Lecture 12: Search (III)

EMAT31530/March 2018/Raul Santos-Rodriguez

Have a look at ...

... Russell and Norvig (Ch. 3)

Summary

Objective: find the minimum cost path from s_{start} to an s satisfying $Goal(s)$ =TRUE.

Tree search: Backtracking, BFS, DFS, DFS-ID

State: summary of past actions sufficient to choose future actions optimally

Graph search: Dynamic programming, UCS

Informed vs uninformed: Best first, greedy search, A^*

Outline

Search or how to find a sequence of actions that achieves a goal when no single action will do. Today we will cover

- A^*
- Heuristics
- Relaxation

Informed search

Goal make UCS faster

Problem UCS orders states by cost from s_{start} to s

Idea take into account cost from s to s_{goal}

Best-first node n is selected for expansion based on an evaluation function $f(n)$

Informed search

Goal make UCS faster

Problem UCS orders states by cost from s_{start} to s

Idea take into account cost from s to s_{goal}

Best-first node n is selected for expansion based on an evaluation function $f(n)$

Now our goal is to make UCS faster. If we look at the UCS algorithm, we see that it explores states based on how far they are away from the start state. As a result, it will explore many states which are close to the start state, but actually moving farther away from the goal. Intuitively, we'd like to bias UCS towards exploring states which are closer to the goal: best-first search. Best-first search is an algorithm in which a node n is selected for expansion based on an evaluation function, $f(n)$. The evaluation function is construed as a cost estimate, so the node with the lowest evaluation is expanded first. The implementation of best-first graph search is identical to that for uniform-cost search, except for the use of f instead of g to order the priority queue. The choice of f determines the search strategy.

Heuristics

Definition

A heuristic $h(n)$ is the estimated cost of the cheapest path from the state at node n to a goal state.

- Unlike $g(n)$, it depends only on the state at that node
- Encode additional knowledge of the problem
- Arbitrary and nonnegative
- Constraint: if n is a goal node, then $h(n) = 0$.

Heuristics

Definition

A heuristic $h(n)$ is the estimated cost of the cheapest path from the state at node n to a goal state.

- Unlike $g(n)$, it depends only on the state at that node
- Encode additional knowledge of the problem
- Arbitrary and nonnegative
- Constraint: if n is a goal node, then $h(n) = 0$.

Most best-first algorithms include as a component of f a heuristic function, denoted $h(n)$: $h(n)$ is the estimated cost of the cheapest path from the state at node n to a goal state. (Notice that $h(n)$ takes a node as input, but, unlike $g(n)$, it depends only on the state at that node). Heuristic functions are the most common form in which additional knowledge of the problem is imparted to the search algorithm. We consider them to be arbitrary, nonnegative, problem-specific functions, with one constraint: if n is a goal node, then $h(n) = 0$. $h(n)$ may not be 100% accurate, but it should give better results than pure guesswork. Also, a heuristic should reduce the number of nodes that need to be examined.

Greedy best-first search

A best-first search that uses h to select the next node to expand: greedy search.

Greedy search

$$f(n) = h(n)$$

Is greedy search

- complete? NO (be careful with loops)
- optimal? NO

Greedy best-first search

A best-first search that uses h to select the next node to expand: greedy search.

