Fondamenti di Intelligenza Artificiale

4. Classical Search, Part II: Informed Search

How to Not Play Stupid When Solving a Problem

Prof Sara Bernardini
bernardini@diag.uniroma1.it
www.sara-bernardini.com

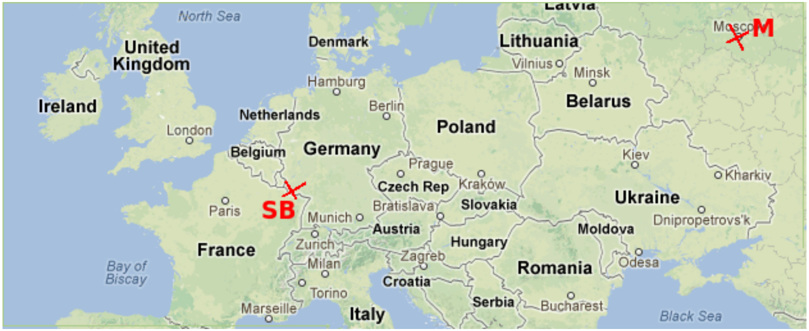
SAPIENZA
UNIVERSITÀ DI ROMA

Spring Term

Agenda

(Not) Playing Stupid

→ Problem: Find a route from Saarbrücken 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 zero knowledge of the problem it is solving.

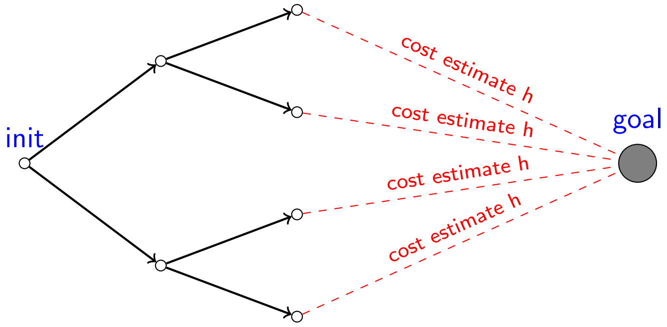
  $\rightarrow$ It can't "focus on roads that go the right direction", because it has no idea what "the right direction" is.

- 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 $\implies$ it 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 $h(s)$.

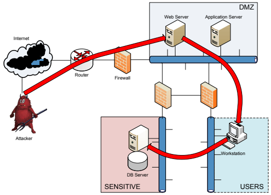
Some Applications

GPS

Robotics

Video Games

Network Security



Reminder: Our Agenda for This Topic

→ Our treatment of the topic "Classical Search" consists of Chapters 3 and 4.

- **Chapter 3:** 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?

  → 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.

  → How to exploit the knowledge in a systematic way?

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

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

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!)

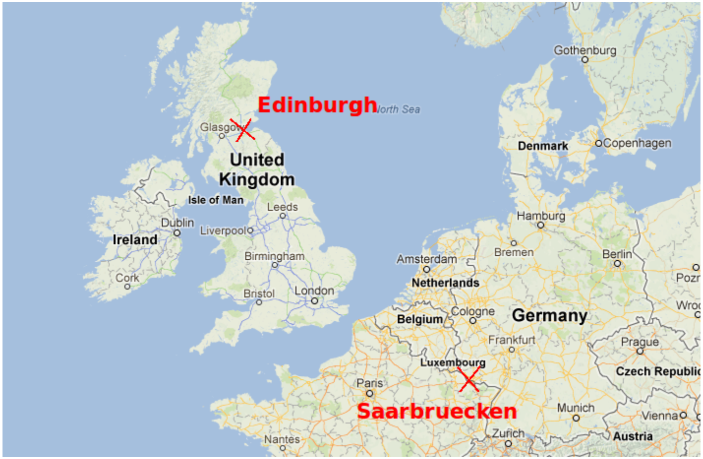
- We want h to be accurate (aka: informative), i.e., "close to" the actual goal distance.
- We also want it to be fast, i.e., a small overhead for computing h.
- These two wishes are in contradiction!

  $\rightarrow$ 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.

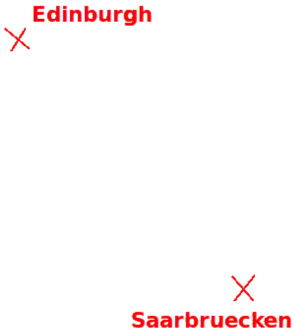
So, how to? $\rightarrow$ Given a problem Π, a heuristic function h for Π can be obtained as goal distance within a simplified (relaxed) problem Π'.

