Artificial Intelligence

Chapter 4, Sections 1-2

Stuart RUSSEL

reorganized by L. Aszalós

April 27, 2016

Outline

- Best-first search
- A^* search
- Heuristics

Review: Tree search

```
function Tree-Search(problem, strategy): a solution or failure
  initialize the search tree with --the initial state of problem
  loop do
    if there are no candidates for expansion
      then return failure
    choose a leaf node for expansion according to strategy
    if the node contains a goal state
      then return the corresponding solution
    else expand the node and add the resulting nodes
                                  to the search tree
    end
```

A strategy is defined by picking the *order of node expansion*

Best-first search

- Idea:

Best-first search

- Idea:
  - use an **evaluation function** for each node

Best-first search

- Idea:
  - use an **evaluation function** for each node
  - estimate of "desirability"

Best-first search

- Idea:
  - use an **evaluation function** for each node
  - estimate of "desirability"
  - Expand most desirable unexpanded node

Best-first search

- Idea:
  - use an **evaluation function** for each node
  - estimate of “desirability”
  - Expand most desirable unexpanded node
- Implementation:

Best-first search

- Idea:
  - use an **evaluation function** for each node
  - estimate of "desirability"
  - Expand most desirable unexpanded node
- Implementation:
  - *fringe* is a queue sorted in decreasing order of desirability

Best-first search

- Idea:
  - use an **evaluation function** for each node
  - estimate of "desirability"
  - Expand most desirable unexpanded node
- Implementation:
  - *fringe* is a queue sorted in decreasing order of desirability
- Special cases:

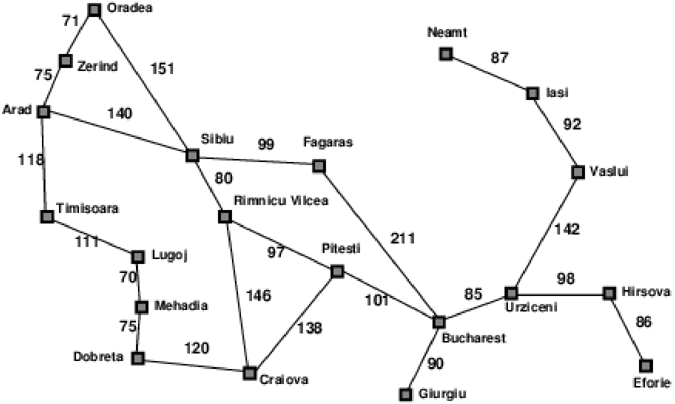
Best-first search

- Idea:
  - use an **evaluation function** for each node
  - estimate of "desirability"
  - Expand most desirable unexpanded node
- Implementation:
  - *fringe* is a queue sorted in decreasing order of desirability
- Special cases:
  - greedy search

Best-first search

- Idea:
  - use an **evaluation function** for each node
  - estimate of "desirability"
  - Expand most desirable unexpanded node
- Implementation:
  - *fringe* is a queue sorted in decreasing order of desirability
- Special cases:
  - greedy search
  - A^* search

Romania with step costs in km



Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy search

- Evaluation function $h(n)$

Greedy search

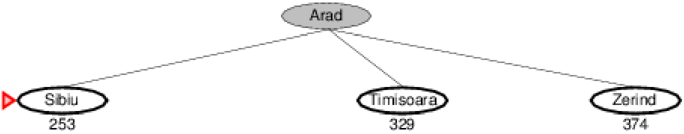
- Evaluation function $h(n)$
  - Heuristic

Greedy search

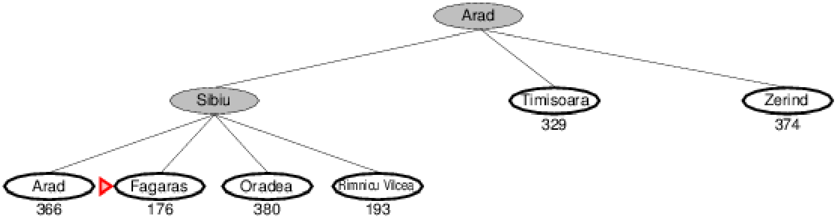
- Evaluation function $h(n)$
  - Heuristic
  - estimate of cost from n to the closest goal