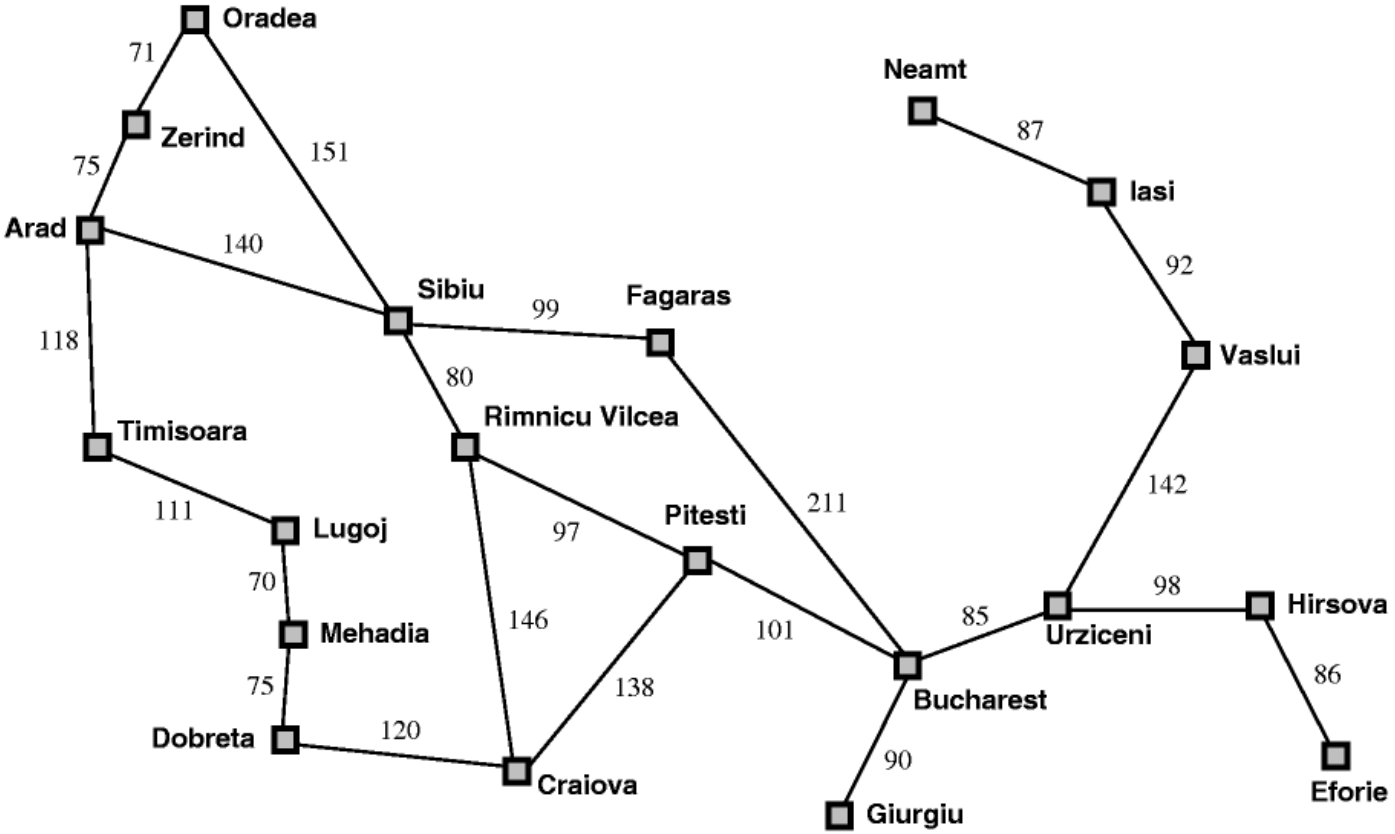
Greedy search

$$f(n) = h(n)$$

Is greedy search
- complete? NO (be careful with loops)
- optimal? NO

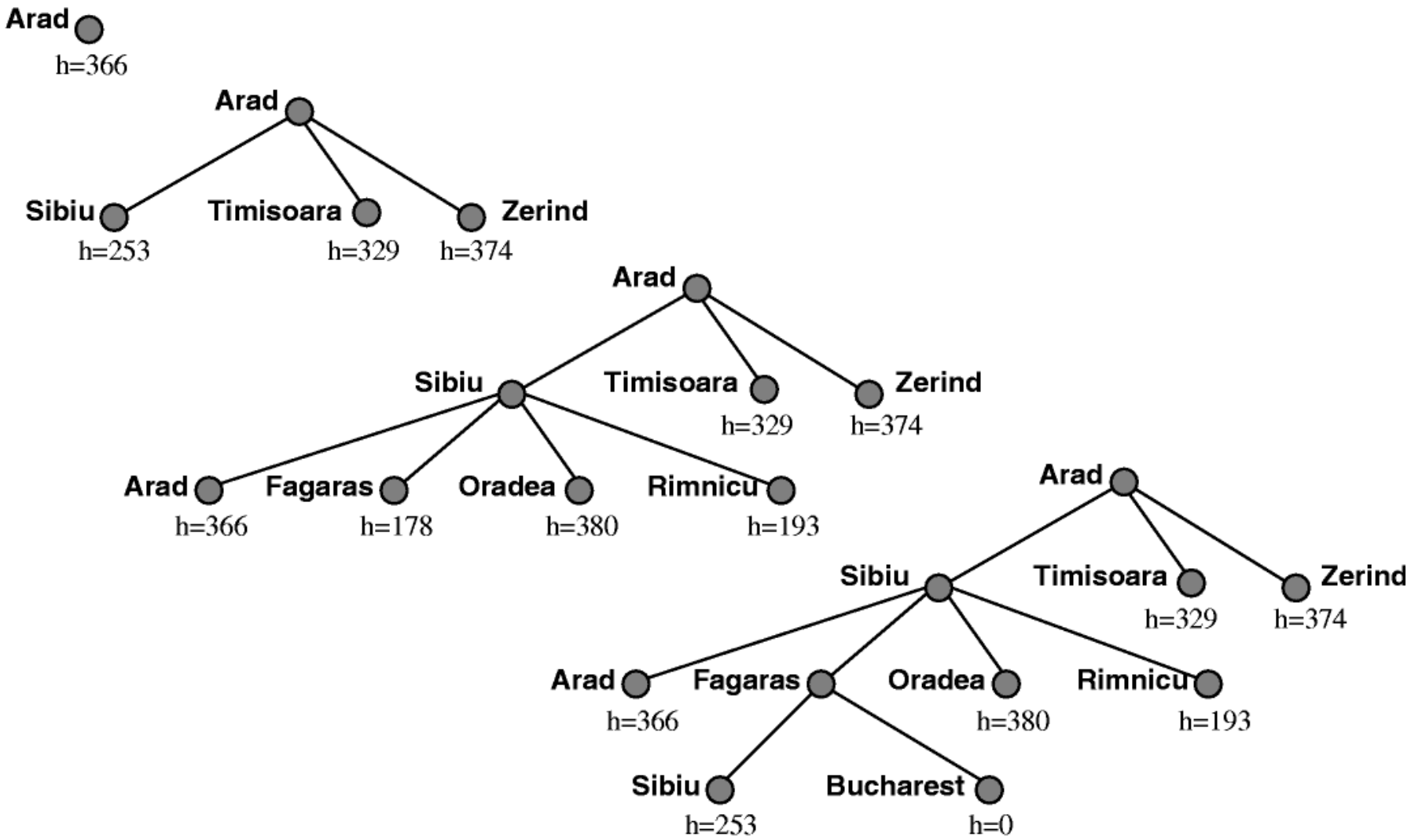
Greedy best-first search tries to expand the node that is closest to the goal, on the grounds that this is likely to lead to a solution quickly. Thus, it evaluates nodes by using just the heuristic function; that is, $f(n) = h(n)$. Greedy search picks the most likely node (based on some heuristic value) from the partially expanded tree at each stage. It tends to find a shorter path than depth-first or breadth-first search, but does not guarantee to find the best path. However, it can easily be fooled into taking poor paths.

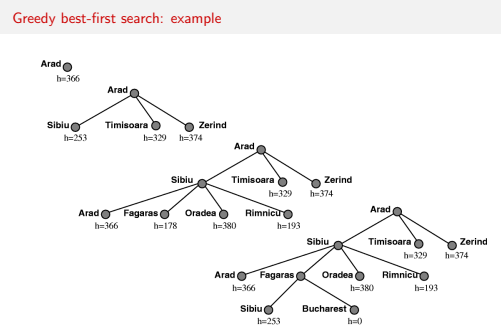
Greedy best-first search: example



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 best-first search: example





For this problem, greedy search leads to a minimal cost search because no node off the solution path is expanded. However, it does not find the optimal path: the path it found via Sibiu and Fagaras to Bucharest is 32 miles longer than the path through Pimnicu Vilcea and Pitesti. Hence, the algorithm always chooses what looks locally best, rather than worrying about whether or not it will be best in the long run. (This is why its called greedy search.) Greedy search is susceptible to false starts. Consider the problem of getting from Iasi to Fagaras. *h* suggests that Neamt be expanded first, but it is a deadend. The solution is to go first to Vaslui and then continue to Urziceni, Bucharest and Fagaras. Note that if we are not careful to detect repeated states, the solution will never be found - the search will oscillate between Neamt and Iasi. Greedy search resembles DFS in the way that it prefers to follow a single path to the goal and backup only when a deadend is encountered. It suffers from the same defects as DFS - it is not optimal and it is incomplete because it can start down an infinite path and never try other possibilities.

