Artificial Intelligence

4. Classical Search, Part II: Informed Search

How to Not Play Stupid When Solving a Problem

Jana Koehler Álvaro Torralba

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Summer Term 2019

Thanks to Prof. Hoffmann for slide sources

Agenda

1. Introduction
2. Heuristic Functions
3. Systematic Search: Algorithms
4. Systematic Search: Performance
5. Local Search
6. Conclusion

(Not) Playing Stupid

→ Problem: Find a route to Moscow.

- "Look at all locations 10km distant from SB, look at all locations 20km distant from SB, ..." = **Breadth-first search.**
- "Just keep choosing arbitrary roads, following through until you hit an ocean, then back up ..." = **Depth-first search.**
- "Focus on roads that go the right direction." = **Informed search!**

Informed Search: Basic Idea

Recall: Search strategy=how to choose the next node to expand?

- Blind Search: Rigid procedure using the same expansion order no matter which problem it is applied to.

  $\rightarrow$ Blind search has 0 knowledge of the problem it is solving.

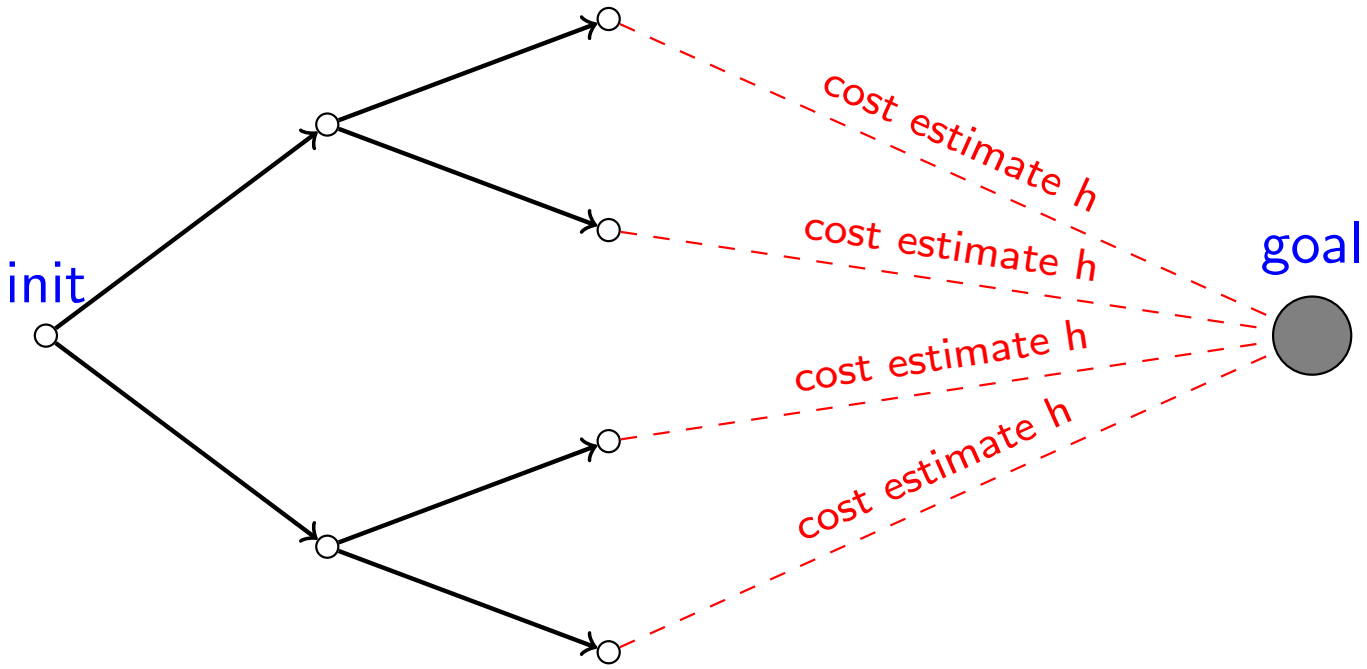
  $\rightarrow$ It can't "focus on roads that go the right direction", because it has no idea what "the right direction" is.

- Informed Search: Knowledge of the "goodness" of expanding a state s is given in the form of a heuristic function $h(s)$, which estimates the cost of an optimal (cheapest) path from s to the goal.

  $\rightarrow$ "$h(s)$ larger than where I came from $\implies$ seems s is not the right direction."

$\rightarrow$ Informed search is a way of giving the computer knowledge about the problem it is solving, thereby stopping it from doing stupid things.

Informed Search: Basic Idea, ctd.



→ 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 $h(s)$.

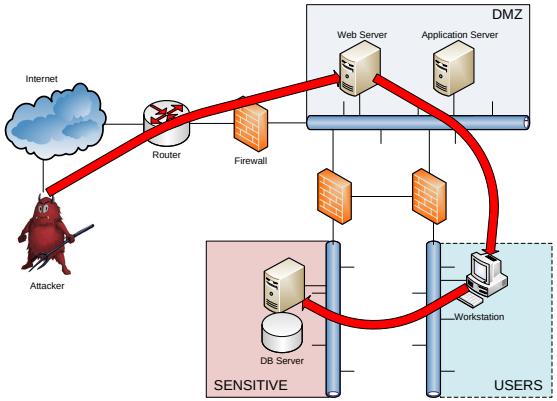
Some Applications

GPS

Robotics

Video Games

Network Security



Reminder: Our Agenda for This Topic

→ Our treatment of the topic "Classical Search" consists of Chapters 4 and 5.

- **Chapter 4:** Basic definitions and concepts; blind search.

  → Sets up the framework. Blind search is ideal to get our feet wet. It is not wide-spread in practice, but it is among the state of the art in certain applications (e.g., software model checking).

- **This Chapter:** Heuristic functions and informed search.

  → Classical search algorithms exploiting the problem-specific knowledge encoded in a heuristic function. Typically much more efficient in practice.

Our Agenda for This Chapter

- **Heuristic Functions:** How are heuristic functions h defined? What are relevant properties of such functions? How can we obtain them in practice?

  $\rightarrow$ Which "problem knowledge" do we wish to give the computer?

- **Systematic Search: Algorithms:** How to use a heuristic function h while still guaranteeing completeness/optimality of the search.

  $\rightarrow$ How to exploit the knowledge in a systematic way?

- **Systematic Search: Performance:** Empirical and theoretical observations.

  $\rightarrow$ What can we say about the performance of heuristic search? Is it actually better than blind search?

- **Local Search:** Overview of methods foresaking completeness/optimality, taking decisions based only on the local surroundings.

  $\rightarrow$ How to exploit the knowledge in a greedy way?

Heuristic Functions

Definition (Heuristic Function, h^*). *Let Π be a problem with states S. A heuristic function, short heuristic, for Π is a function $h : S \mapsto \mathbb{R}_0^+ \cup \{\infty\}$ so that, for every goal state s, we have $h(s) = 0$.*

The perfect heuristic h^* is the function assigning every $s \in S$ the cost of a cheapest path from s to a goal state, or ∞ if no such path exists.

Notes:

- We also refer to $h^*(s)$ as the goal distance of s.
- $h(s) = 0$ on goal states: If your estimator returns "I think it's still a long way" on a goal state, then its "intelligence" is, um ...
- Return value ∞: To indicate dead ends, from which the goal can't be reached anymore.
- The value of h depends only on the *state* s, not on the *search node* (i.e., the path we took to reach s). I'll sometimes abuse notation writing "$h(n)$" instead of "$h(n.\text{State})$".

Why "Heuristic"?

What's the meaning of "heuristic"?

- Heuristik: Ancient Greek $\varepsilon\upsilon\rho\iota\sigma\kappa\varepsilon\iota\nu$ (= "I find"); aka: $\varepsilon\upsilon\rho\eta\kappa\alpha$!
- Popularized in modern science by George Polya: "How to Solve It" (published 1945).
- Same word often used for: "rule of thumb", "imprecise solution method".
- In classical search (and many other problems studied in AI), it's the mathematical term just explained.

Heuristic Functions: The Eternal Trade-Off

Distance "estimate"? (h is an arbitrary function in principle!)

- We want h to be accurate (aka: informative), i.e., "close to" the actual goal distance.
- We also want it to be fast, i.e., a small overhead for computing h.
- These two wishes are in contradiction!

  $\rightarrow$ Extreme cases? $h = 0$: no overhead at all, completely un-informative. $h = h^*$: perfectly accurate, overhead=solving the problem in the first place.

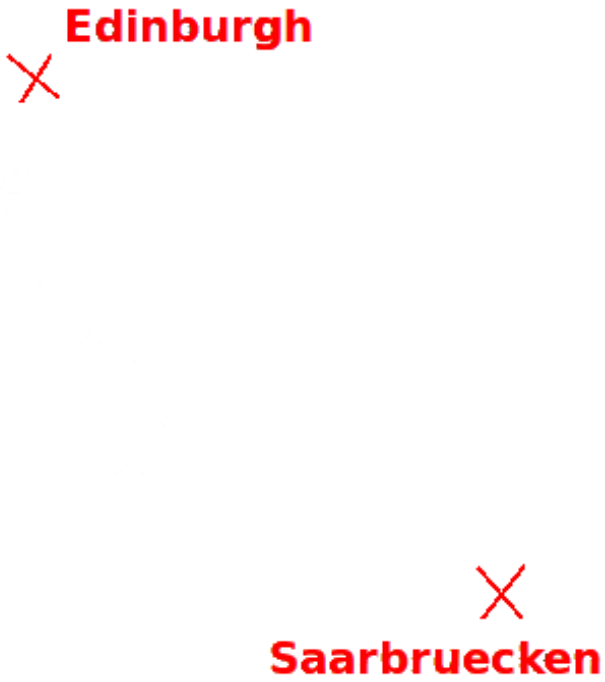
$\rightarrow$ We need to trade off the accuracy of h against the overhead for computing $h(s)$ on every search state s.

So, how to? $\rightarrow$ Given a problem Π, a heuristic function h for Π can be obtained as goal distance within a simplified (relaxed) problem Π'.

Heuristic Functions from Relaxed Problems: Example 1

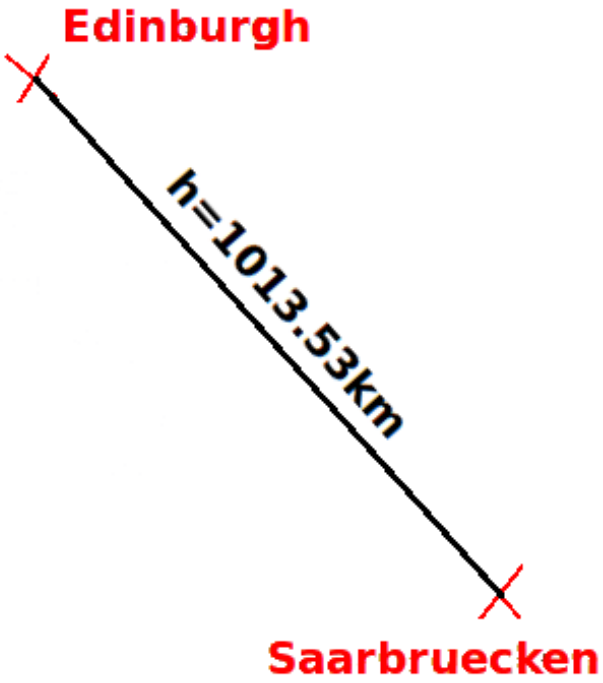
Problem II: Find a route from Saarbruecken To Edinburgh.

Heuristic Functions from Relaxed Problems: Example 1



Relaxed Problem Π': Throw away the map.

Heuristic Functions from Relaxed Problems: Example 1



Heuristic function h: Straight line distance.

