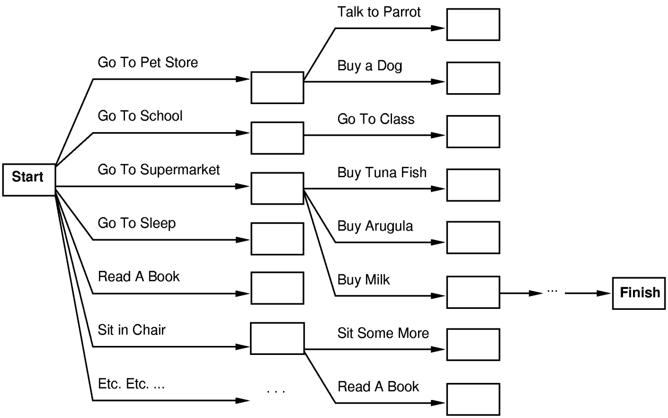
1. Introduction
2. What Are Heuristic Functions?
3. How to Use Heuristic Functions?
4. How to Obtain Heuristic Functions?
5. Conclusion

Reminder: Our Long-Term Agenda

Fill in (some) details on these choices:

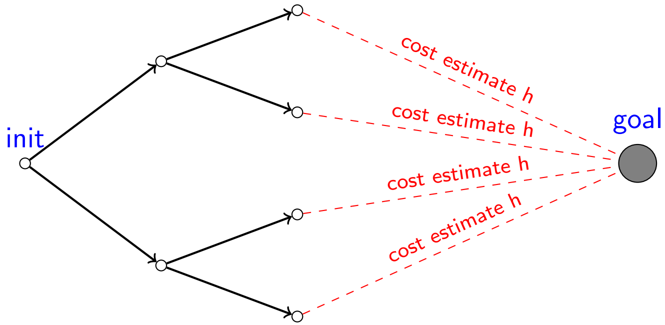
1. Search space: Progression vs. regression.
   → **Previous Chapter**
2. Search algorithm: Uninformed vs. heuristic; systematic vs. local.
   → **This Chapter**
3. Search control: Heuristic functions and pruning methods.
   → **Chapters 8–20**

Looking at a Gazillion States?



→ Use heuristic function to guide the search towards the goal!

Heuristic Search



→ Heuristic function h estimates the cost of an optimal path from a state s to the goal; search prefers to expand states s with small $h(s)$.

Live Demo vs. Breadth-First Search:

http://qiao.github.io/PathFinding.js/visual/

Our Agenda for This Chapter

2. **What Are Heuristic Functions?** Gives the basic definition, and introduces a number of important properties that we will be considering throughout the course.
3. **How to Use Heuristic Functions?** Recaps the basic heuristic search algorithms from AI'17, and adds a few new ones. Gives a few planning-specific algorithms and explanations.
4. **How to Obtain Heuristic Functions?** Recaps the concept of "Relaxation" from AI'17: A basic explanation how heuristic functions are derived in practice.

Heuristic Functions

Definition (Heuristic Function). *Let Π be a planning task with state space $\Theta_\Pi = (S, L, c, T, I, S^G)$. A heuristic function, short heuristic, for Π is a function $h : S \mapsto \mathbb{R}_0^+ \cup \{\infty\}$. Its value $h(s)$ for a state s is referred to as the state's heuristic value, or h value.*

Definition (Remaining Cost, h^*). *Let Π be a planning task with state space $\Theta_\Pi = (S, L, c, T, I, S^G)$. For a state $s \in S$, the state's remaining cost is the cost of an optimal plan for s, or ∞ if there exists no plan for s. The perfect heuristic for Π, written h^*, assigns every $s \in S$ its remaining cost as the heuristic value.*

→ Heuristic functions h estimate remaining cost h^*.

→ These definitions apply to both, STRIPS and FDR.

Heuristic Functions: The Eternal Trade-Off

What does it mean, "estimate remaining cost"?

- In principle, the "estimate" is an arbitrary function. In practice, we want it to be accurate (aka: informative), i.e., close to the actual remaining cost.
- We also want it to be fast, i.e., a small overhead for computing h.
- These two wishes are in contradiction! Extreme cases?

→ We need to trade off the accuracy of h against the overhead of computing it. → **Chapters 8–17**

→ What exactly is "accuracy"? How does it affect search performance? Interesting and challenging subject! We'll consider this in **Chapter 17**.

Questionnaire

Question!

For root-finding on a map, the straight-line distance heuristic certainly has small overhead. But is it accurate?