Best of both: UCS + Greedy

Idea

A^* takes into account the cost from the root node to the current node and estimates the path cost from the current node to the goal node

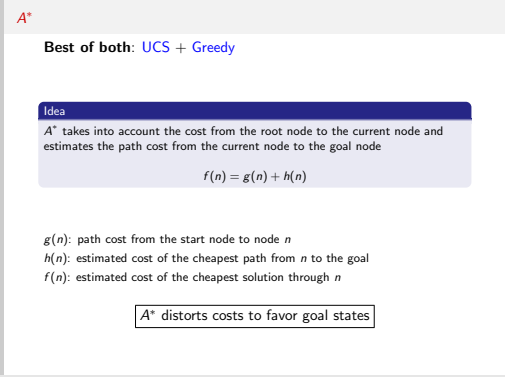
$$f(n) = g(n) + h(n)$$

$g(n)$: path cost from the start node to node n

$h(n)$: estimated cost of the cheapest path from n to the goal

$f(n)$: estimated cost of the cheapest solution through n

A^* distorts costs to favor goal states

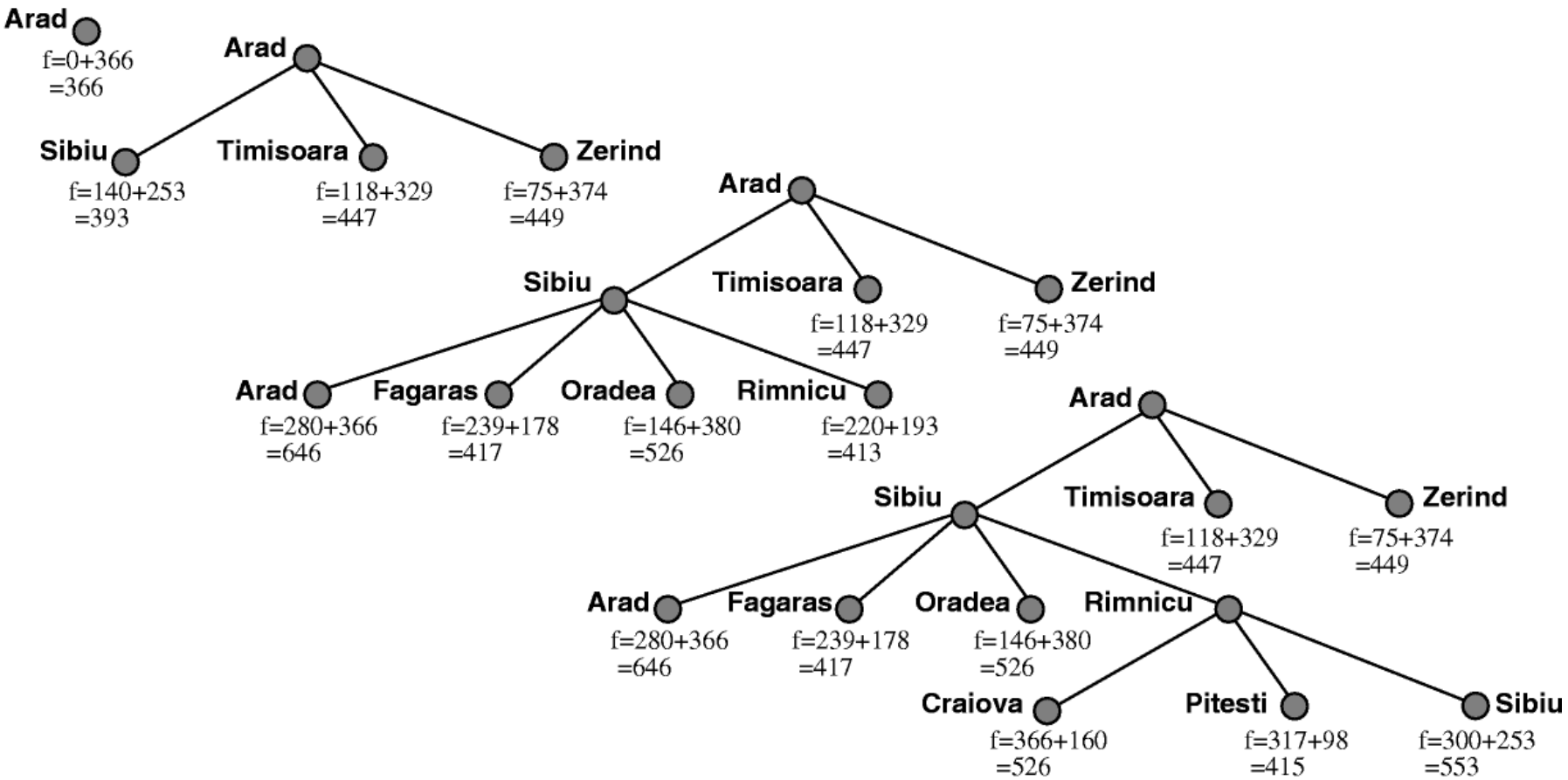


Algorithm A^* is a best-first search algorithm that relies on an open list and a closed list to find a path that is both optimal and complete towards the goal. It works by combining the benefits of the uniform-cost search and greedy search algorithms. A^* makes use of both elements by including two separate path finding functions in its algorithm that take into account the cost from the root node to the current node and estimates the path cost from the current node to the goal node.
The first function is $g(n)$, which calculates the path cost between the start node and the current node. The second function is $h(n)$, which is a heuristic to calculate the estimated path cost from the current node to the goal node:

$$f(n) = g(n) + h(n)$$

It represents the path cost of the most efficient estimated path towards the goal. A^* continues to re-evaluate both $g(n)$ and $h(n)$ throughout the search for all of the nodes that it encounters in order to arrive at the minimal cost path to the goal. This algorithm is extremely popular for pathfinding in strategy computer games.

A^*: example


Arad
f=0+366
=366
Arad
Sibiu
f=140+253
=393
Timisoara
f=118+329
=447
Zerind
f=75+374
=449
Arad
Sibiu
Timisoara
f=118+329
=447
Zerind
f=75+374
=449
Arad
f=280+366
=646
Fagaras
f=239+178
=417
Oradea
f=146+380
=526
Rimnicu
f=220+193
=413
Arad
Sibiu
Timisoara
f=118+329
=447
Zerind
f=75+374
=449
Arad
f=280+366
=646
Fagaras
f=239+178
=417
Oradea
f=146+380
=526
Rimnicu
Craiova
f=366+160
=526
Pitesti
f=317+98
=415
Sibiu
f=300+253
=553

A^*: algorithm

Algorithm: A* search [Hart/Nilsson/Raphael, 1968]

Run uniform cost search with modified edge costs:

$$Cost'(s, a) = Cost(s, a) + h(Succ(s, a)) - h(s)$$

if $h(n)$ satisfies certain conditions, is A^* search

- complete? YES
- optimal? YES

The algorithm is identical to UCS, using $g + h$ instead of g

A^*: conditions for optimality

Will any heuristic work? NO

Admissibility

A heuristic $h(n)$ is said to be an admissible heuristic if it never overestimates the cost to reach the goal. For every node n,

$$h(n) \leq h^*(n),$$

where $h^*(n)$ is the true cost to reach the goal state from n.

If $h(n)$ is not admissible, the method is called A.

A^*: conditions for optimality

A^*: conditions for optimality

Will any heuristic work? NO

Admissibility

A heuristic $h(n)$ is said to be an admissible heuristic if it never overestimates the cost to reach the goal. For every node n,

$$h(n) \leq h^*(n),$$

where $h^*(n)$ is the true cost to reach the goal state from n.

If $h(n)$ is not admissible, the method is called A.

Admissible heuristics are optimistic because they think the cost of solving the problem is less than it actually is.

Question

Consider the heuristic from our example (straight line distance).

Is the heuristic admissible?

- Yes
- No

Question

Consider the heuristic from our example (straight line distance).

Is the heuristic admissible?

- Yes
- No

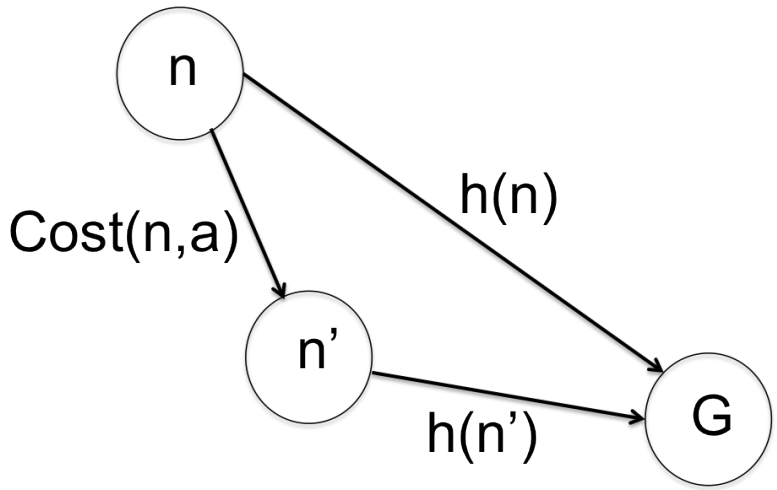
The straight-line distance is admissible because the shortest path between any two points is a straight line.