Greedy search example

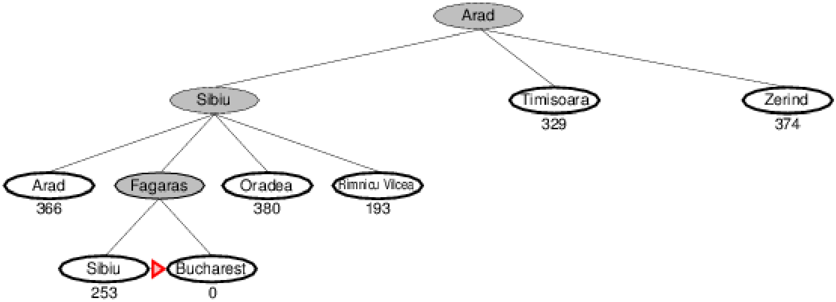
Greedy search example



Greedy search example



Greedy search example



Properties of greedy search

- Complete

Properties of greedy search

- Complete
  - No–can get stuck in loops,

Properties of greedy search

- Complete
  - No–can get stuck in loops,
    - e.g., with Oradea as goal, Iasi $\rightarrow$ Neamt $\rightarrow$ Iasi $\rightarrow$ Neamt $\rightarrow$

Properties of greedy search

- Complete
  - No–can get stuck in loops,
    - e.g., with Oradea as goal, Iasi $\rightarrow$ Neamt $\rightarrow$ Iasi $\rightarrow$ Neamt $\rightarrow$
  - Complete in finite space with repeated-state checking

Properties of greedy search

- Complete
  - No–can get stuck in loops,
    - e.g., with Oradea as goal, Iasi $\rightarrow$ Neamt $\rightarrow$ Iasi $\rightarrow$ Neamt $\rightarrow$
  - Complete in finite space with repeated-state checking
- Time

Properties of greedy search

- Complete
  - No–can get stuck in loops,
    - e.g., with Oradea as goal, Iasi $\rightarrow$ Neamt $\rightarrow$ Iasi $\rightarrow$ Neamt $\rightarrow$
  - Complete in finite space with repeated-state checking
- Time
  - $O(b^m)$, but a good heuristic can give dramatic improvement

Properties of greedy search

- Complete
  - No–can get stuck in loops,
    - e.g., with Oradea as goal, Iasi $\rightarrow$ Neamt $\rightarrow$ Iasi $\rightarrow$ Neamt $\rightarrow$
  - Complete in finite space with repeated-state checking
- Time
  - $O(b^m)$, but a good heuristic can give dramatic improvement
- Space

Properties of greedy search

- Complete
  - No–can get stuck in loops,
    - e.g., with Oradea as goal, Iasi $\rightarrow$ Neamt $\rightarrow$ Iasi $\rightarrow$ Neamt $\rightarrow$
  - Complete in finite space with repeated-state checking
- Time
  - $O(b^m)$, but a good heuristic can give dramatic improvement
- Space
  - $O(b^m)$—keeps all nodes in memory

Properties of greedy search

- Complete
  - No–can get stuck in loops,
    - e.g., with Oradea as goal, Iasi $\rightarrow$ Neamt $\rightarrow$ Iasi $\rightarrow$ Neamt $\rightarrow$
  - Complete in finite space with repeated-state checking
- Time
  - $O(b^m)$, but a good heuristic can give dramatic improvement
- Space
  - $O(b^m)$—keeps all nodes in memory
- Optimal

Properties of greedy search

- Complete
  - No–can get stuck in loops,
    - e.g., with Oradea as goal, Iasi $\rightarrow$ Neamt $\rightarrow$ Iasi $\rightarrow$ Neamt $\rightarrow$
  - Complete in finite space with repeated-state checking
- Time
  - $O(b^m)$, but a good heuristic can give dramatic improvement
- Space
  - $O(b^m)$—keeps all nodes in memory
- Optimal
  - No