(A): No
(B): Yes
(C): Sometimes
(D): Maybe

Properties of Individual Heuristic Functions

Definition (Safe/Goal-Aware/Admissible/Consistent). *Let Π be a planning task with state space $\Theta_\Pi = (S, L, c, T, I, S^G)$, and let h be a heuristic for Π. The heuristic is called:*

- *safe if, for all $s \in S$, $h(s) = \infty$ implies $h^*(s) = \infty$;*
- *goal-aware if $h(s) = 0$ for all goal states $s \in S^G$;*
- *admissible if $h(s) \leq h^*(s)$ for all $s \in S$;*
- *consistent if $h(s) \leq h(s') + c(a)$ for all transitions $s \xrightarrow{a} s'$.*

$\rightarrow$ Relationships:

Proposition. *Let Π be a planning task, and let h be a heuristic for Π. If h is admissible, then h is goal-aware. If h is admissible, then h is safe. If h is consistent and goal-aware, then h is admissible. No other implications of this form hold.*

Consistency: Illustration

Properties of Individual Heuristic Functions, ctd.

Examples:

- Is h =Manhattan distance in the 15-Puzzle safe/goal-aware/admissible/ consistent? All yes. Easy for goal-aware and safe (h is never ∞). Consistency: Moving a tile can't decrease h by more than 1.
- Is h =straight line distance safe/goal-aware/admissible/consistent?
- An admissible but inconsistent heuristic: To-Moscow with $h(SB) = 1000$, $h(KL) = 100$.

→ In practice, most heuristics are safe and goal-aware, and admissible heuristics are typically consistent.

What about inadmissible heuristics?

- Inadmissible heuristics typically arise as approximations of admissible heuristics that are too costly to compute. (Examples: **Chapter 9**)

Domination Between Heuristic Functions

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 $h(s) \leq h'(s)$.*

$\rightarrow$ h' dominates h = "h' provides a lower bound at least as good as h".

Remarks:

- Example: h' =Manhattan Distance vs. h =Misplaced Tiles in 15-Puzzle: Each misplaced tile accounts for at least 1 (typically, more) in h'.
- h^* dominates every other admissible heuristic.
- Modulo tie-breaking, the search space of A^* under h' can only be smaller than that under h. (See [Holte (2010)] for details)
- In **Chapter 17**, we will consider much more powerful concepts, comparing entire *families* of heuristic functions.

Additivity of Heuristic Functions

Definition (Additivity). *Let Π be a planning task, and let $h_1, \dots, h_n$ be admissible heuristics for Π. We say that $h_1, \dots, h_n$ are additive if $h_1 + \dots + h_n$ is admissible, i.e., for all states s in Π we have $h_1(s) + \dots + h_n(s) \leq h^*(s)$.*

→ An ensemble of heuristics is additive if its sum is admissible.

Remarks:

- Example: h_1 considers only tiles 1 . . . 7, and h_2 considers only tiles 8 . . . 15, in the 15-Puzzle: The two estimates are then, intuitively, "independent".
  (h_1 and h_2 are orthogonal projections → **Chapter 12**)
- We can always combine $h_1, \dots, h_n$ admissibly by taking the $\max$. Taking $\sum$ is *much* stronger; in particular, $\sum$
- In **Chapters 15–16**, we will devise a third, strictly more general, technique to admissibly combine heuristic functions.

What Works Where in Planning?

Blind (no h) **vs. heuristic:**

- For satisficing planning, heuristic search vastly outperforms blind algorithms pretty much everywhwere.
- For optimal planning, heuristic search also is better (but the difference is not as huge).

Systematic (maintain all options) **vs. local** (maintain only a few) **:**

- For satisficing planning, there are successful instances of each.
- For optimal planning, systematic algorithms are required.

$\rightarrow$ Here, we briefly cover the search algorithms most successful in planning. For more details (in particular, for blind search), refer to AI'18 Chapters 4 and 5.

Reminder: Greedy Best-First Search and A^*

For simplicity, duplicate elimination omitted and using AI'17 notation:

function Greedy Best-First Search $[A^*]$*(problem)* **returns** a solution, or failure
  node $\leftarrow$ a node n with n.*state*=*problem.InitialState*
  frontier $\leftarrow$ a priority queue ordered by ascending h $[g+h]$, only element n
  loop do
    if *Empty?(frontier)* **then return** failure
    $n \leftarrow$ *Pop(frontier)*
    if *problem.GoalTest*(n.*State*) **then return** *Solution*(n)
    for each *action* a **in** *problem.Actions*(n.*State*) **do**
      $n' \leftarrow$ *ChildNode(problem,*n,a*)*
      Insert(n', $h(n')$ $[g(n')+h(n')]$, *frontier*)

$\rightarrow$ Greedy best-first search explores states by increasing heuristic value h.
A^* explores states by increasing plan-cost estimate $g+h$.