Heuristic Functions from Relaxed Problems: Example 1



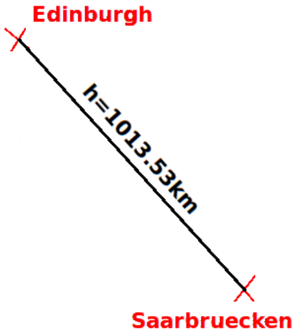
Problem II: Find a route from Saarbruecken To Edinburgh.

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

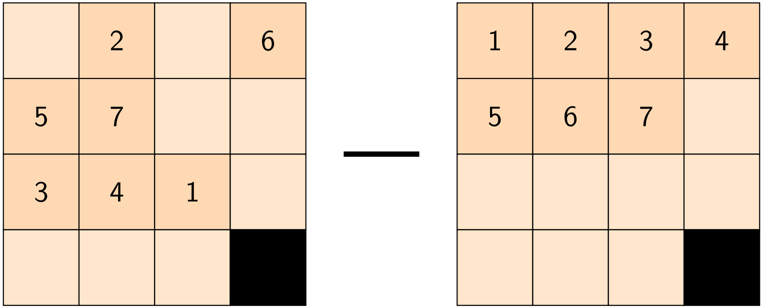
9	2	12	6
5	7	14	13
3	4	1	11
15	10	8	

—

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

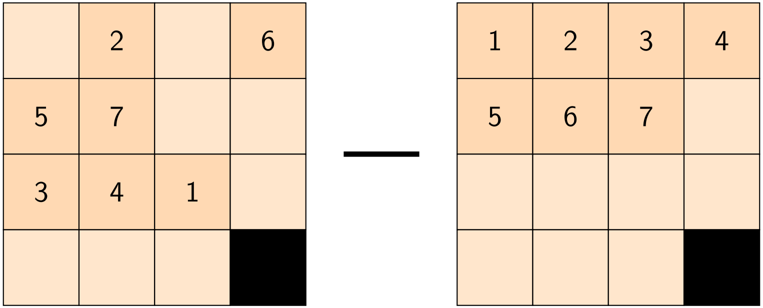
Problem II: Move tiles to transform left state into right state.

Heuristic Functions from Relaxed Problems: Example 2



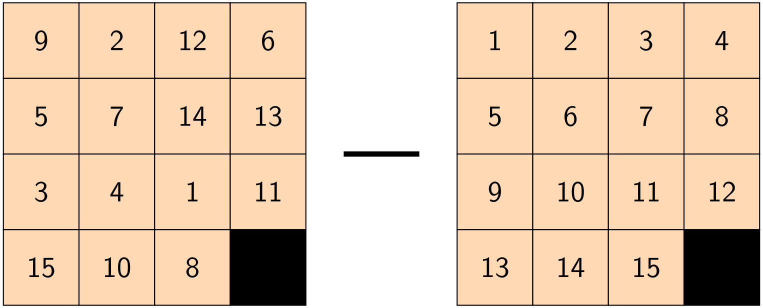
Relaxed Problem Π': Don't distinguish tiles 8–15.

Heuristic Functions from Relaxed Problems: Example 2



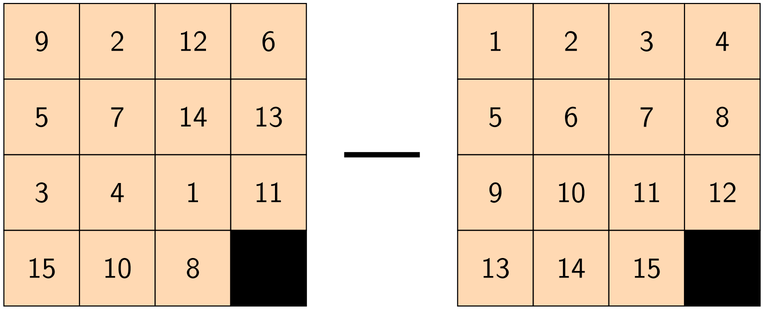
Heuristic function h: Length of solution to reduced puzzle.

Heuristic Functions from Relaxed Problems: Example 3



- 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.