Heuristic Functions from Relaxed Problems: Example 2

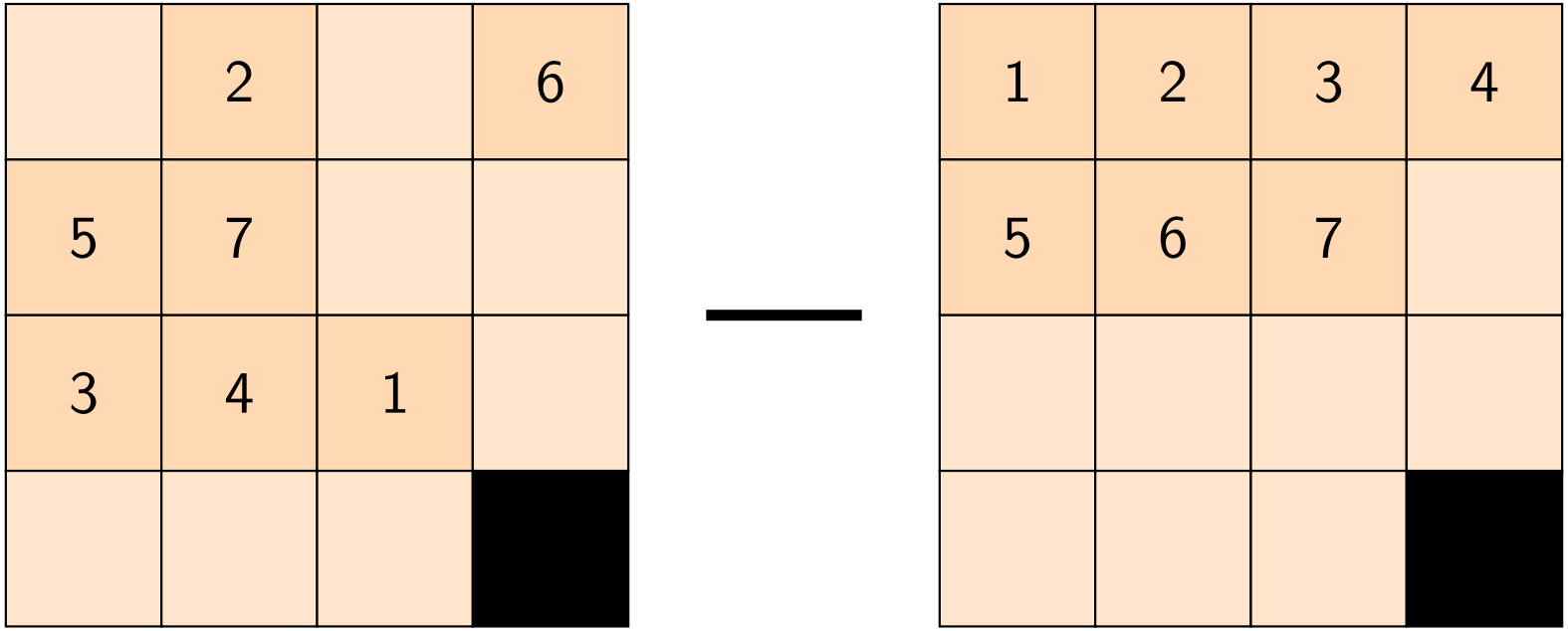
9	2	12	6
5	7	14	13
3	4	1	11
15	10	8	

—

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

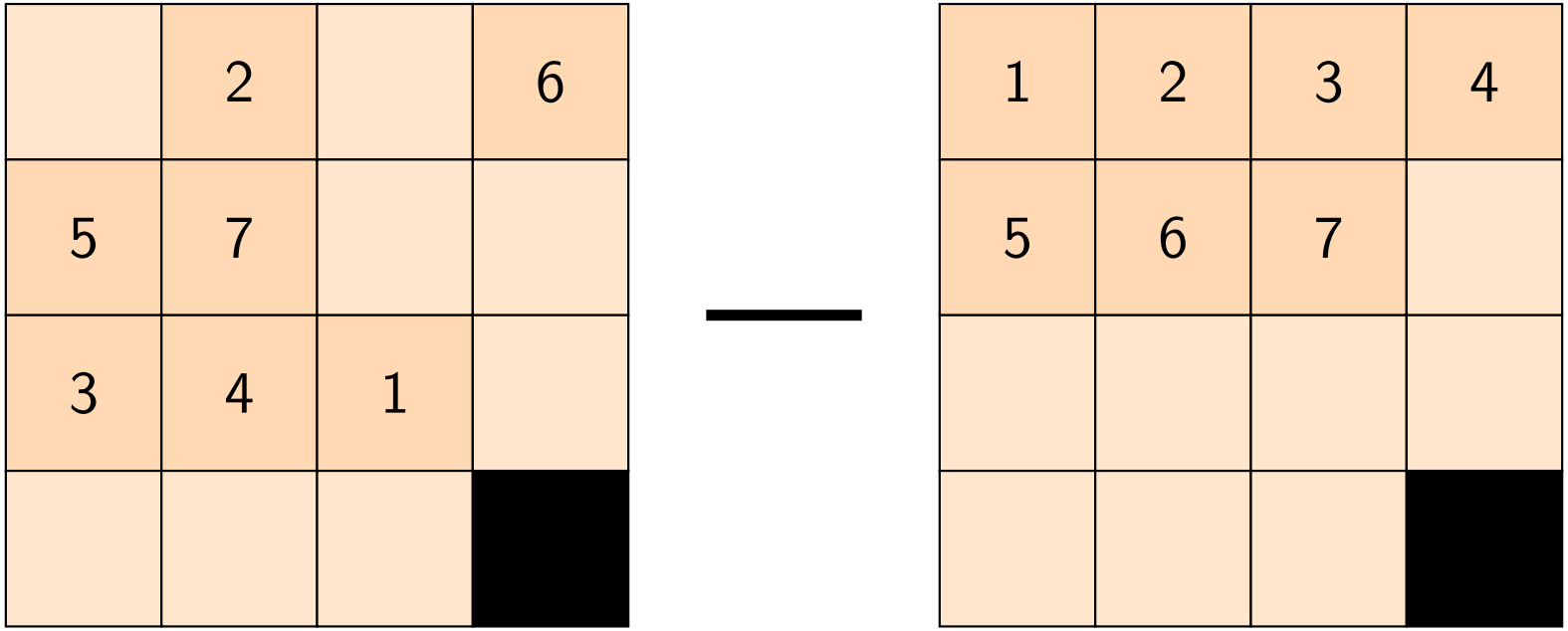
Problem Π: Move tiles to transform left state into right state.

Heuristic Functions from Relaxed Problems: Example 2



Relaxed Problem Π': Don't distinguish tiles 8–15.

Heuristic Functions from Relaxed Problems: Example 2



Heuristic function h: Length of solution to reduced puzzle.

Heuristic Functions from Relaxed Problems: Example 3

9	2	12	6
5	7	14	13
3	4	1	11
15	10	8	

—

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

- Problem Π: Move tiles to transform left state into right state.
- Relaxed Problem Π': Allow to move each tile to any neighbor cell, regardless of the situation.
- Heuristic function h: Manhattan distance. Here: 36.

Heuristic Functions from Relaxed Problems: Example 4

9	2	12	6
5	7	14	13
3	4	1	11
15	10	8	

—

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

- Problem Π: Move tiles to transform left state into right state.
- Relaxed Problem Π': Allow to move each tile to *any cell* in a single move, regardless of the situation.
- Heuristic function h: Number of misplaced tiles. Here: 13.

Heuristic Function Pitfalls: Example Path Planning

h^*:

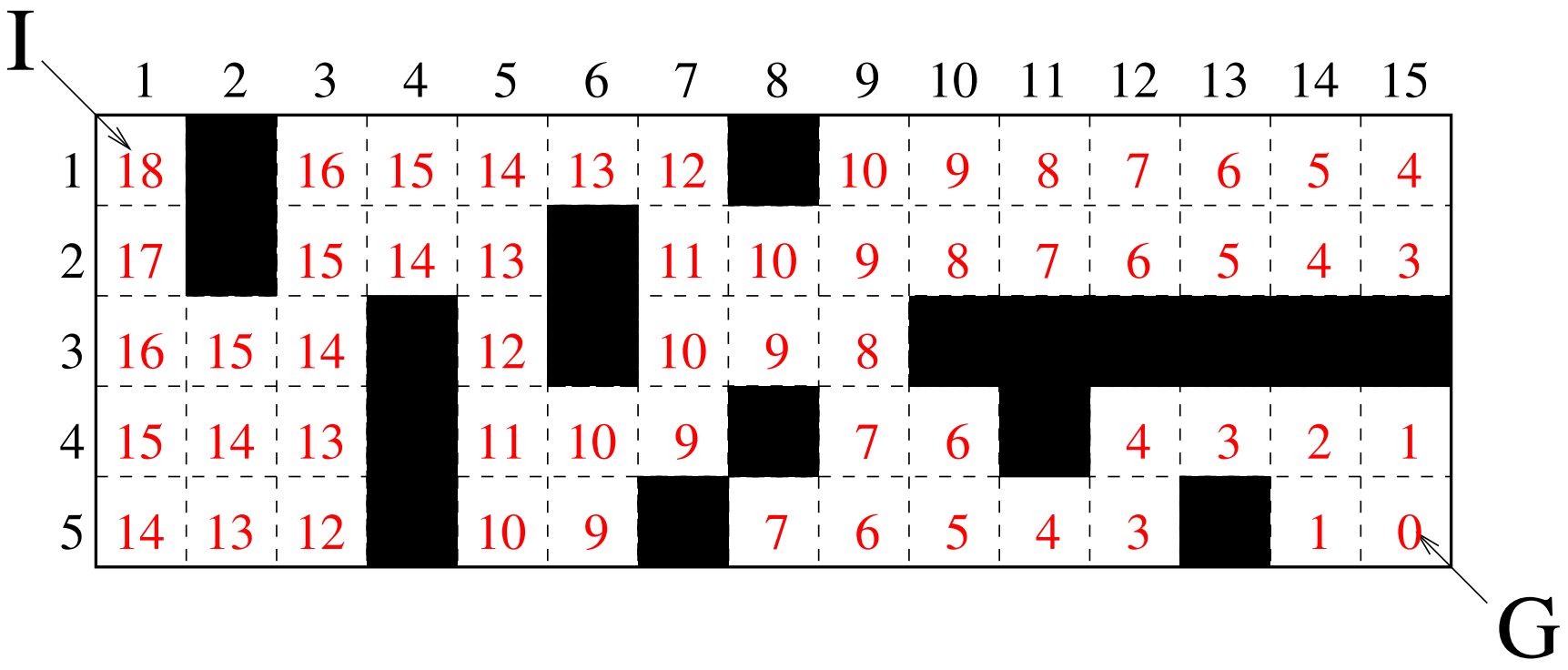
I

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	24		18	17	16	15	14		12	13	14	15	16	17	18
2	23		19	18	17		13	12	11	12	13	14	15	16	17
3	22	21	20		16		12	11	10						
4	23	22	21		15	14	13		9	8		4	3	2	1
5	24	23	22		16	15		9	8	7	6	5		1	0

G

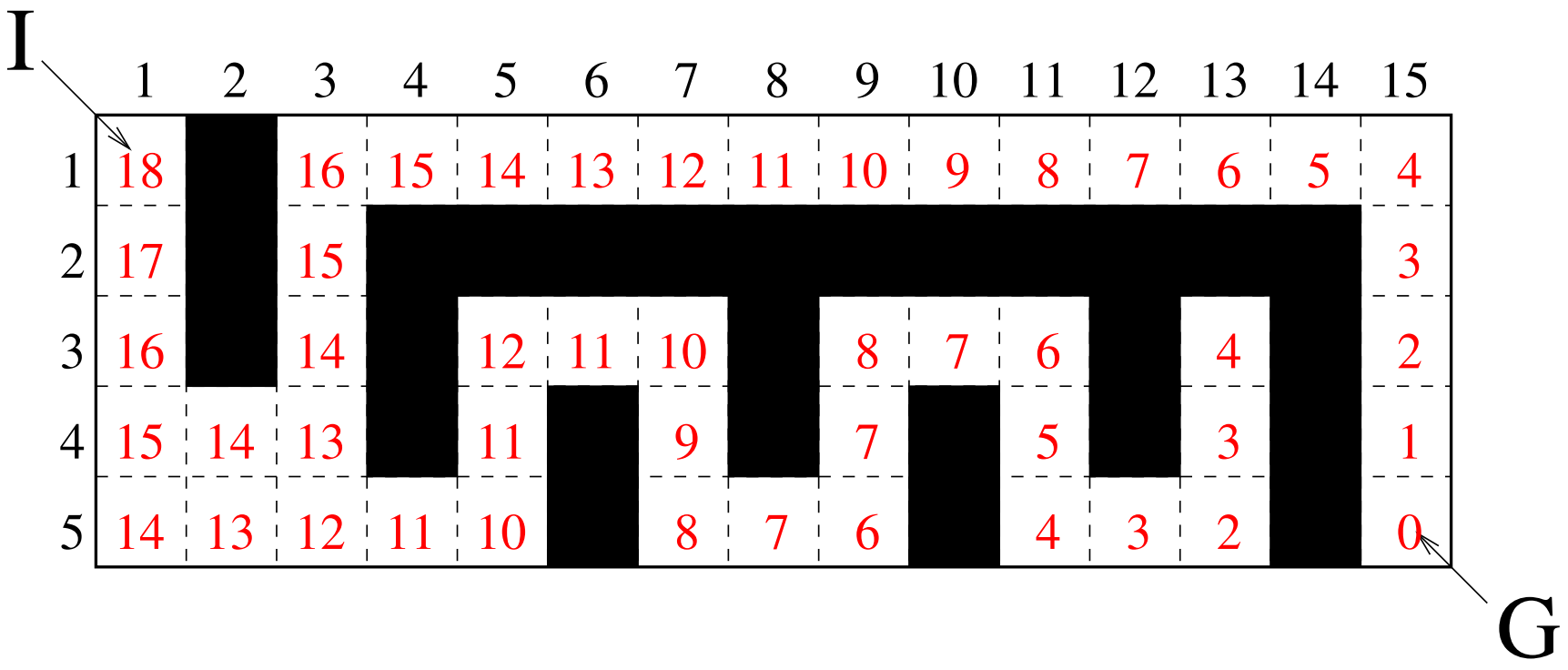
Heuristic Function Pitfalls: Example Path Planning