A^*: conditions for optimality

Consistency or monotonicity

A heuristic $h(n)$ is consistent if, for every node n and every successor n' of n generated by any action a, the estimated cost of reaching the goal from n is no greater than the step cost of getting to n' plus the estimated cost of reaching the goal from n':

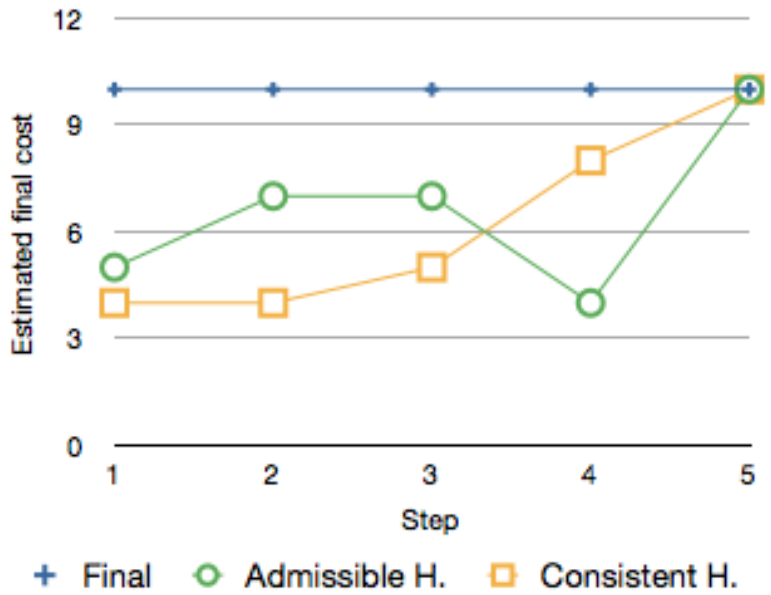
$$h(n) \leq Cost(n, a) + h(n')$$



A^*: conditions for optimality

Corollary

Every consistent heuristic is also admissible.



Comparison of an admissible but inconsistent and a consistent heuristic evaluation function.

Optimality of $A*$

1. If $h(n)$ is consistent, then the values of $f(n)$ along any path are nondecreasing.

Proof

Suppose n' is a successor of n; then $g(n') = g(n) + Cost(n, a)$ for some action a, and we have

$$f(n') = g(n') + h(n') = g(n) + Cost(n, a) + h(n') \geq g(n) + h(n) = f(n)$$

2. Whenever A^* selects a node n for expansion, the optimal path to that node has been found.

Proof

Were this not the case, there would have to be another frontier node n' on the optimal path from the start node to n, because f is nondecreasing along any path, n' would have lower f-cost than n and would have been selected first.

Optimality of $A*$

1. If $h(n)$ is consistent, then the values of $f(n)$ along any path are nondecreasing.

Proof

Suppose n' is a successor of n; then $g(n') = g(n) + Cost(n, a)$ for some action a, and we have