A^* search

- Idea:

A^* search

- Idea:
  - avoid expanding paths that are already expensive

A^* search

- Idea:
  - avoid expanding paths that are already expensive
- Evaluation function $f(n) = g(n) + h(n)$

A^* search

- Idea:
  - avoid expanding paths that are already expensive
- Evaluation function $f(n) = g(n) + h(n)$
  - $g(n)$ = cost so far to reach n

A^* search

- Idea:
  - avoid expanding paths that are already expensive
- Evaluation function $f(n) = g(n) + h(n)$
  - $g(n) =$ cost so far to reach n
  - $h(n) =$ estimated cost to goal from n

A^* search

- Idea:
  - avoid expanding paths that are already expensive
- Evaluation function $f(n) = g(n) + h(n)$
  - $g(n) =$ 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

A^* search

- Idea:
  - avoid expanding paths that are already expensive
- Evaluation function $f(n) = g(n) + h(n)$
  - $g(n) =$ 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
- A^* search uses an **admissible** heuristic

A^* search

- Idea:
  - avoid expanding paths that are already expensive
- Evaluation function $f(n) = g(n) + h(n)$
  - $g(n)$ = 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
- A^* search uses an **admissible** heuristic
  - i.e., $h(n) \leq h^*(n)$ where $h^*(n)$ is the *true* cost from n.

A^* search

- Idea:
  - avoid expanding paths that are already expensive
- Evaluation function $f(n) = g(n) + h(n)$
  - $g(n) =$ 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
- A^* search uses an **admissible** heuristic
  - i.e., $h(n) \leq h^*(n)$ where $h^*(n)$ is the *true* cost from n.
  - Also require $h(n) \geq 0$, so $h(G) = 0$ for any goal G.

A^* search

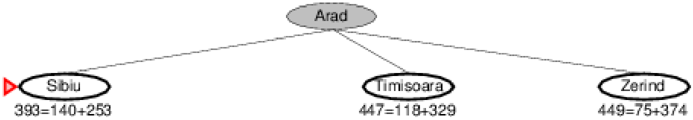
- Idea:
  - avoid expanding paths that are already expensive
- Evaluation function $f(n) = g(n) + h(n)$
  - $g(n) =$ cost so far to reach n
  - $h(n) =$ estimated cost to goal from n
  - $f(n) =$ estimated total cost of path through n to goal
- A^* search uses an **admissible** heuristic
  - i.e., $h(n) \leq h^*(n)$ where $h^*(n)$ is the *true* cost from n.
  - Also require $h(n) \geq 0$, so $h(G) = 0$ for any goal G.
  - E.g., $h_{\mathrm{SLD}}(n)$ never overestimates the actual road distance

A^* search

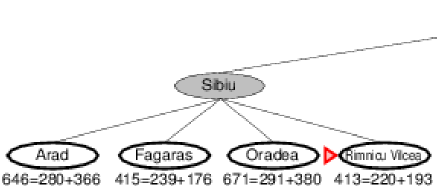
- Idea:
  - avoid expanding paths that are already expensive
- Evaluation function $f(n) = g(n) + h(n)$
  - $g(n) =$ cost so far to reach n
  - $h(n) =$ estimated cost to goal from n
  - $f(n) =$ estimated total cost of path through n to goal
- A^* search uses an **admissible** heuristic
  - i.e., $h(n) \leq h^*(n)$ where $h^*(n)$ is the *true* cost from n.
  - Also require $h(n) \geq 0$, so $h(G) = 0$ for any goal G.
  - E.g., $h_{\mathrm{SLD}}(n)$ never overestimates the actual road distance
- **Theorem**: A^* search is optimal