Manhattan Distance, "accurate h":

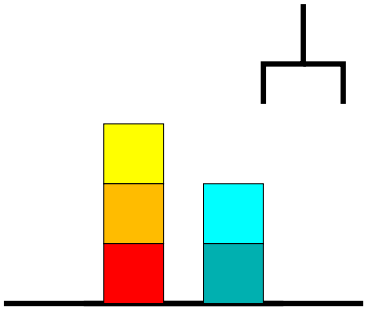


Heuristic Function Pitfalls: Example Path Planning

Manhattan Distance, "inaccurate h":



Questionnaire



- n blocks, 1 hand.
- A single action either takes a block with the hand or puts a block we're holding onto some other block/the table.
- The goal is a set of statements "on(x,y)".

Question!

Consider $h :=$ number of goal statements that are not currently true. Is the error relative to h^* bounded by a constant?

(A): Yes. (B): No.

$\rightarrow$ No. There are examples where the error grows linearly in n. Example: Block b_1 is currently beneath a stack of $b_n, \ldots, b_2$ and the goal is on(b_1, b_2). Then $h(s) = 1$ but $h^*(s) = 2n$ (pick/put-down for each $b_n, \ldots, b_2$; pick/put-on-b_2 for b_1).

Properties of Heuristic Functions

Definition (Admissibility, Consistency). *Let Π be a problem with state space Θ and states S, and let h be a heuristic function for Π. We say that h is admissible if, for all $s \in S$, we have $h(s) \leq h^*(s)$. We say that h is consistent if, for all transitions $s \xrightarrow{a} s'$ in Θ, we have $h(s) - h(s') \leq c(a)$.*

In other words . . .

- Admissibility: lower bound on goal distance.
- Consistency: when applying an action a, the heuristic value cannot decrease by more than the cost of a.

Properties of Heuristic Functions, ctd.

Proposition (Consistency $\implies$ Admissibility). *Let Π be a problem, and let h be a heuristic function for Π. If h is consistent, then h is admissible.*

Proof. We need to show that $h(s) \leq h^*(s)$ for all s. For states s where $h^*(s) = \infty$, this is trivial. For all other states, we show the claim by induction over the length of the cheapest path to a goal state.

Base case: s is a goal state. Then $h(s) = 0$ by definition of heuristic functions, so $h(s) \leq h^*(s) = 0$ as desired.

Step case: Assume the claim holds for all states s' with a cheapest goal path of length n. Say s has a cheapest goal path of length $n+1$, the first transition of which is $s \xrightarrow{a} s'$. By consistency, we have $h(s) - h(s') \leq c(a)$ and thus (a) $h(s) \leq h(s') + c(a)$. By construction, s' has a cheapest goal path of length n and thus, by induction hypothesis, (b) $h(s') \leq h^*(s')$. By construction, (c) $h^*(s) = h^*(s') + c(a)$. Inserting (b) into (a), we get $h(s) \leq h^*(s') + c(a)$. Inserting (c) into the latter, we get $h(s) \leq h^*(s)$ as desired.

Properties of Heuristic Functions: Examples

Admissibility and consistency:

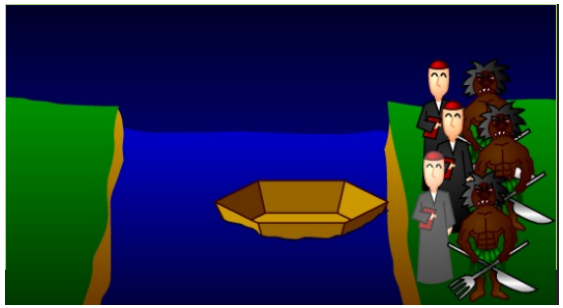
- Is straight line distance admissible/consistent? Yes. Consistency: If you drive 100km, then the straight line distance to Moscow can't decrease by more than 100km.
- Is goal distance of the "reduced puzzle" (slide 15) admissible/consistent? Yes. Consistency: Moving a tile can't decrease goal distance in the reduced puzzle by more than 1. Same for misplaced tiles/Manhattan distance.
- Can somebody come up with an admissible but inconsistent heuristic? To-Moscow with $h(SB) = 1000$, $h(KL) = 100$.

$\rightarrow$ In practice, admissible heuristics are typically consistent.

Inadmissible heuristics:

- Inadmissible heuristics typically arise as approximations of admissible heuristics that are too costly to compute. (We'll meet some examples of this in **Chapter 15**.)

Questionnaire



- 3 missionaries, 3 cannibals.
- Boat that holds ≤ 2.
- Never leave k missionaries alone with $> k$ cannibals.

Question!

Is $h :=$ number of persons at right bank consistent/admissible?

(A): Only consistent.
(B): Only admissible.
(C): None.
(D): Both.

$\rightarrow$ (A): No: If h is consistent then it is admissible, so "only consistent" can't happen (for any heuristic).

$\rightarrow$ (B): No: h is not admissible because a single move of the boat may get more than 1 person to the desired bank (example: 1 missionary and 1 cannibal at the wrong bank, with the boat).

$\rightarrow$ (C): Yes: h is not admissible so it can't be consistent either.

$\rightarrow$ (D): No, see above.

Before We Begin

Systematic search vs. local search:

- Systematic search strategies: No limit on the number of search nodes kept in memory at any point in time.

  → Guarantee to consider all options at some point, thus complete.
- Local search strategies: Keep only one (or a few) search nodes at a time.

  → No systematic exploration of all options, thus incomplete.

Tree search vs. graph search:

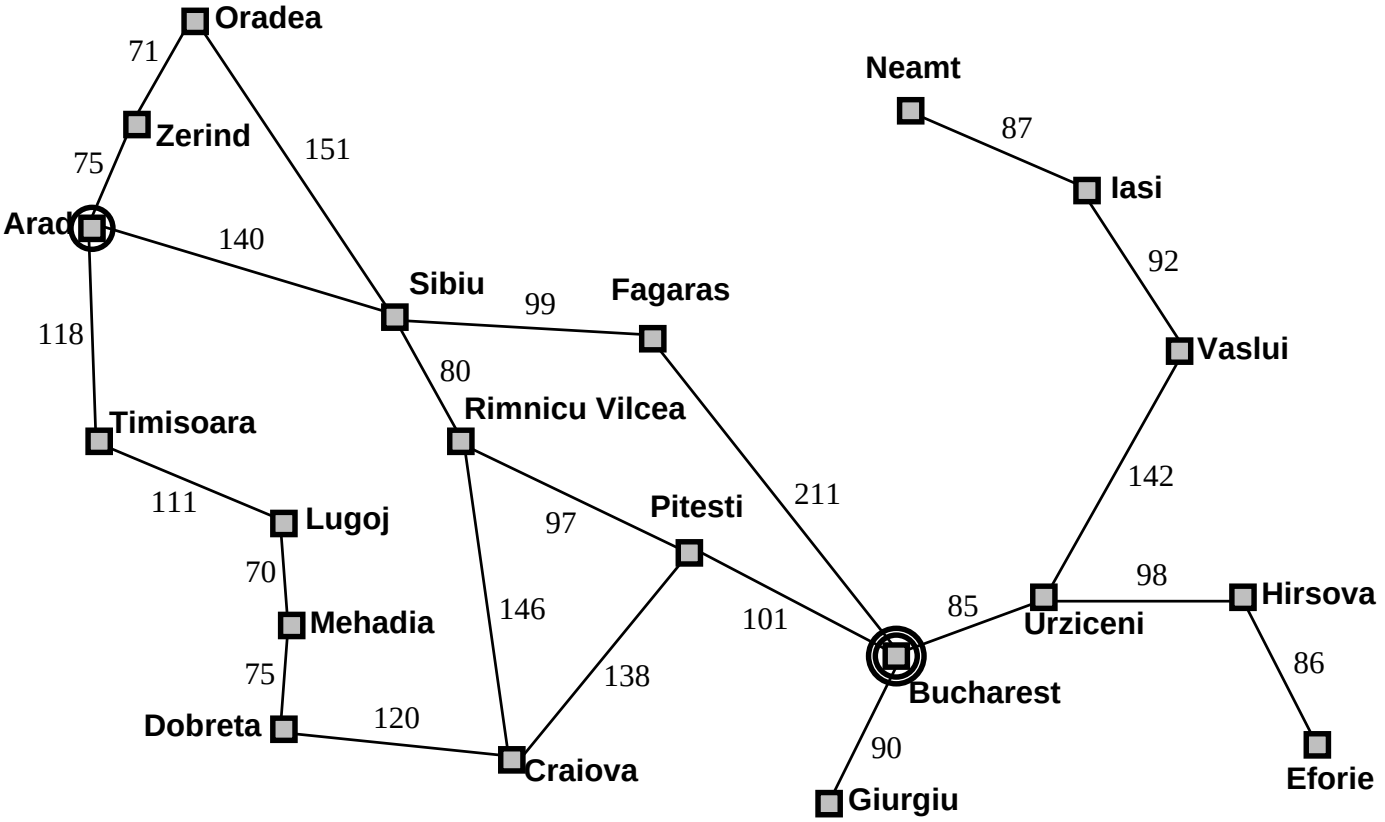
- For the systematic search strategies, we consider graph search algorithms exclusively, i.e., we use duplicate pruning.
- There also are tree search versions of these algorithms. These are easier to understand, but aren't used in practice. (Maintaining a complete open list, the search is memory-intensive anyway.)

Greedy Best-First Search

```
function Greedy Best-First Search(problem) returns a solution, or failure
  node ← a node n with n.state=problem.InitialState
  frontier ← a priority queue ordered by ascending h, only element n
  explored ← empty set of states
  loop do
      if Empty?(frontier) then return failure
      n ← Pop(frontier)
      if problem.GoalTest(n.State) then return Solution(n)
      explored ← explored ∪n.State
      for each action a in problem.Actions(n.State) do
         n′ ← ChildNode(problem,n,a)
         if n′.State∉explored ∪ States(frontier) then Insert(n′, h(n′), frontier)
```

- Frontier ordered by ascending h.
- Duplicates checked at successor generation, against both the frontier and the explored set.