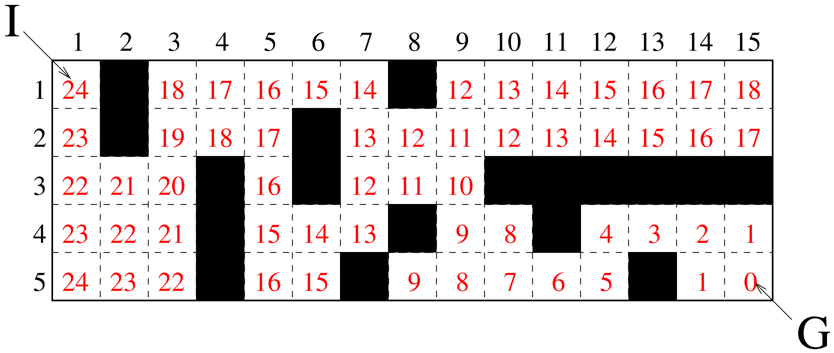
Heuristic Functions from Relaxed Problems: Example 4



- 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

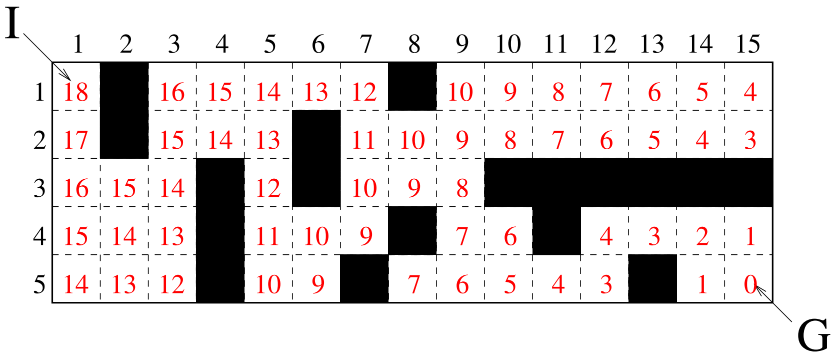
h^*:



	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

Heuristic Function Pitfalls: Example Path Planning

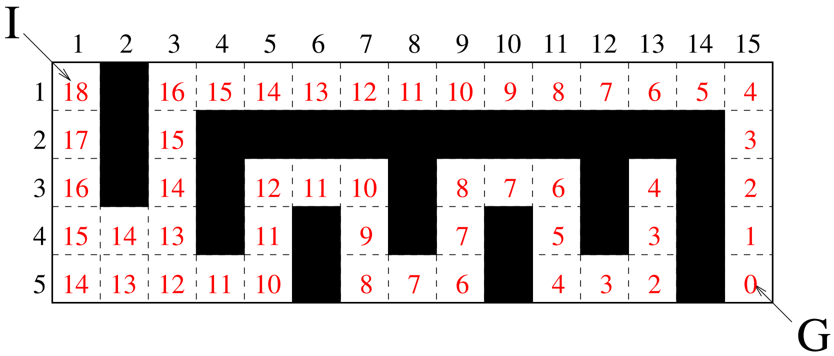
Manhattan Distance, “accurate h”:



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	18	⬛	16	15	14	13	12	⬛	10	9	8	7	6	5	4
2	17	⬛	15	14	13	⬛	11	10	9	8	7	6	5	4	3
3	16	15	14	⬛	12	⬛	10	9	8	⬛	⬛	⬛	⬛	⬛	⬛
4	15	14	13	⬛	11	10	9	⬛	7	6	⬛	4	3	2	1
5	14	13	12	⬛	10	9	⬛	7	6	5	4	3	⬛	1	0

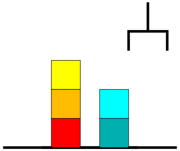
Heuristic Function Pitfalls: Example Path Planning

Manhattan Distance, “inaccurate h”:



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	18	■	16	15	14	13	12	11	10	9	8	7	6	5	4
2	17	■	15	■	■	■	■	■	■	■	■	■	■	■	3
3	16	■	14	■	12	11	10	■	8	7	6	■	4	■	2
4	15	14	13	■	11	■	9	■	7	■	5	■	3	■	1
5	14	13	12	11	10	■	8	7	6	■	4	3	2	■	0

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 $b_n, \dots, b_2$ and the goal is on(b_1, b_2). Then $h(s) = 1$ but $h^*(s) = 2n$ (pick/put-down for each $b_n, \dots, b_2$; 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.
  $\rightarrow$An admissible heuristic never overestimates the cost to the goal.
- 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 $\Longrightarrow$ Admissibility). *Let* Π *be a problem, and let* h *be a heuristic function for* Π*. If* h *is consistent, then* h *is admissible.*

Properties of Heuristic Functions: Examples

Admissibility and consistency:

- Is straight line distance admissible/consistent? Yes. Consistency: If you drive 100km to Moscow, 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.