A^* search example

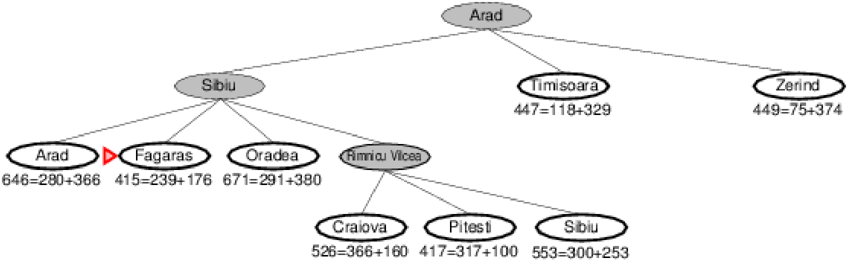
A^* search example



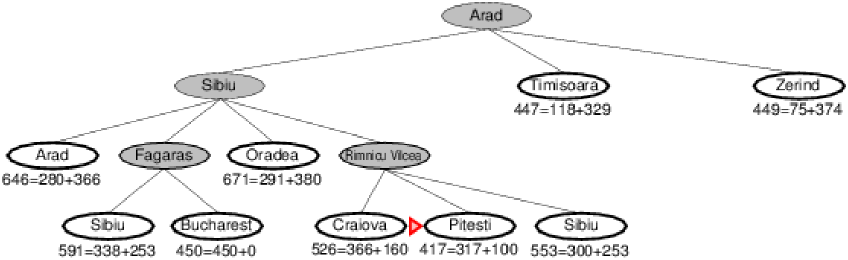
A^* search example



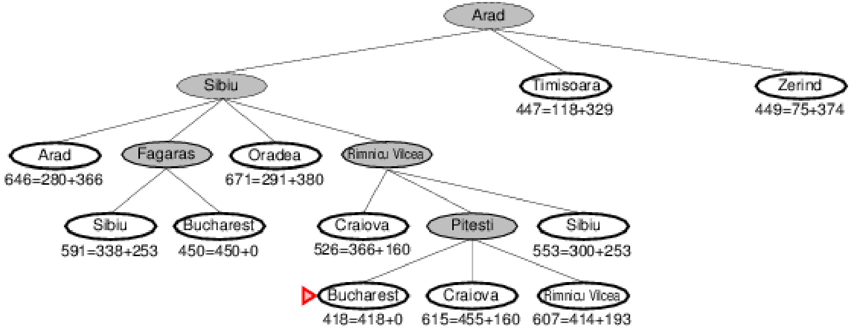
A^* search example



A^* search example

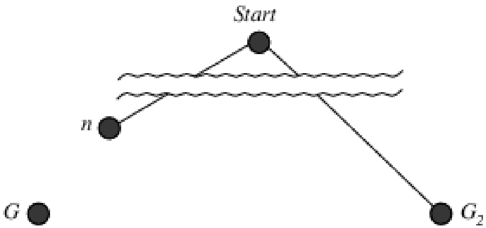


A^* search example



Optimality of A^* (standard proof)

Suppose some suboptimal goal G_2 has been generated and is in the queue. Let n be an unexpanded node on a shortest path to an optimal goal G.



$$\begin{aligned} f(G_2) &= g(G_2) && \text{since } h(G_2) = 0 \\ &> g(G) && \text{since } G_2 \text{ is suboptimal} \\ &\geq f(n) && \text{since } h \text{ is admissible} \end{aligned}$$