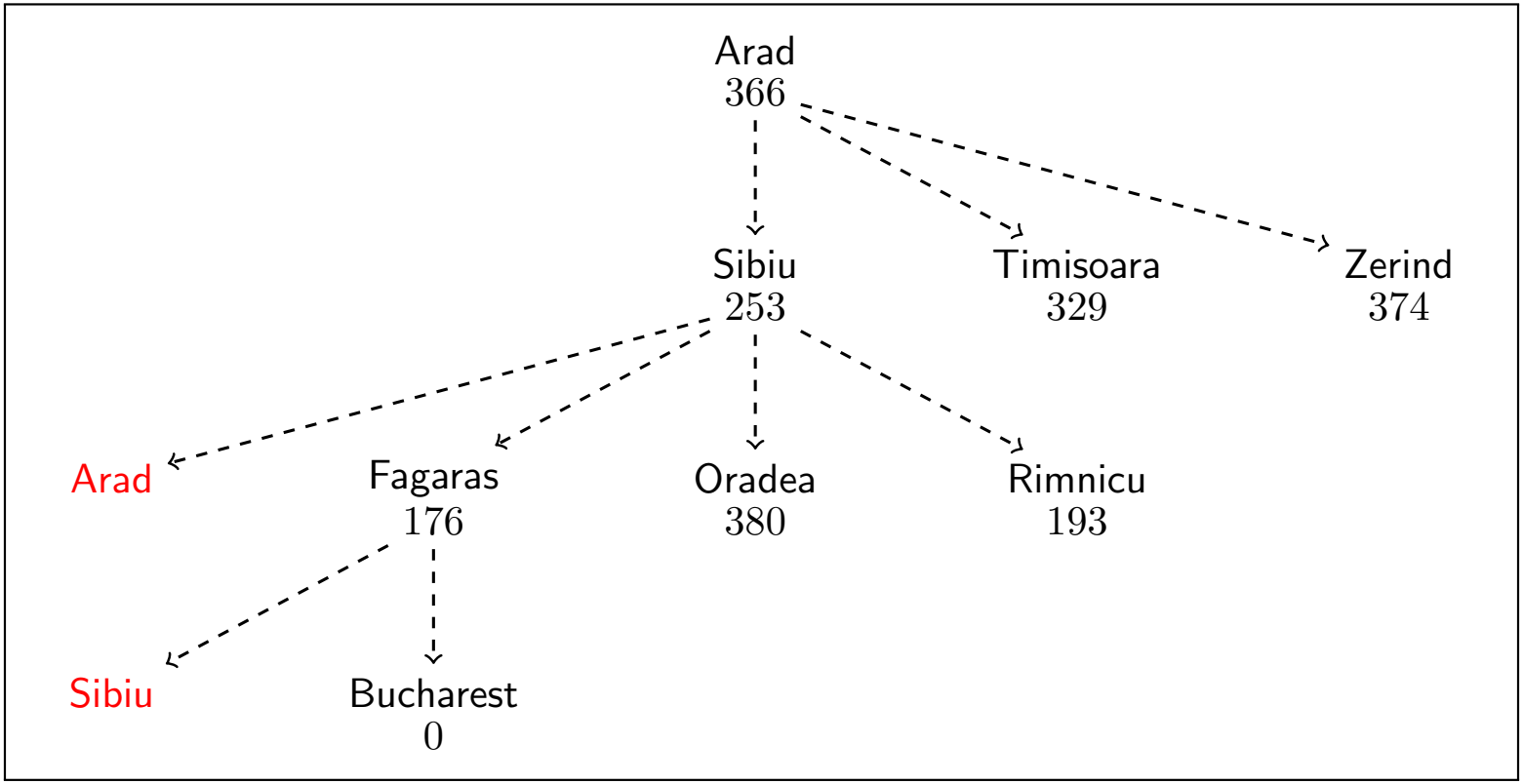
Greedy Best-First Search: Route to Bucharest



Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy Best-First Search: Route to Bucharest

Subscripts: h. Red nodes: removed by duplicate pruning.



Greedy Best-First Search: Guarantees

- Completeness: Yes, thanks to duplicate elimination and our assumption that the state space is finite.
- Optimality? No (h might lead us to Moscow via Paris).

Can we do better than this?

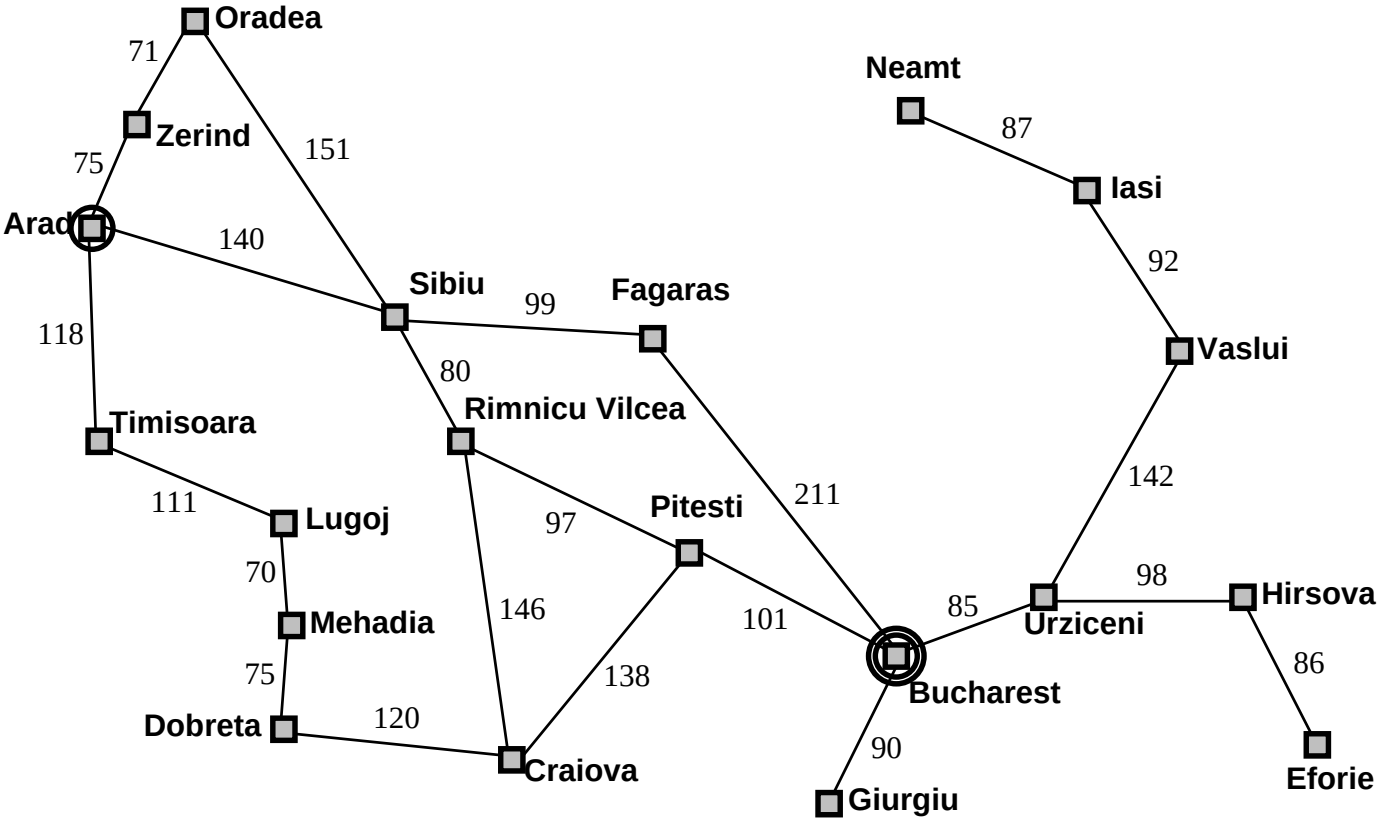
$\rightarrow$ Yes: A^* is complete *and* optimal.

A*

```
function A*(problem) returns a solution, or failure
  node ← a node n with n.State=problem.InitialState
  frontier ← a priority queue ordered by ascending g + h, only element n
  explored ← empty set of states
  loop do
      if Empty?(frontier) then return failure
      n ← Pop(frontier)
      if problem.GoalTest(n.State) then return Solution(n)
      explored ← explored∪n.State
      for each action a in problem.Actions(n.State) do
         n' ← ChildNode(problem,n,a)
         if n'.State∉explored ∪ States(frontier) then
            Insert(n', g(n') + h(n'), frontier)
         else if ex. n'' ∈frontier s.t. n''.State= n'.State and g(n') < g(n'') then
              replace n'' in frontier with n'
```

- Frontier ordered by ascending $g + h$.
- Duplicates handled exactly as in uniform-cost search.