Greedy Best-First Search: Remarks

Properties:

- Complete? Yes, with duplicate elimination. (If $h(s) = \infty$ states are pruned, h needs to be safe.)
- Optimal? No. (Even for perfect heuristics! E.g., say the start state has two transitions to goal states, one of which costs a million bucks while the other one is for free. Nothing keeps Greedy Best-First Search from choosing the bad one.)

Technicalities:

- Duplicate elimination: Insert child node n' only if n'.State is not already contained in *explored* $\cup$ States(*frontier*). (Cf. AI'17)

Bottom line: Fast but not optimal $\implies$ satisficing planning.

A^*: Remarks

Properties:

- Complete? Yes. (Even without duplicate detection; if $h(s) = \infty$ states are pruned, h needs to be safe.)
- Optimal? Yes, for admissible heuristics.

Technicalities:

- "Plan-cost estimate" $g(s) + h(s)$ known as f-value $f(s)$ of s.
  $\rightarrow$ If $g(s)$ is taken from a cheapest path to s, then $f(s)$ is a lower bound on the cost of a plan through s.
- Duplicate elimination: If n'.State$\notin$*explored* $\cup$ States(*frontier*), then insert n'; else, insert n' only if the new path is cheaper than the old one, and if so remove the old path. (Cf. AI'17)

Bottom line: Optimal for admissible h $\Longrightarrow$ optimal planning, **with such** h.

Weighted A^*

For simplicity, duplicate elimination omitted and using AI'17 notation:

function Weighted A^**(problem)* **returns** a solution, or failure
  node ← a node n with n*.state=problem.InitialState*
  frontier ← a priority queue ordered by ascending $g + W * h$, only element n
  loop do
    if *Empty?(frontier)* **then return** failure
    $n \leftarrow$ *Pop(frontier)*
    if *problem.GoalTest*(n*.State*) **then return** *Solution*(n)
    for each *action* a **in** *problem.Actions*(n*.State*) **do**
      $n' \leftarrow$ *ChildNode(problem,*n,a*)*
      Insert(n', [$g(n') + W * h(n')$], *frontier)*

→ Weighted A^* explores states by increasing weighted-plan-cost estimate $g + W * h$.

Weighted A^*: Remarks

The weight $W \in \mathbb{R}_0^+$ is an **algorithm parameter:**

- For $W = 0$, weighted A^* behaves like?
- For $W = 1$, weighted A^* behaves like? A^*.
- For $W = 10^{100}$, weighted A^* behaves like?

Properties:

- For $W > 1$, weighted A^* is bounded suboptimal.

  $\rightarrow$ If h is admissible, then the solutions returned are at most a factor W more costly than the optimal ones.

Bottom line: Allows to interpolate between greedy best-first search and A^*, trading off plan quality against computational effort.

Hill-Climbing

function Hill-Climbing **returns** a solution
  node ← a node n with n.*state*=*problem.InitialState*
  loop do
    if *problem.GoalTest*(n.*State*) **then return** *Solution*(n)
    N ← the set of all child nodes of n
    n ← an element of N minimizing h /* (random tie breaking) */

Remarks:

- Is this complete or optimal? No.
- Can easily get stuck in local minima where immediate improvements of $h(n)$ are not possible.
- Many variations: tie-breaking strategies, restarts, ... (cf. AI'17)

Enforced Hill-Climbing [Hoffmann and Nebel (2001)]

function Enforced Hill-Climbing **returns** a solution
  node ← a node n with n.*state*=*problem.InitialState*
  loop do
    if *problem.GoalTest*(n.*State*) **then return** *Solution*(n)
    Perform breadth-first search for a node n' s.t. $h(n') < h(n)$
    $n \leftarrow n'$

Remarks:

- Is this optimal? No.
- Is this complete? See next slide.