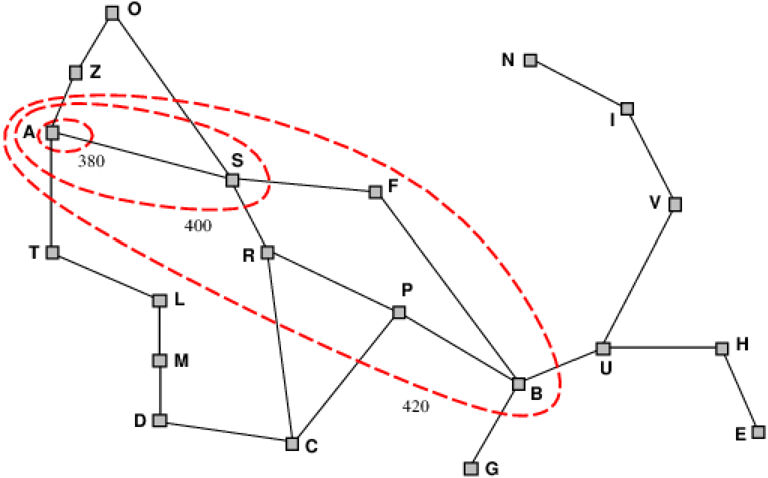
Since $f(G_2) > f(n)$, A^* will never select G_2 for expansion

Optimality of A^* (more useful)

Lemma: A^* expands nodes in order of increasing f value

Gradually adds "f-contours" of nodes (cf. breadth-first adds layers)

Contour i has all nodes with $f = f_i$, where $f_i < f_{i+1}$



Properties of A^*

- Complete

Properties of A^*

- Complete
  - Yes, unless there are infinitely many nodes with $f \leq f(G)$

Properties of A^*

- Complete
  - Yes, unless there are infinitely many nodes with $f \leq f(G)$
- Time

Properties of A^*

- Complete
  - Yes, unless there are infinitely many nodes with $f \leq f(G)$
- Time
  - Exponential in [relative error in $h \times$ length of soln.]

Properties of A^*

- Complete
  - Yes, unless there are infinitely many nodes with $f \leq f(G)$
- Time
  - Exponential in [relative error in $h \times$ length of soln.]
- Space

Properties of A^*

- Complete
  - Yes, unless there are infinitely many nodes with $f \leq f(G)$
- Time
  - Exponential in [relative error in h $\times$ length of soln.]
- Space
  - Keeps all nodes in memory

Properties of A^*

- Complete
  - Yes, unless there are infinitely many nodes with $f \leq f(G)$
- Time
  - Exponential in [relative error in h $\times$ length of soln.]
- Space
  - Keeps all nodes in memory
- Optimal

Properties of A^*

- Complete
  - Yes, unless there are infinitely many nodes with $f \leq f(G)$
- Time
  - Exponential in [relative error in $h \times$ length of soln.]
- Space
  - Keeps all nodes in memory
- Optimal
  - Yes—cannot expand f_{i+1} until f_i is finished

Properties of A^*

- Complete
  - Yes, unless there are infinitely many nodes with $f \leq f(G)$
- Time
  - Exponential in [relative error in $h \times$ length of soln.]
- Space
  - Keeps all nodes in memory
- Optimal
  - Yes—cannot expand f_{i+1} until f_i is finished
    - A^* expands all nodes with $f(n) < C^*$

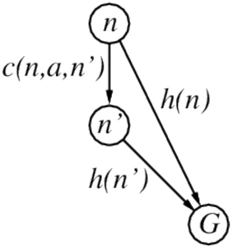
Properties of A^*

- Complete
  - Yes, unless there are infinitely many nodes with $f \leq f(G)$
- Time
  - Exponential in [relative error in $h \times$ length of soln.]
- Space
  - Keeps all nodes in memory
- Optimal
  - Yes—cannot expand f_{i+1} until f_i is finished
    - ★ A^* expands all nodes with $f(n) < C^*$
    - ★ A^* expands some nodes with $f(n) = C^*$

Properties of A^*

- Complete
  - Yes, unless there are infinitely many nodes with $f \leq f(G)$
- Time
  - Exponential in [relative error in $h \times$ length of soln.]
- Space
  - Keeps all nodes in memory
- Optimal
  - Yes—cannot expand f_{i+1} until f_i is finished
    - A^* expands all nodes with $f(n) < C^*$
    - A^* expands some nodes with $f(n) = C^*$
    - A^* expands no nodes with $f(n) > C^*$