$$f(n') = g(n') + h(n') = g(n) + Cost(n, a) + h(n') \geq g(n) + h(n) = f(n)$$

2. Whenever A^* selects a node n for expansion, the optimal path to that node has been found.

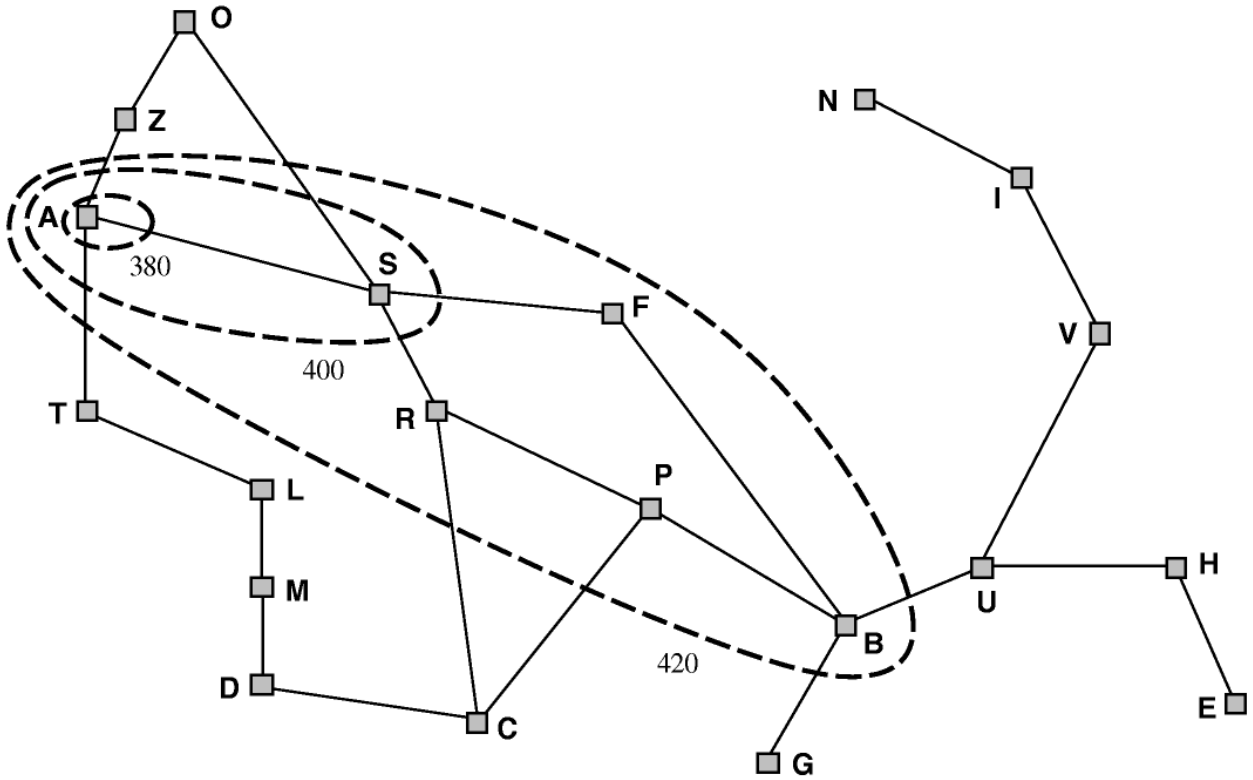
Proof

Were this not the case, there would have to be another frontier node n' on the optimal path from the start node to n, because f is nondecreasing along any path, n' would have lower f-cost than n and would have been selected first.

One final observation is that among optimal algorithms of this type - algorithms that extend search paths from the root and use the same heuristic information - A^* is optimally efficient for any given consistent heuristic. That is, no other optimal algorithm is guaranteed to expand fewer nodes than A^*.

A^*: contours

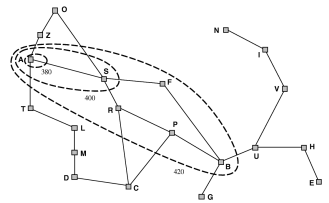
f-costs are nondecreasing along any path $\rightarrow$ contours in the state space



A^*: contours

A^*: contours

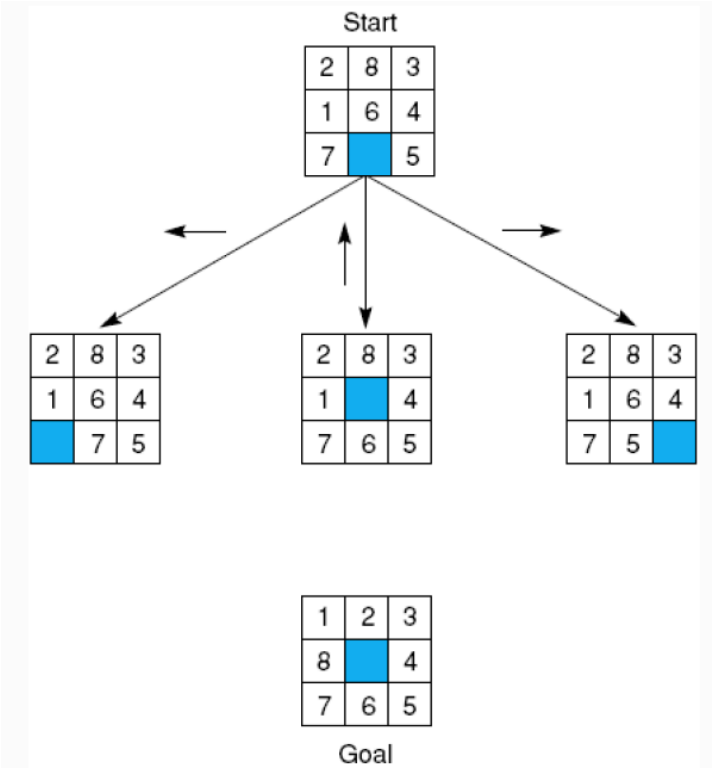
f-costs are nondecreasing along any path → contours in the state space



Inside the contour labeled 400, all nodes have $f(n)$ less than or equal to 400, and so on. Then, because A^* expands the frontier node of lowest f-cost, we can see that an A^* search fans out from the start node, adding nodes in concentric bands of increasing f-cost. With uniform-cost search (A^* search using $h(n) = 0$), the bands will be circular around the start state.