→ In practice, admissible heuristics are typically consistent.

Inadmissible heuristics:

- Inadmissible heuristics typically arise as approximations of admissible heuristics that are too costly to compute.

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 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.)

Best-First Search

Informed search uses problem-specific knowledge.

- Use an evaluation function $f(n)$ for each node n.
  $\rightarrow$ Estimate of "desirability" of expanding node n.
- Expand **most desirable** unexpanded node.

Implementation: frontier is a queue sorted in decreasing order of desirability.

Two special cases:

- Greedy Best-First Search
- A* Search

Greedy Best-First Search: Ideas

Strategy: Expand the node that is **closest** to the goal.

Evaluation function: $f(n) = h(n)$ (heuristic function).
$\rightarrow$ Estimate of cost from n to the closest goal.

E.g., $h_{SLD}(n) =$ straight-line distance from n to Bucharest.

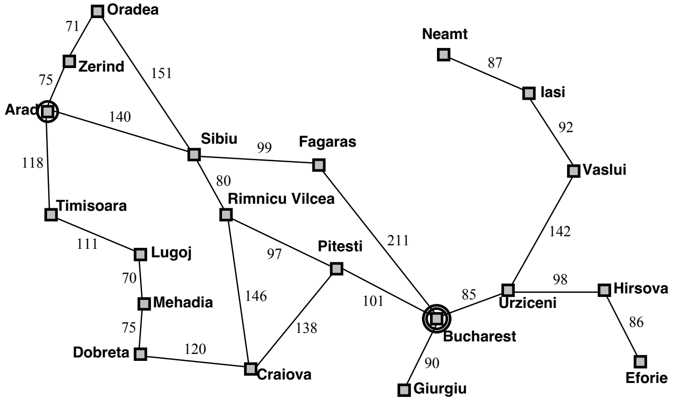
Greedy search expands the node that **appears** to be closest to goal.

Greedy Best-First Search: Algorithm

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

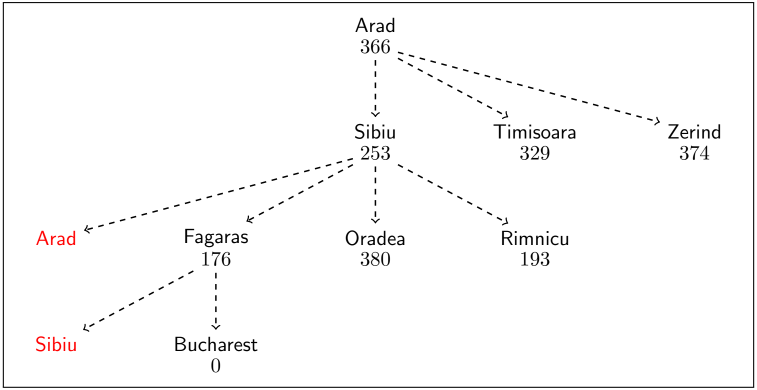
Greedy Best-First Search: Route to Bucharest



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*: Ideas

Main idea: avoid expanding paths that are already expensive.

Evaluation function: $f(n) = g(n) + h(n)$ where

- $g(n)$ = cost so far to reach n
- $h(n)$ = estimated cost to goal from n
- $f(n)$ = estimated total cost of path through n to goal

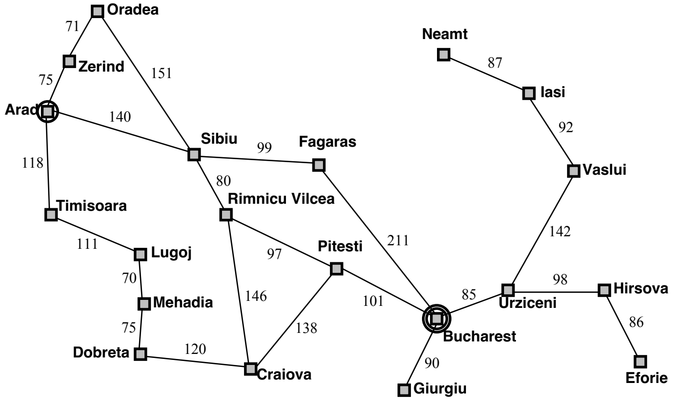
Frontier ordered by ascending $g + h$.