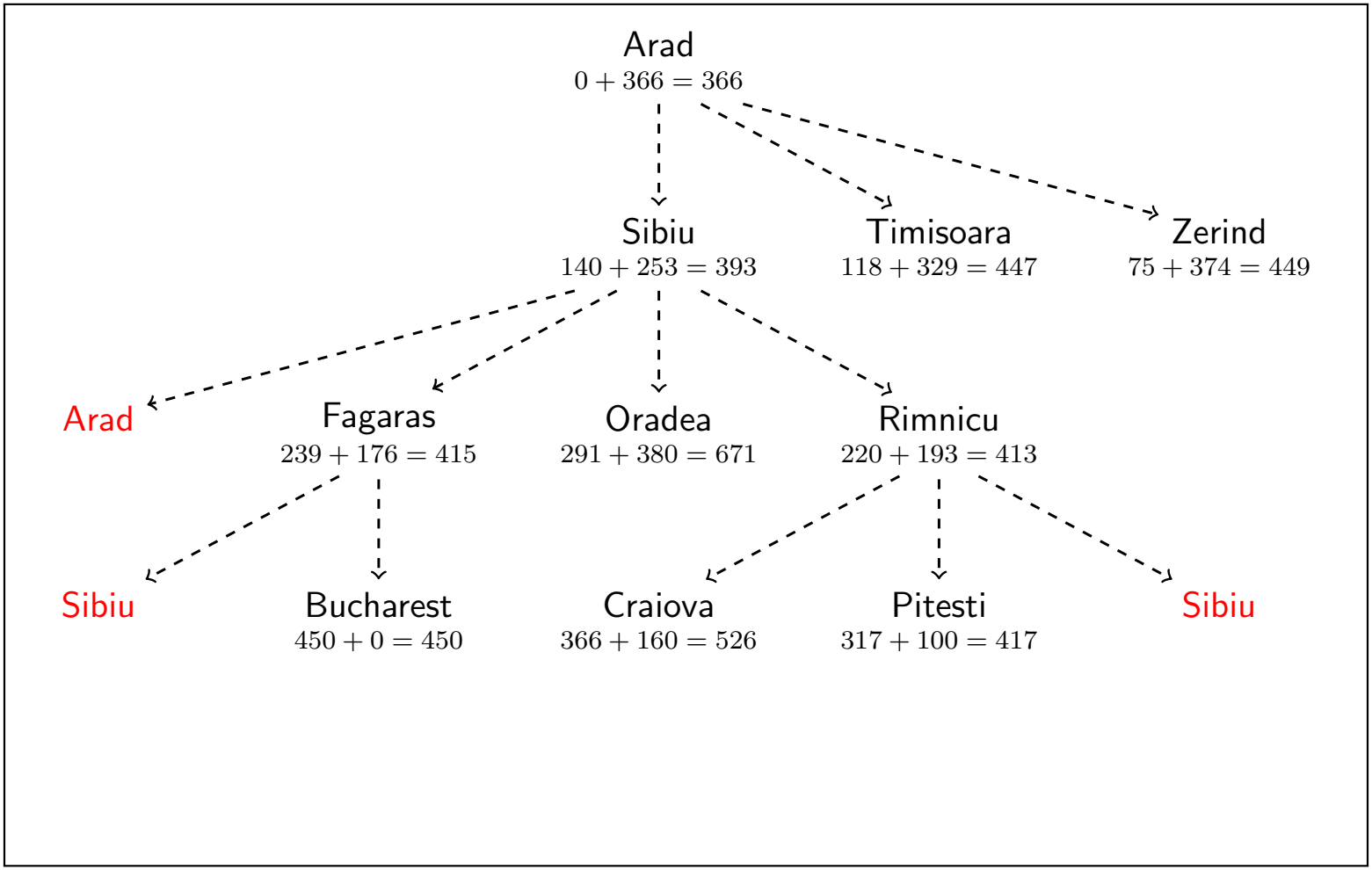
A*: Route to Bucharest



Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

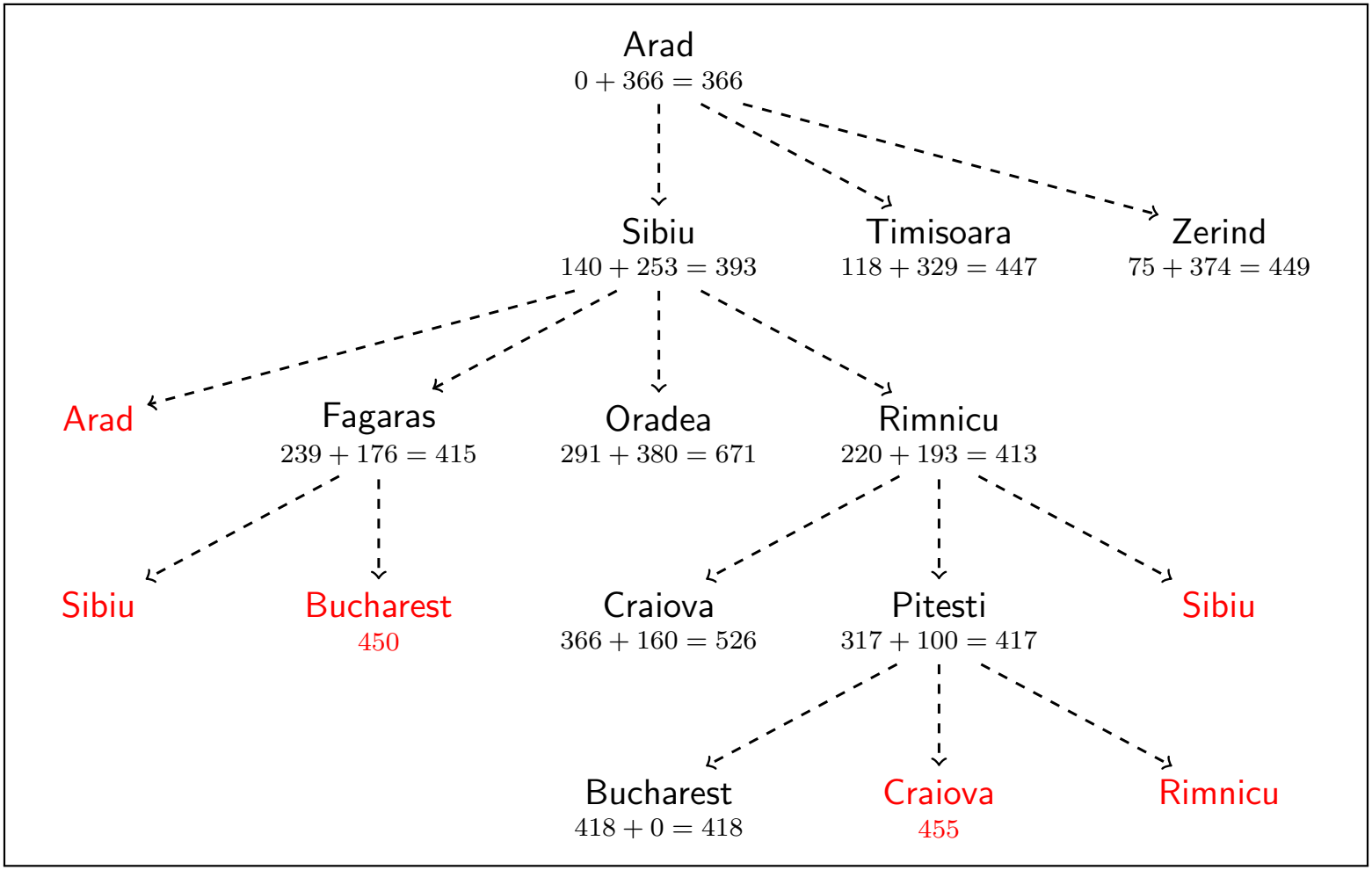
A^*: Route to Bucharest

Subscripts: $g + h$. Red nodes: removed by duplicate pruning (without subscript), or because of better path (with subscript g).



A^*: Route to Bucharest

Subscripts: $g + h$. Red nodes: removed by duplicate pruning (without subscript), or because of better path (with subscript g).



Questionnaire

Question!

If we set $h(s) := 0$ for all states s, what does greedy best-first search become?

(A): Breadth-first search
(B): Depth-first search
(C): Uniform-cost search
(D): Depth-limited search

$\rightarrow$ h implies no node ordering at all. The search order is determined by how we break ties in the open list. We *basically* get (A) with FIFO, (B) with LIFO, and (C) when ordering on g (in each case, differences remain in the handling of duplicate states etc).

Question!

If we set $h(s) := 0$ for all states s, what does A^* become?

(A): Breadth-first search
(B): Depth-first search
(C): Uniform-cost search
(D): Depth-limited search

$\rightarrow$ (C): The *only* difference between A^* and uniform-cost search is the use of $g + h$ instead of g to order the open list.

Optimality of A^*: Proof, Step 1

Idea: The proof is via a correspondence to uniform-cost search.

$\rightarrow$ Step 1: Capture the heuristic function in terms of action costs.

Definition. *Let Π be a problem with state space $\Theta = (S, A, c, T, I, S^G)$, and let h be a consistent heuristic function for Π. We define the h-weighted state space as $\Theta^h = (S, A^h, c^h, T^h, I, S^G)$ where:*

- $A^h := \{a[s, s'] \mid a \in A, s \in S, s' \in S, (s, a, s') \in T\}$.
- $c^h : A^h \mapsto \mathbb{R}_0^+$ *is defined by* $c^h(a[s, s']) := c(a) - [h(s) - h(s')]$.
- $T^h = \{(s, a[s, s'], s') \mid (s, a, s') \in T\}$.

$\rightarrow$ Subtract, from each action cost, the "gain in heuristic value".

Lemma. Θ^h *is well-defined, i.e.,* $c(a) - [h(s) - h(s')] \geq 0$.

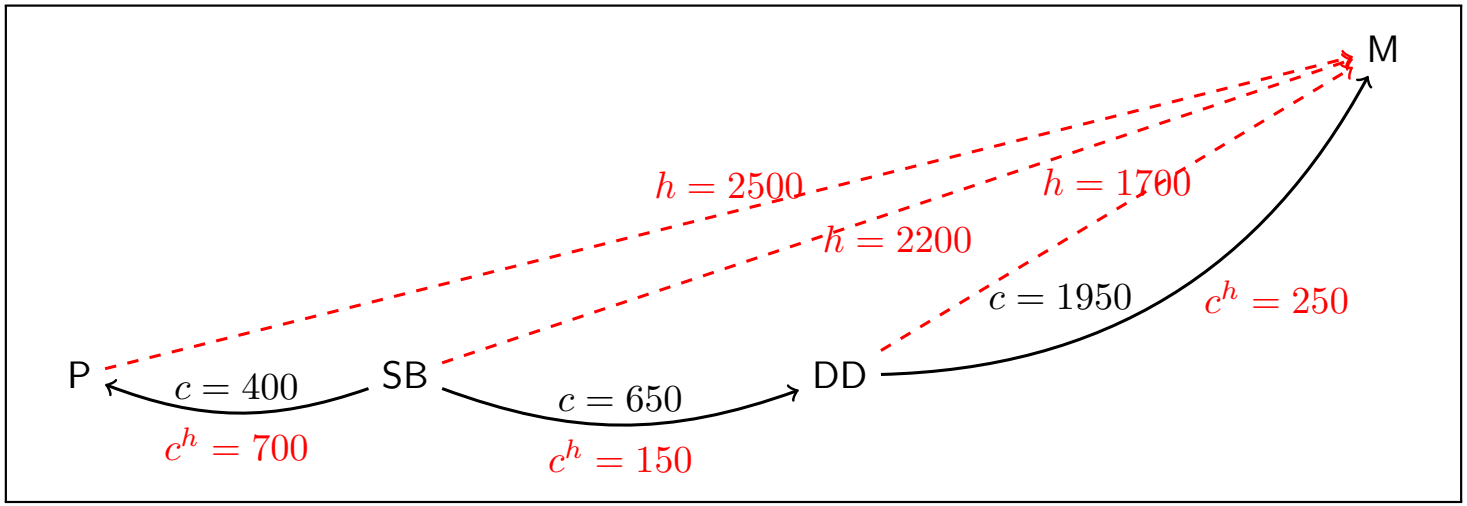
Proof. By consistency, $h(s) - h(s') \leq c(a)$. This implies the claim.

Optimality of A^*: Proof – Illustration

Example: Finding a route from SB to Moscow

- States: P (Paris), SB, DD (Dresden), M (Moscow).
- Actions: $c(\text{SBtoP}) = 400$, $c(\text{SBtoDD}) = 650$, $c(\text{DDtoM}) = 1950$.
- Heuristic (straight line distance): $h(\text{Paris}) = 2500$, $h(\text{SB}) = 2200$, $h(\text{DD}) = 1700$.

Θ and Θ^h: (Proof Step 1, Definition on previous slide)



Optimal solution: (Proof Step 2, Lemma (A) on next slide)

- $g^* = 2600$ in Θ and $g^* = 400 = 2600 - h(\text{SB})$ in Θ^h.

Optimality of A^*: Proof, Step 2

→ Step 2: Identify the correspondence.

Lemma (A). *Θ and Θ^h have the same optimal solutions.*

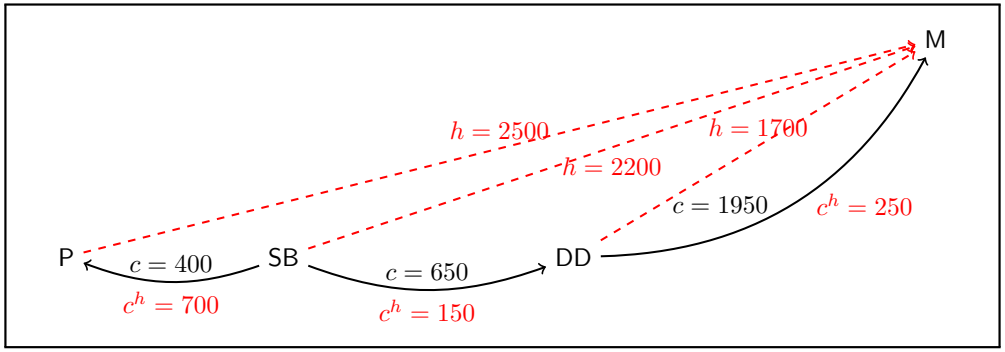
Proof. Let $I = s_0 \xrightarrow{a_1} s_1, \ldots, s_{n-1} \xrightarrow{a_n} s_n$ be a solution in Θ, $s_n \in S^G$. The cost of the same path in Θ^h is $[-h(s_0) + c(a_1) + h(s_1)] + [-h(s_1) + c(a_2) + h(s_2)] + \cdots + [-h(s_{n-1}) + c(a_n) + h(s_n)] = \sum_{i=1}^{n} c(a_i) - h(I) + h(s_n) = [\sum_{i=1}^{n} c(a_i)] - h(I)$. Thus the costs of solution paths in Θ^h are those of Θ, minus a constant. The claim follows.

Lemma (B). *The search space of A^* on Θ is isomorphic to that of uniform-cost search on Θ^h.*

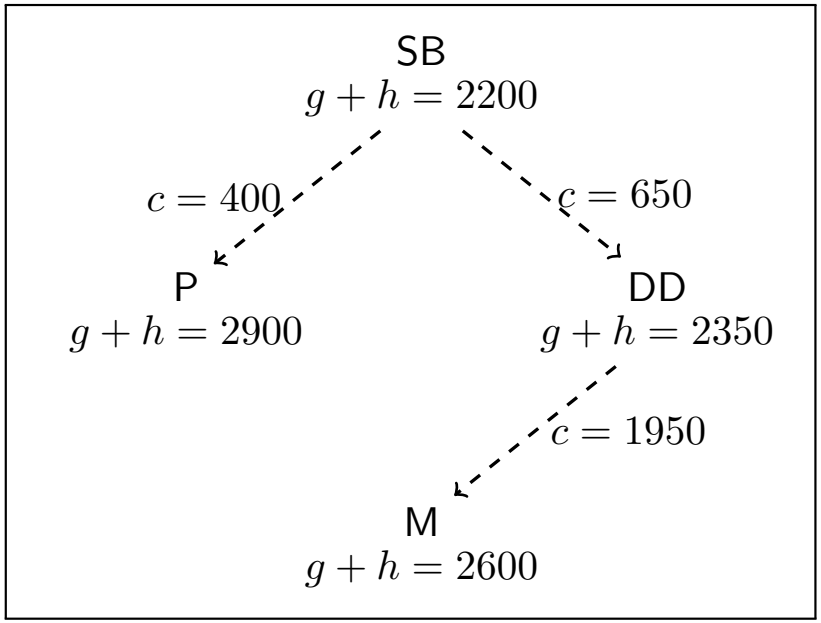
Proof. Let $I = s_0 \xrightarrow{a_1} s_1, \ldots, s_{n-1} \xrightarrow{a_n} s_n$ be any path in Θ. The $g + h$ value, used by A^*, is $[\sum_{i=1}^{n} c(a_i)] + h(s_n)$. The g value in Θ^h, used by uniform-cost search on Θ^h, is $[\sum_{i=1}^{n} c(a_i)] - h(I) + h(s_n)$ (see previous proof). The difference $-h(I)$ is constant, so the ordering of open list is the same. QED as the duplicate elimination is identical.