Proof of lemma: Consistency



A heuristic is **consistent** if $h(n) \leq c(n, a, n') + h(n')$
If h is consistent, we have

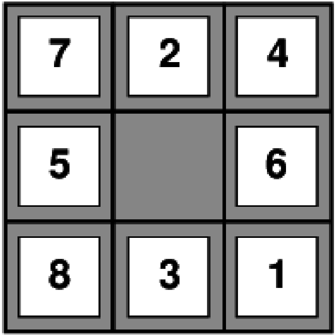
$$
\begin{aligned}
f(n') &= g(n') + h(n') \\
&= g(n) + c(n, a, n') + h(n') \\
&\geq g(n) + h(n) \\
&= f(n)
\end{aligned}
$$

I.e., $f(n)$ is nondecreasing along any path.

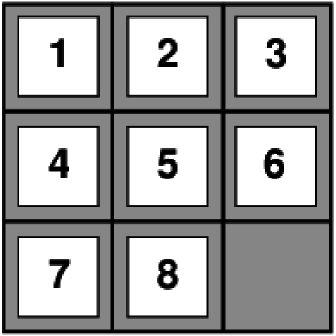
Admissible heuristics

E.g., for the 8-puzzle:

- $h_1(n)$ = number of misplaced tiles
- $h_2(n)$ = total **Manhattan** distance (i.e., no. of squares from desired location of each tile)



Start State



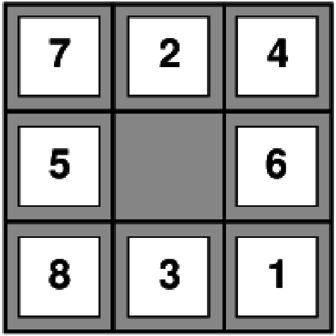
Goal State

$h_1(S) =$

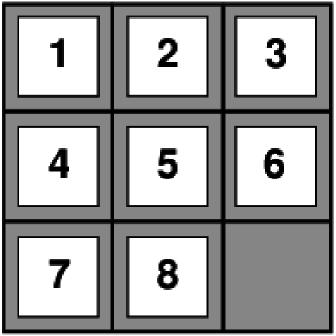
Admissible heuristics

E.g., for the 8-puzzle:

- $h_1(n)$ = number of misplaced tiles
- $h_2(n)$ = total **Manhattan** distance (i.e., no. of squares from desired location of each tile)



Start State



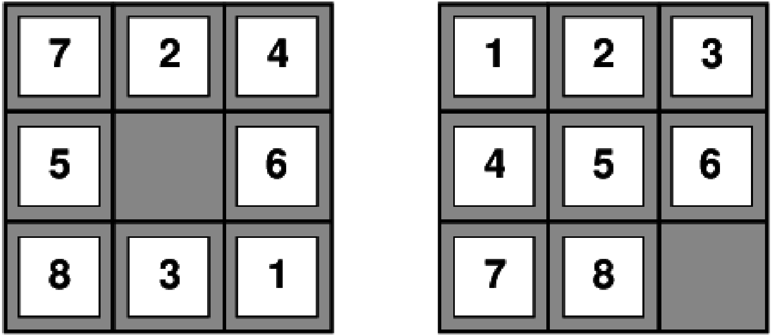
Goal State

$h_1(S) = 6$

Admissible heuristics

E.g., for the 8-puzzle:

- $h_1(n)$ = number of misplaced tiles
- $h_2(n)$ = total **Manhattan** distance (i.e., no. of squares from desired location of each tile)



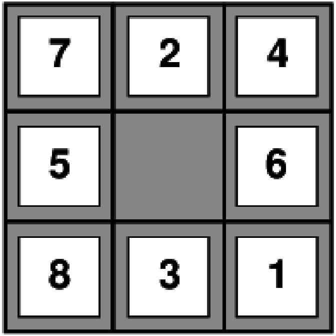
Start State Goal State

$h_1(S) = 6$ $h_2(S) =$

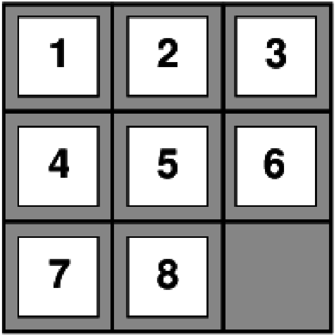
Admissible heuristics

E.g., for the 8-puzzle:

- $h_1(n)$ = number of misplaced tiles
- $h_2(n)$ = total **Manhattan** distance (i.e., no. of squares from desired location of each tile)



Start State



Goal State

$h_1(S) = 6$ $h_2(S) = 4+0+3+3+1+0+2+1 = 14$

Dominance

If $h_2(n) \geq h_1(n)$ for all n (both admissible) then h_2 **dominates** h_1 and is better for search

Typical search costs:

$d = 14$ IDS $= 3{,}473{,}941$ nodes
$A^*(h_1) = 539$ nodes
$A^*(h_2) = 113$ nodes

$d = 24$ IDS $\approx 54{,}000{,}000{,}000$ nodes
$A^*(h_1) = 39{,}135$ nodes
$A^*(h_2) = 1{,}641$ nodes

Given any admissible heuristics h_a, h_b,

$$h(n) = \max(h_a(n), h_b(n))$$

is also admissible and dominates h_a, h_b

Relaxed problems

- Admissible heuristics can be derived from the *exact* solution cost of a *relaxed* version of the problem

Relaxed problems

- Admissible heuristics can be derived from the *exact* solution cost of a *relaxed* version of the problem
- If the rules of the 8-puzzle are relaxed so that a tile can move *anywhere*, then $h_1(n)$ gives the shortest solution

Relaxed problems

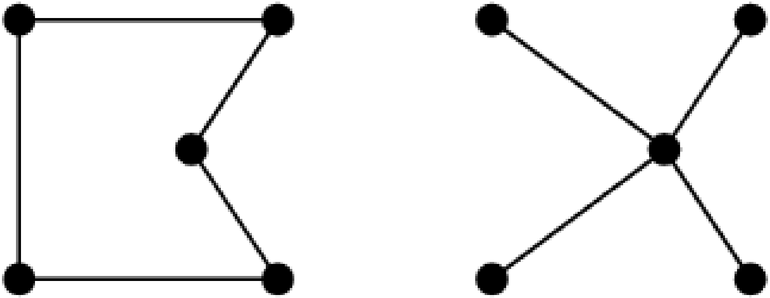
- Admissible heuristics can be derived from the *exact* solution cost of a *relaxed* version of the problem
- If the rules of the 8-puzzle are relaxed so that a tile can move *anywhere*, then $h_1(n)$ gives the shortest solution
- If the rules are relaxed so that a tile can move to *any adjacent square*, then $h_2(n)$ gives the shortest solution

Relaxed problems

- Admissible heuristics can be derived from the *exact* solution cost of a *relaxed* version of the problem
- If the rules of the 8-puzzle are relaxed so that a tile can move *anywhere*, then $h_1(n)$ gives the shortest solution
- If the rules are relaxed so that a tile can move to *any adjacent square*, then $h_2(n)$ gives the shortest solution
- Key point: the optimal solution cost of a relaxed problem is no greater than the optimal solution cost of the real problem

Relaxed problems contd.

- Well-known example: **travelling salesperson problem** (TSP)
  - Find the shortest tour visiting all cities exactly once



- **Minimum spanning tree** can be computed in $O(n^2)$ and is a lower bound on the shortest (open) tour

Summary

- Heuristic functions estimate costs of shortest paths
- Good heuristics can dramatically reduce search cost
- Greedy best-first search expands lowest $h\}$
  - incomplete and not always optimal
- A^* search expands lowest $g + h$
  - complete and optimal
  - also optimally efficient (up to tie-breaks, for forward search
- Admissible heuristics can be derived from exact solution of relaxed problems