A^*

```
function $\mathrm{A}^*$(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 $\notin$ 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.

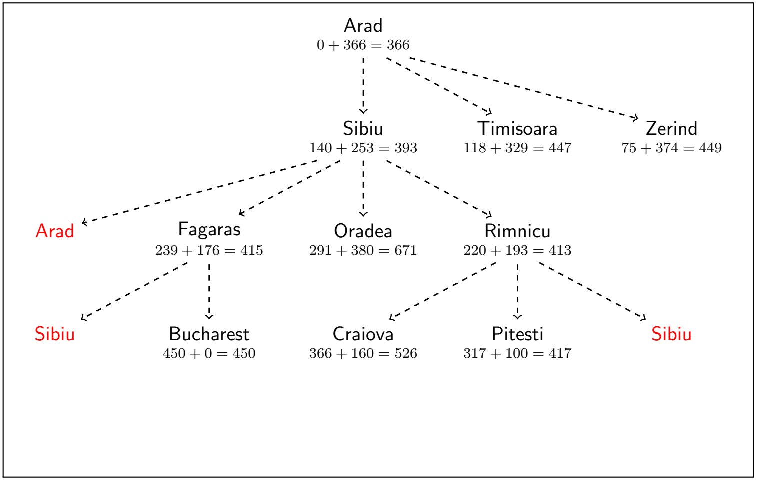
A*: Route to Bucharest



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

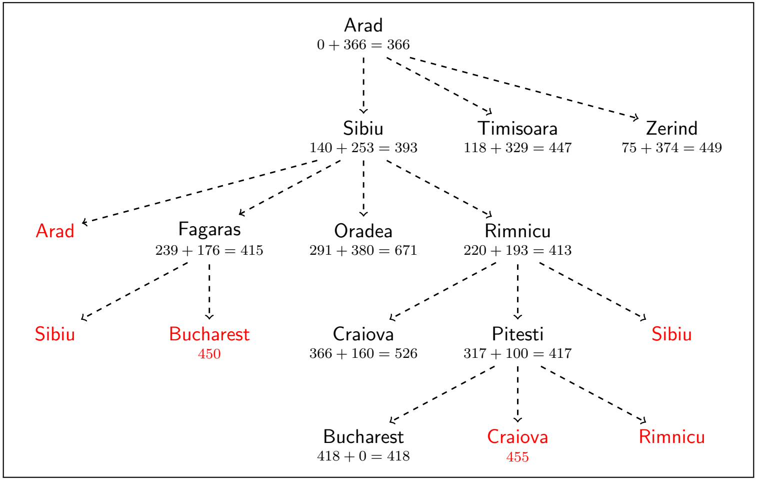
A^*: Route to Bucharest

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

$\rightarrow$ $h = 0$ implies no node ordering at all. 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).

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.

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 some 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)]!

$\rightarrow$ Systematically analyzing the practical behavior of heuristic search remains one of the biggest research challenges.

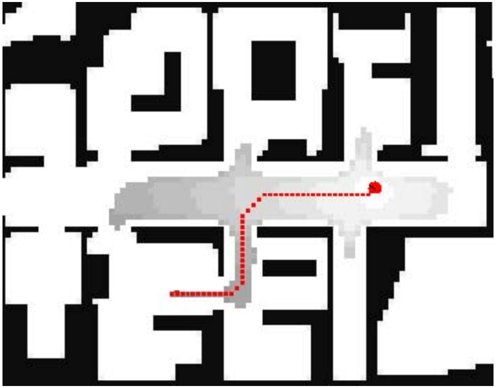
$\rightarrow$ There is little hope to prove practical sub-exponential search bounds.

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	6,384	39	25
10	47,127	93	39
12	3,644,035	227	73
14	-	539	113
16	-	1,301	211
18	-	3,056	363
20	-	7,276	676
22	-	18,094	1,219
24	-	39,135	1,641

Empirical Performance: A^* in Path Planning



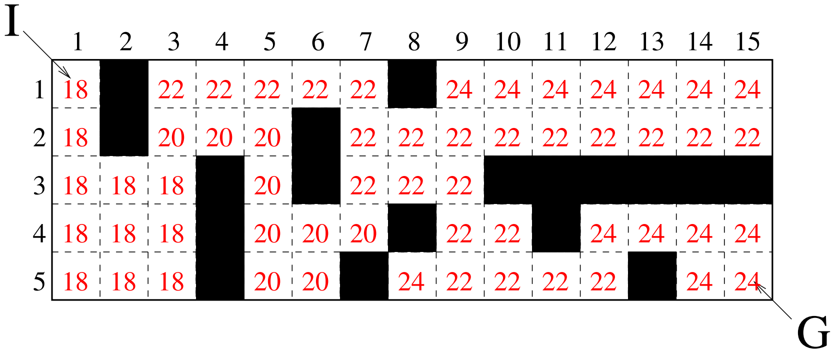
→ Difference to breadth-first search? That would explore all grid cells in a *circle* around the initial state!