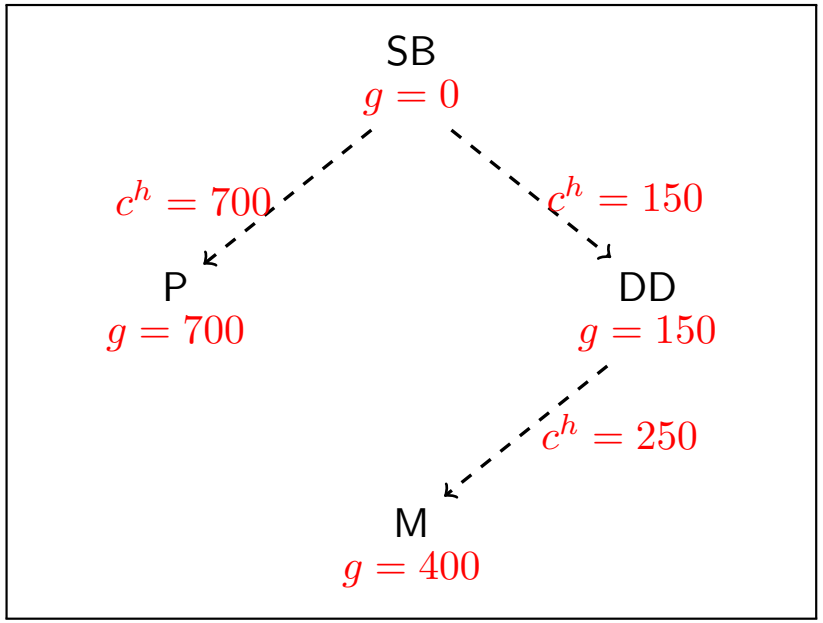
Optimality of A^*: Proof – Illustration

Θ and Θ^h:



A^* on Θ (left) and uniform-cost search on Θ^h (right): (Proof Step 2, Lemma (B) on previous slide)





Optimality of A^*: Proof, Step 3

→ Step 3: Put the pieces together.

Theorem (Optimality of A^*). *Let Π be a problem, and let h be a heuristic function for Π. If h is consistent, then the solution returned by A^* (if any) is optimal.*

Proof. Denote by Θ the state space of Π. Let $\vec{s}(A^*, \Theta)$ be the solution returned by A^* run on Θ. Denote by $\vec{S}(UCS, \Theta^h)$ the set of solutions that could in principle be returned by uniform-cost search run on Θ^h.

By Lemma (B), $\vec{s}(A^*, \Theta) \in \vec{S}(UCS, \Theta^h)$: With appropriate tie-breaking between nodes with identical g value, uniform cost search will return $\vec{s}(A^*, \Theta)$. By optimality of uniform-cost search (**Chapter 4**), every element of $\vec{S}(UCS, \Theta^h)$ is an optimal solution for Θ^h. Thus $\vec{s}(A^*, \Theta)$ is an optimal solution for Θ^h. With Lemma (A), this implies that $\vec{s}(A^*, \Theta)$ is an optimal solution for Θ, which is what we needed to prove.

Optimality of A^*: Different Variants

- Our variant of A^* does duplicate elimination but not re-opening.
- Re-opening: check, when generating a node n containing state s that is already in the explored set, whether *(*) the new path to s is cheaper*. If so, remove s from the explored set and insert n into the frontier.
- With a consistent heuristic, (*) can't happen so we don't need re-opening for optimality.
- Given admissible but inconsistent h, if we either don't use duplicate elimination at all, or use duplicate elimination *with* re-opening, then A^* is optimal as well. Hence the well-known statement "A^* is optimal if h is admissible".

  $\rightarrow$ But for our variant (as per slide 29), being admissible is NOT enough for optimality! Frequent implementation bug!

$\rightarrow$ Recall: In practice, admissible heuristics are typically consistent. That's why I chose to present this variant.

And now, let's relax a bit ...

https://www.youtube.com/channel/UCXsg2PT89piH2vB_R0Kd-Lw
https://www.movingai.com/SAS/

More videos: http://movingai.com/

→ Illustrations of various issues in heuristic search/A^* that go deeper than our introductory material here.

Provable Performance Bounds: Extreme Case

Let's consider an extreme case: What happens if $h = h^*$?

Greedy Best-First Search:

- If all action costs are strictly positive, when we expand a state, at least one of its successors has strictly smaller h. The search space is linear in the length of the solution.
- If there are 0-cost actions, the search space may still be exponentially big (e.g., if all actions costs are 0 then $h^* = 0$).

$\mathbf{A}^*$:

- If all action costs are strictly positive, *and* we break ties $(g(n) + h(n) = g(n') + h(n'))$ by smaller h, then the search space is linear in the length of the solution.
- Otherwise, the search space may still be exponentially big.

Provable Performance Bounds: More Interesting Cases?

"Almost perfect" heuristics:

$$|h^*(n) - h(n)| \leq c \text{ for a constant } c$$

- Basically the only thing that lead to <u>some</u> interesting results.
- If the state space is a tree (only one path to every state), and there is only one goal state: linear in the length of the solution [Gaschnig (1977)].
- But if these additional restrictions do not hold: exponential even for very simple problems and for $c = 1$ [Helmert and Röger (2008)]!

→ Systematically analyzing the practical behavior of heuristic search remains one of the biggest research challenges.

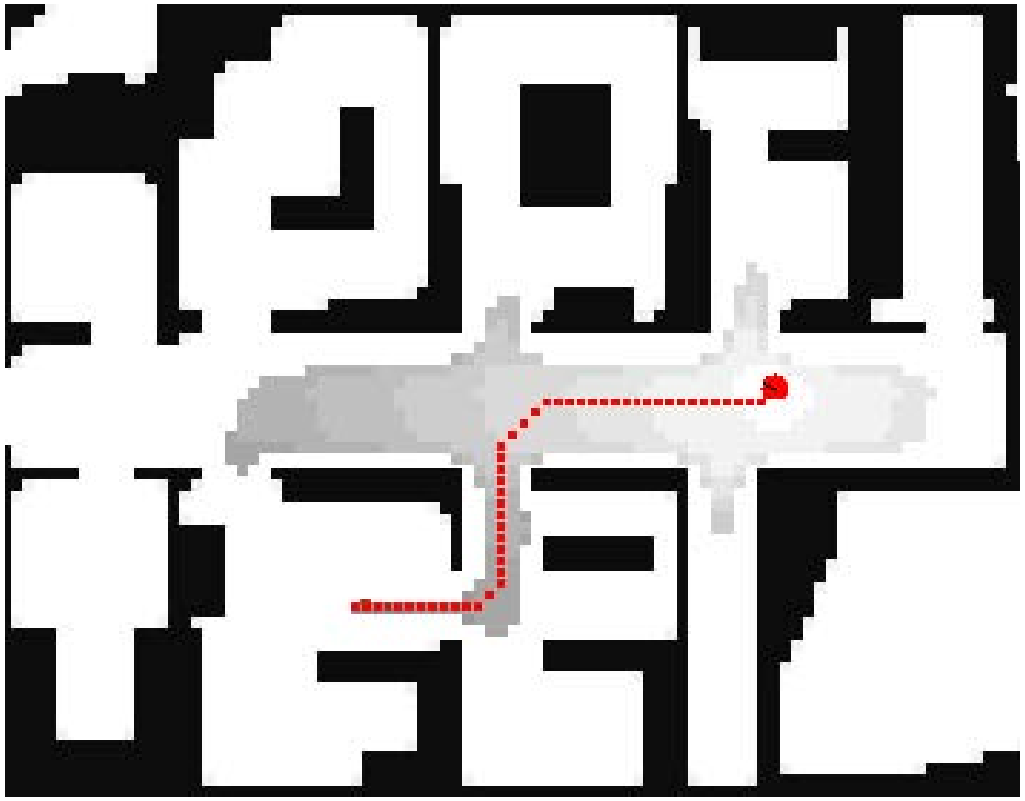
→ There is little hope to prove practical sub-exponential-search bounds. (But there are some interesting insights one *can* gain → FAI.)

Empirical Performance: A^* in the 8-Puzzle

Without Duplicate Elimination; d = length of solution:

	Number of search nodes generated		
	Iterative	A^* with	
d	Deepening Search	misplaced tiles h	Manhattan distance h
2	10	6	6
4	112	13	12
6	680	20	18
8	6384	39	25
10	47127	93	39
12	3644035	227	73
14	-	539	113
16	-	1301	211
18	-	3056	363
20	-	7276	676
22	-	18094	1219
24	-	39135	1641

Empirical Performance: A^* in Path Planning

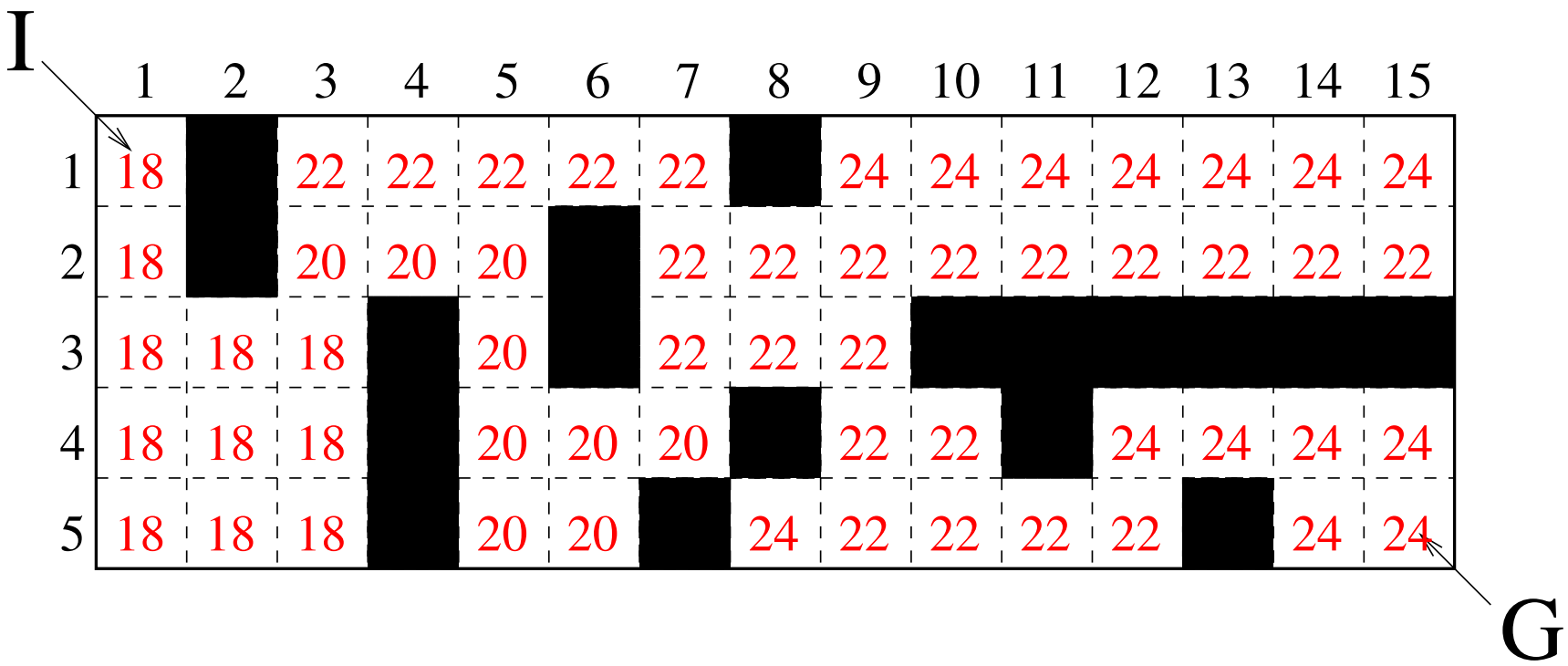


Live Demo vs. Breadth-First Search:

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

Greedy Best-First vs. A^*: Illustration Path Planning

$\mathrm{A}^*(g+h)$, "accurate h":

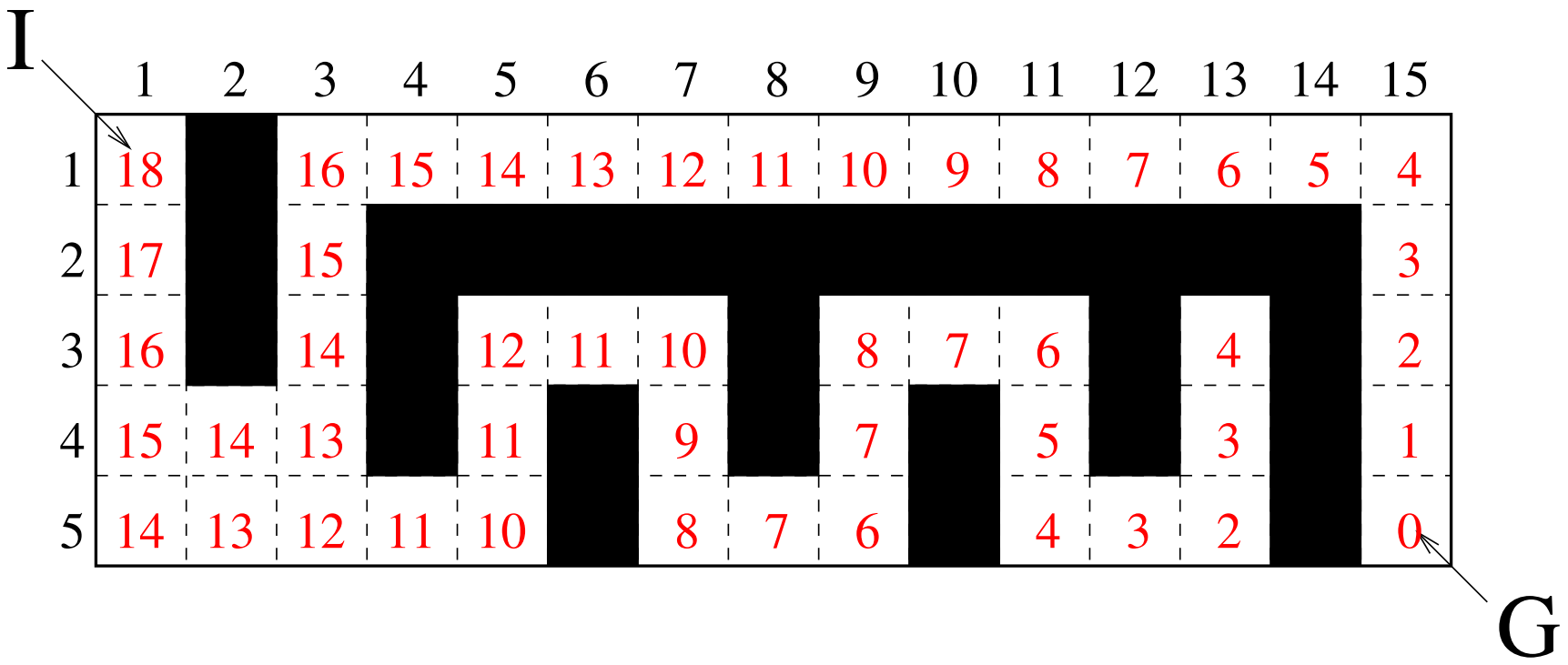


$\rightarrow$ In A^* with a consistent heuristic, $g+h$ always increases monotonically (h cannot *de*crease by more than g *in*creases).

$\rightarrow$ We need more search, in the "right upper half". This is typical: Greedy best-first search tends to be faster than A^*.

Greedy Best-First vs. A^*: Illustration Path Planning

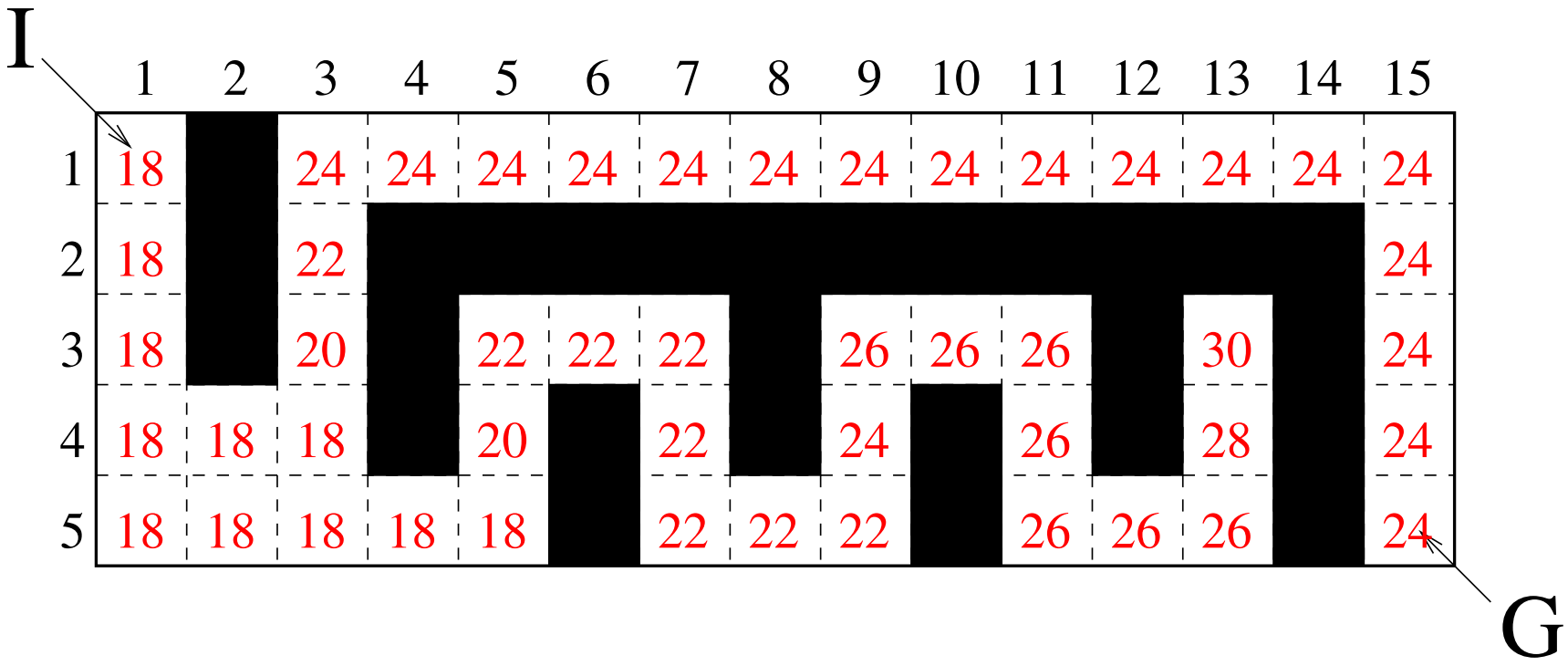
Greedy best-first search, "inaccurate h":



$\rightarrow$ Search will be mis-guided into the "dead-end street".

Greedy Best-First vs. A^*: Illustration Path Planning

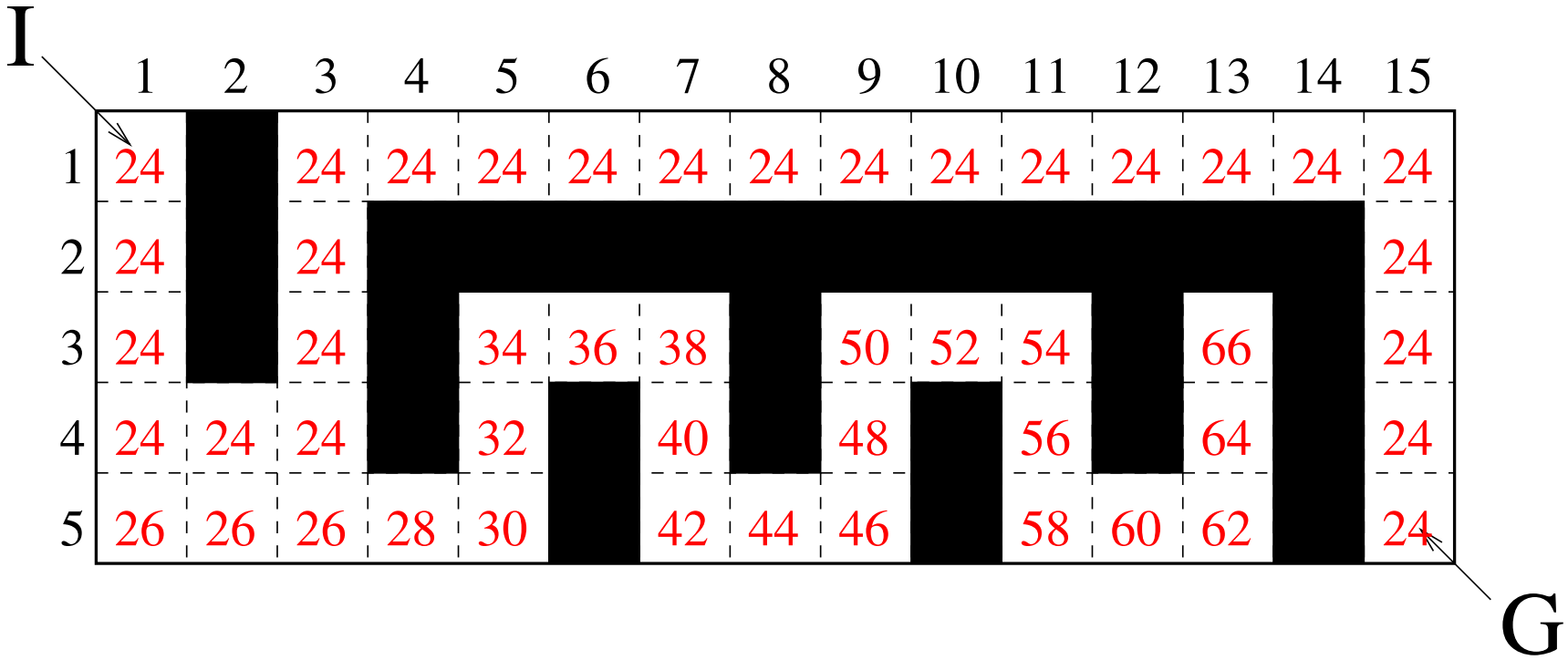
$\mathrm{A}^*(g+h)$, "inaccurate h":



$\rightarrow$ We will search less of the "dead-end street". For very "bad heuristics", $g+h$ gives better search guidance than h, and A^* is faster.

Greedy Best-First vs. A^*: Illustration Path Planning

$\mathrm{A}^*(g+h)$ using h^*:



$\rightarrow$ With $h = h^*$, $g + h$ remains constant on optimal paths.

Questionnaire

Question!

1. Is A^* always at least as fast as uniform-cost search? 2. Does it always expand at most as many states?

(A): No and no.
(B): Yes and no.
(C): No and Yes.
(D): Yes and yes.

$\rightarrow$ Regarding 1.: No, simply because computing h takes time. So the overall runtime may get worse.