Questionnaire

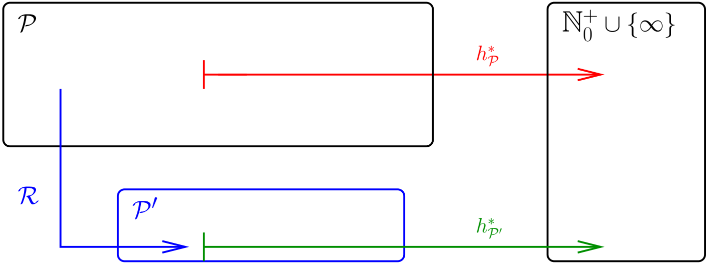
function Enforced Hill-Climbing **returns** a solution
  node $\leftarrow$ a node n with n.*state*=*problem.InitialState*
  loop do
    if *problem.GoalTest*(n.*State*) **then return** *Solution*(n)
    Perform breadth-first search for a node n' s.t. $h(n') < h(n)$
    $n \leftarrow n'$

Question!

Assume that $h(s) = 0$ if and only if s is a goal state. Is Enforced Hill-Climbing complete?

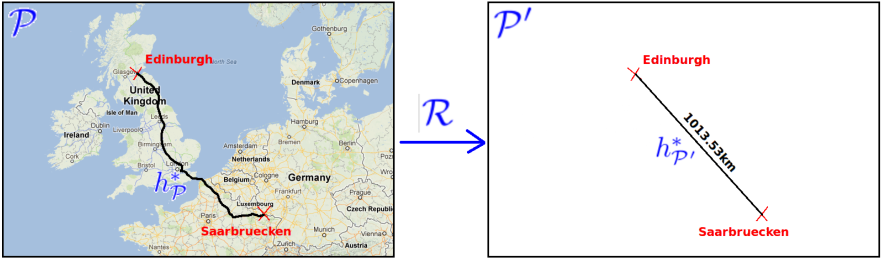
Heuristic Functions from Relaxed Problems

How to Relax



- You have a class $\mathcal{P}$ of problems, whose perfect heuristic $h^*_{\mathcal{P}}$ you wish to estimate.
- You define a class $\mathcal{P}'$ of *simpler problems*, whose perfect heuristic $h^*_{\mathcal{P}'}$ can be used to *estimate* $h^*_{\mathcal{P}}$.
- You define a transformation – the relaxation mapping $\mathcal{R}$ – that maps instances $\Pi \in \mathcal{P}$ into instances $\Pi' \in \mathcal{P}'$.
- Given $\Pi \in \mathcal{P}$, you let $\Pi' := \mathcal{R}(\Pi)$, and estimate $h^*_{\mathcal{P}}(\Pi)$ by $h^*_{\mathcal{P}'}(\Pi')$.

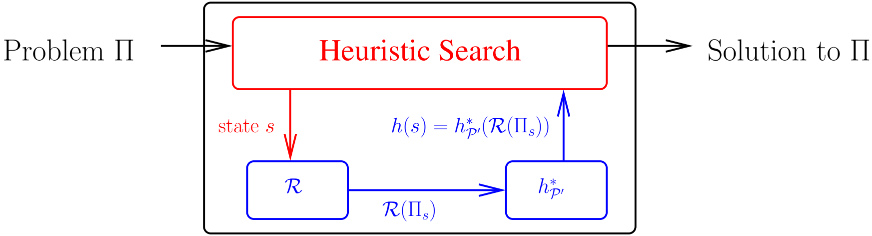
Relaxation in Route-Finding



- Problem class $\mathcal{P}$: Route finding.
- Perfect heuristic $h^*_{\mathcal{P}}$ for $\mathcal{P}$: Length of a shortest route.
- Simpler problem class $\mathcal{P}'$:
- Perfect heuristic $h^*_{\mathcal{P}'}$ for $\mathcal{P}'$:
- Transformation $\mathcal{R}$:

How to Relax During Search: Overview

Attention! Search uses the real (un-relaxed) Π. The relaxation is applied **only within the call to** $h(s)$**!!!**

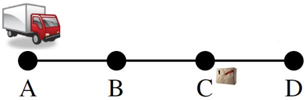


- Here, Π_s is Π with initial state replaced by s, i.e., $\Pi = (P, A, I, G)$ changed to (P, A, s, G): The task of finding a plan for search state s.
- A common student mistake is to instead apply the relaxation once to the whole problem, then doing the whole search "within the relaxation".
- Slides 34 and 32 illustrate the correct search process in detail.