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":



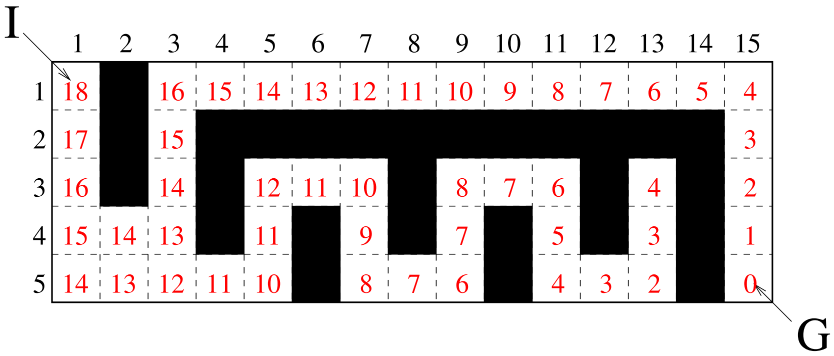
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	18	■	22	22	22	22	22	■	24	24	24	24	24	24	24
2	18	■	20	20	20	■	22	22	22	22	22	22	22	22	22
3	18	18	18	■	20	■	22	22	22	■	■	■	■	■	■
4	18	18	18	■	20	20	20	■	22	22	■	24	24	24	24
5	18	18	18	■	20	20	■	24	22	22	22	22	■	24	24

$\rightarrow$ 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

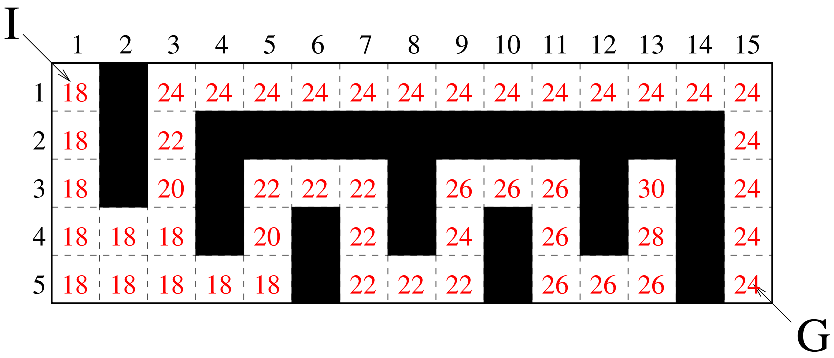
Greedy best-first search, "inaccurate h":



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

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

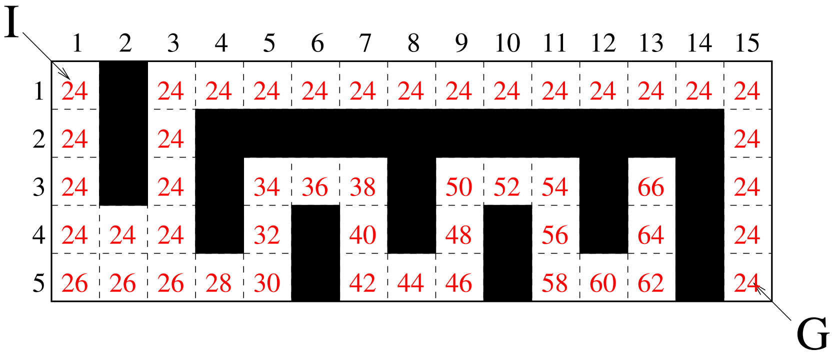
$\mathbf{A}^*(g+h)$, "inaccurate h":



$\rightarrow$ 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

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



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

Questionnaire

Question!

1. Is $\mathbf{A}^*$ 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".

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.)

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 duplicate 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.

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.