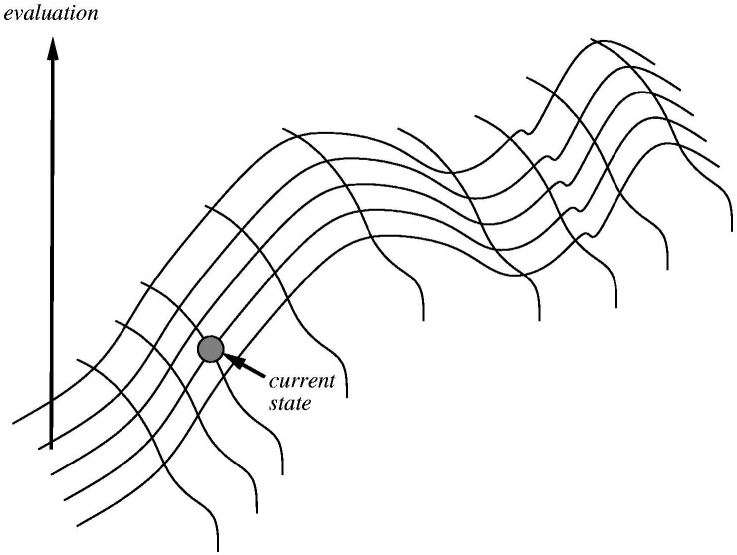
$\rightarrow$ Regarding 2.: "Yes, but". Setting $h(s) := 0$ for uniform-cost search, both algorithms expand *only* states s where $g^*(s) + h(s) \leq g^*$, and *must* expand all states where $g^*(s) + h(s) < g^*$.
Non-zero h can only reduce the latter. Which s with $g^*(s) + h(s) = g^*$ are explored depends on the tie-breaking used (which state to expand if there is more than one state with minimal $g + h$ in the open list). So the answer is "yes but only if the tie-breaking in both algorithms is the same".

Local Search

Do *you* "think through all possible options" before choosing your path to the Mensa?

→ Sometimes, "going where your nose leads you" works quite well.

What is the computer's "nose"?



→ Local search takes decisions based on the h values of immediate neighbor states.

Hill Climbing

function Hill-Climbing*(problem)*
  $n \leftarrow$ a node n with n.*state*=*problem.InitialState*
  loop do
    $n' \leftarrow$ among child nodes n' of n with minimal $h(n')$,
      randomly pick one
    if $h(n') \geq h(n)$ **then return** *the path to* n
    $n \leftarrow n'$

$\rightarrow$ Hill-Climbing keeps choosing actions leading to a direct successor state with best heuristic value. It stops when no more immediate improvements can be made.

- Alternative name (more fitting, here): Gradient-Descent.
- Often used in optimization problems where all "states" are feasible solutions, and we can choose the search neighborhood ("child nodes") freely. (Return just n.State, rather than the path to n)

Local Search: Guarantees and Complexity

Guarantees:

- Completeness: No. Search ends when no more immediate improvements can be made (= local minimum, up next). This is not guaranteed to be a solution.
- Optimality: No, for the same reason.

Complexity:

- Time: We stop once the value doesn't strictly increase, so the state space size is a bound.
  $\rightarrow$ Note: This bound is (a) huge, and (b) applies to a single run of Hill-Climbing, which typically does not find a solution.
- Memory: Basically no consumption: $O(b)$ states at any moment in time.

Hill Climbing: Example 8-Queens Problem

18	12	14	13	13	12	14	14
14	16	13	15	12	14	12	16
14	12	18	13	15	12	14	14
15	14	14	♛	13	16	13	16
♛	14	17	15	♛	14	16	16
17	♛	16	18	15	♛	15	♛
18	14	♛	15	15	14	♛	16
14	14	13	17	12	14	12	18

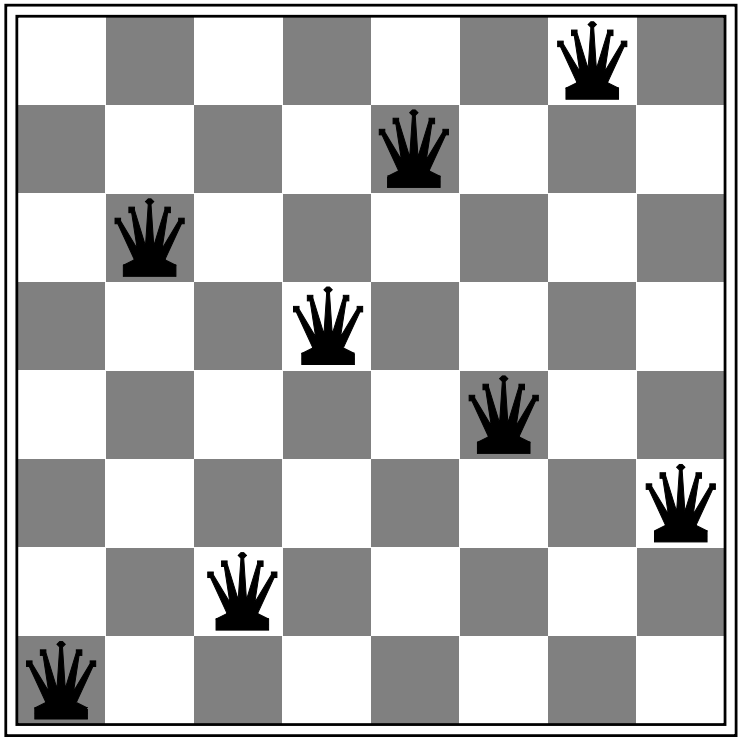
Problem: Place the queens so that they don't attack each other.

Heuristic: Number of pairs attacking each other.

Neighborhood: Move any queen within its column.

→ Starting from random initialization, solves only 14% of cases.

A Local Minimum in the 8-Queens Problem



→ Current h value is? 1. To get $h = 0$, we must move either of the 2 queens involved in the conflict. But every such move results in at least 2 new conflicts, so this is a local minimum.

Local Search: Difficulties

Difficulties:

- Local minima: All neighbors look worse (have a worse h value) than the current state (e.g.: previous slide).

  $\rightarrow$ If we stop, the solution may be sub-optimal (or not even feasible). If we don't stop, where to go next?

- Plateaus: All neighbors have the same h value as the current state.

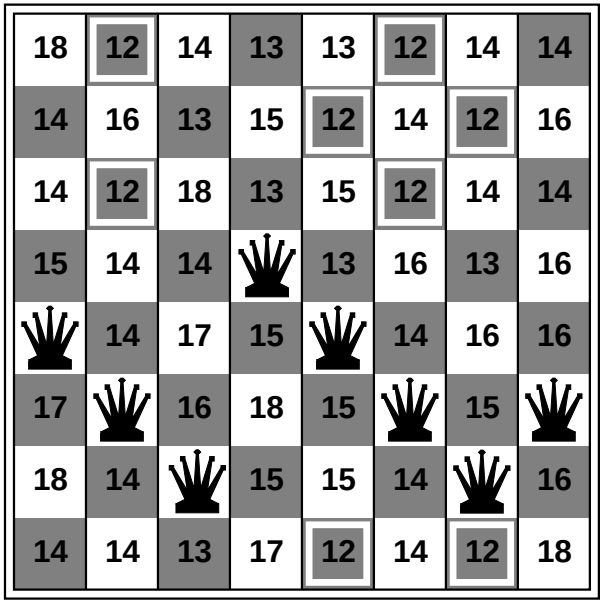
  $\rightarrow$ Moves will be chosen completely at random.

Strategies addressing these:

- Re-start when reaching a local minimum, or when we have spent a certain amount of time without "making progress".
- Do random walks in the hope that these will lead out of the local minimum/plateau.

$\rightarrow$ Configuring these strategies requires lots of algorithm parameters. Selecting good values is a big issue in practice. (Cross your fingers ...)

Hill Climbing With Sideways Moves: Example 8-Queens



Heuristic: Number of pairs attacking each other.

Neighborhood: Move any queen within its column.

Sideways Moves: In contrast to slide 50, allow up to 100 consecutive moves in which the h value does not get better.

→ Starting from random initialization, solves 94% of cases!

→ Successful local search procedures often combine randomness (exploration) with following the heuristic (exploitation).

One (Out of Many) Other Local Search Strategies: Simulated Annealing

Idea: Taking inspiration from processes in physics (objects cooling down), inject "noise" systematically: first a lot, then gradually less.

```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
  inputs: problem, a problem
          schedule, a mapping from time to "temperature"

  current ← MAKE-NODE(problem.INITIAL-STATE)
  for t = 1 to ∞ do
      T ← schedule(t)
      if T = 0 then return current
      next ← a randomly selected successor of current
      ΔE ← next.VALUE − current.VALUE
      if ΔE > 0 then current ← next
      else current ← next only with probability e^(ΔE/T)
```

→ Used since early 80s for VSLI Layout and other optimization problems.

Questionnaire

Question!

Can local minima occur in route planning with h :=straight line distance?

(A): Yes. (B): No.

$\rightarrow$ Yes, namely if, from the current position, all roads lead away from the target. For example, following straight line distance from Rome to Egypt will lead you to? A beach in southern Italy ... (similar for the dead-end street with Manhattan Distance in the path planning "bad case", cf. slide 18).

Questionnaire, ctd.

Question!

What is the maximum size of plateaus in the 15-puzzle with $h :=$Manhattan distance?

(A): 0 (B): 1

(C): 2 (D): ∞

$\rightarrow$ 0: Any move in the 15-puzzle changes the position of exactly one tile x. So the Manhattan distance changes for x and remains the same for all other tiles, thus the overall Manhattan distance changes. So the h value of every neighbor is different from the current one, and there aren't any plateaus.

Summary

- Heuristic functions h map each state to an estimate of its goal distance. This provides the search with knowledge about the problem at hand, thus making it more focussed.
- h is admissible if it lower-bounds goal distance. h is consistent if applying an action cannot reduce its value by more than the action's cost. Consistency implies admissibility. In practice, admissible heuristics are typically consistent.
- Greedy best-first search explores states by increasing h. It is complete but not optimal.
- A^* explores states by increasing $g + h$. It is complete. If h is consistent, then A^* is optimal. (If h is admissible but not consistent, then we need to use re-opening to guarantee optimality.)
- Local search takes decisions based on its direct neighborhood. It is neither complete nor optimal, and suffers from local minima and plateaus. Nevertheless, it is often successful in practice.

Topics We Didn't Cover Here

- Bounded Sub-optimal Search: Giving a guarantee weaker than "optimal" on the solution, e.g., within a constant factor W of optimal.
- Limited-Memory Heuristic Search: Hybrids of A^* with depth-first search (using linear memory), algorithms allowing to make best use of a given amount M of memory, ...
- External Memory Search: Store the open/closed list on the hard drive, group states to minimize the number of drive accesses.
- Search on the GPU: How to use the GPU for part of the search work?
- Real-Time Search: What if there is a fixed deadline by which we must return a solution? (Often: fractions of seconds ...)
- Lifelong Search: When our problem changes, how can we re-use information from previous searches?
- Non-Deterministic Actions: What if there are several possible outcomes?
- Partial Observability: What if parts of the world state are unknown?
- Reinforcement Learning Problems: What if, a priori, the solver does not know anything about the world it is acting in?

Reading

- *Chapter 3: Solving Problems by Searching*, Sections 3.5 and 3.6 [Russell and Norvig (2010)].

  Content: Section 3.5: A less formal account of what I cover in "Systematic Search Strategies". My main changes pertain to making precise how Greedy Best-First Search and A^* handle duplcate checking: Imho, with respect to this aspect RN is *much* too vague. For A^*, not getting this right is the primary source of bugs leading to non-optimal solutions.

  Section 3.6 (and parts of Section 3.5): A less formal account of what I cover in "Heuristic Functions". Gives many complementary explanations, nice as additional background reading.

- *Chapter 4: Beyond Classical Search*, Section 4.1 [Russell and Norvig (2010)].

  Content: Similar to what I cover in "Local Search Strategies", except it mistakenly acts as if local search could not be used to solve classical search problems. Discusses also Genetic Algorithms, and is nice as additional background reading.

References I

John Gaschnig. Exactly how good are heuristics?: Toward a realistic predictive theory of best-first search. In *Proceedings of the 5th International Joint Conference on Artificial Intelligence (IJCAI'77)*, pages 434–441, Cambridge, MA, August 1977. William Kaufmann.

Malte Helmert and Gabriele Röger. How good is almost perfect? In Dieter Fox and Carla Gomes, editors, *Proceedings of the 23rd National Conference of the American Association for Artificial Intelligence (AAAI'08)*, pages 944–949, Chicago, Illinois, USA, July 2008. AAAI Press.

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