How to Relax During Search: Ignoring Deletes

A Simple Planning Relaxation: Only-Adds

Example: "Logistics"



- Facts P: $\{truck(x) \mid x \in \{A, B, C, D\}\} \cup$ $pack(x) \mid x \in \{A, B, C, D, T\}\}$.
- Initial state I: $\{truck(A)$, $pack(C)\}$.
- Goal G: $\{truck(A)$, $pack(D)\}$.
- Actions A: (Notated as "precondition $\Rightarrow$ adds, $\neg$ deletes")
  - $drive(x, y)$, where x, y have a road: "$truck(x) \Rightarrow truck(y), \neg truck(x)$".
  - $load(x)$: "$truck(x), pack(x) \Rightarrow pack(T), \neg pack(x)$".
  - $unload(x)$: "$truck(x), pack(T) \Rightarrow pack(x), \neg pack(T)$".

Only-Adds Relaxation: Drop the preconditions and deletes.

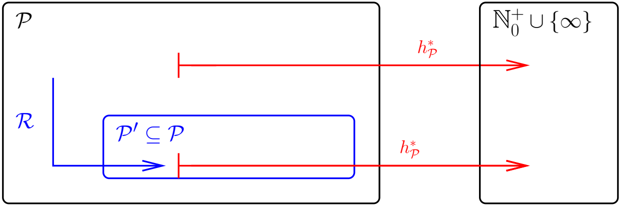
"$drive(x, y)$: $\Rightarrow truck(y)$"; "$load(x)$: $\Rightarrow pack(T)$"; "$unload(x)$: $\Rightarrow pack(x)$".

$\rightarrow$ Heuristic value for I is?

How to Relax During Search: Only-Adds

Only-Adds and Ignoring Deletes are "Native" Relaxations

Native Relaxations: Confusing special case where $\mathcal{P}' \subseteq \mathcal{P}$.



- Problem class $\mathcal{P}$: STRIPS planning tasks.
- Perfect heuristic $h^*_{\mathcal{P}}$ for $\mathcal{P}$: Length h^* of a shortest plan.
- Transformation $\mathcal{R}$: Drop the (preconditions and) delete lists.
- Simpler problem class $\mathcal{P}'$ is a special case of $\mathcal{P}$, $\mathcal{P}' \subseteq P$: STRIPS planning tasks with empty (preconditions and) delete lists.
- Perfect heuristic for $\mathcal{P}'$: Shortest plan for only-adds respectively delete-free STRIPS task.

Questionnaire

Question!

Is Only-Adds a "good heuristic" (accurate goal distance estimates) in ...

(A): Path Planning?

(B): Blocksworld?

(C): Freecell?

(D): SAT? (#unsatisfied clauses)

Summary

- Heuristic functions h map states to estimates of remaining cost. A heuristic can be safe, goal-aware, admissible, and/or consistent. A heuristic may dominate another heuristic, and an ensemble of heuristics may be additive.
- Greedy best-first search can be used for satisficing planning, A^* can be used for optimal planning provided h is admissible. Weighted A^* interpolates between the two.
- Relaxation is a method to compute heuristic functions. Given a problem $\mathcal{P}$ we want to solve, we define a relaxed problem $\mathcal{P}'$. We derive the heuristic by mapping into $\mathcal{P}'$ and taking the solution to this simpler problem as the heuristic estimate.
- During search, the relaxation is used *only inside* the computation of $h(s)$ on each state s; the relaxation does not affect anything else.

Reading

- AI'18 Chapters 4 and 5.
- A word of caution regarding *Artificial Intelligence: A Modern Approach (Third Edition)* [Russell and Norvig (2010)], Sections 3.6.2 and 3.6.3.

  Content: These little sections are aimed at describing basically what I call "How to Relax" here. They do serve to get some intuitions. However, strictly speaking, they're a bit misleading. Formally, a pattern database (Section 3.6.3) *is* what is called a "relaxation" in Section 3.6.2: as we shall see in → **Chapters 11, 12**, pattern databases are abstract transition systems that have more transitions than the original state space. On the other hand, not every relaxation can be usefully described this way; e.g., critical-path heuristics (→ **Chapter 8**) and ignoring-deletes heuristics (→ **Chapter 9**) are associated with very different state spaces.

References I

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

Robert C. Holte. Common misconceptions concerning heuristic search. In Ariel Felner and Nathan R. Sturtevant, editors, *Proceedings of the 3rd Annual Symposium on Combinatorial Search (SOCS'10)*, pages 46–51, Stone Mountain, Atlanta, GA, July 2010. AAAI Press.

Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (Third Edition)*. Prentice-Hall, Englewood Cliffs, NJ, 2010.