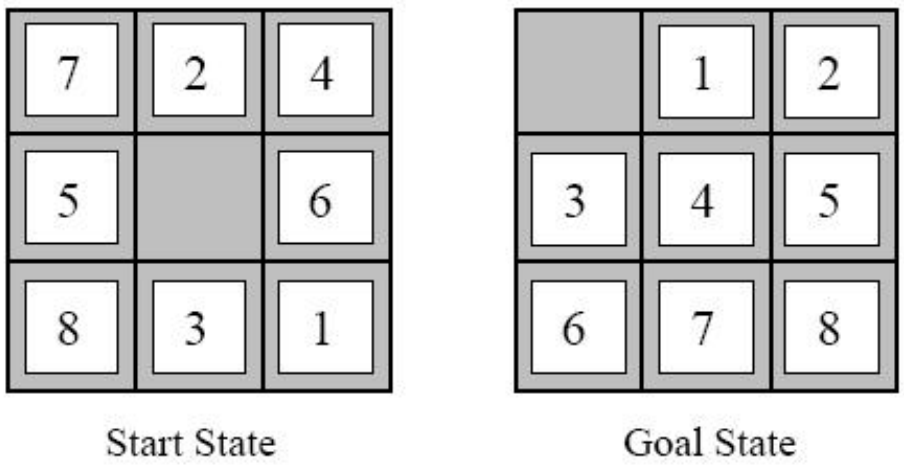
Example: 8-puzzle

Slide the tiles horizontally or vertically into the empty space until the configuration matches the goal configuration.



Example: 8-puzzle

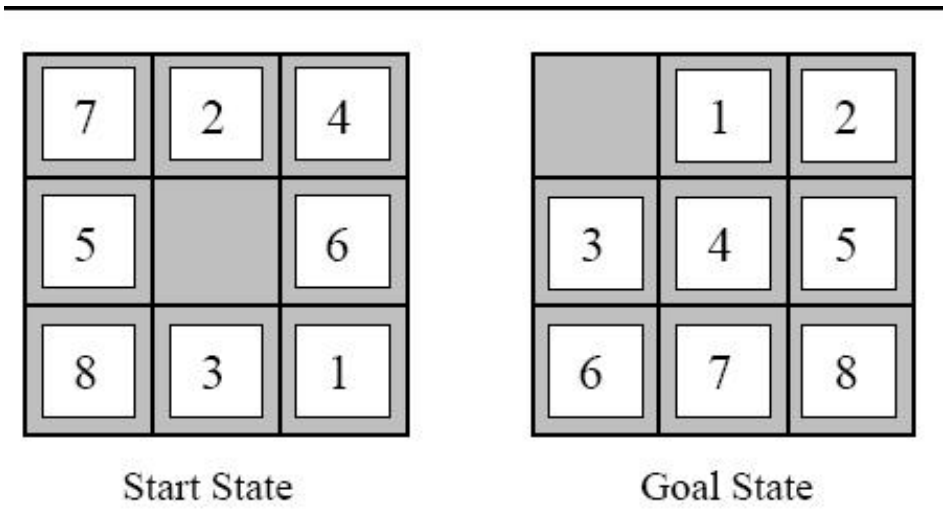
- $h_1(n)$ = number of misplaced tiles
- $h_2(n)$ = total Manhattan distance (i.e., the number of squares from desired location of each tile)
- $h_3(n)$ = 2 × number of direct tile reversals



- h_1 =?
- h_2 =?
- h_3 =?

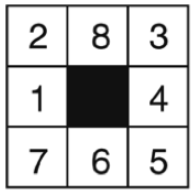
Example: 8-puzzle

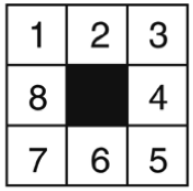
- $h_1(n)$ = number of misplaced tiles
- $h_2(n)$ = total Manhattan distance (i.e., the number of squares from desired location of each tile)
- $h_3(n) = 2 \times$ number of direct tile reversals



- $h_1 = 8$
- $h_2 = 3 + 1 + 2 + 2 + 2 + 3 + 3 + 2 = 18$
- $h_3 = 0$

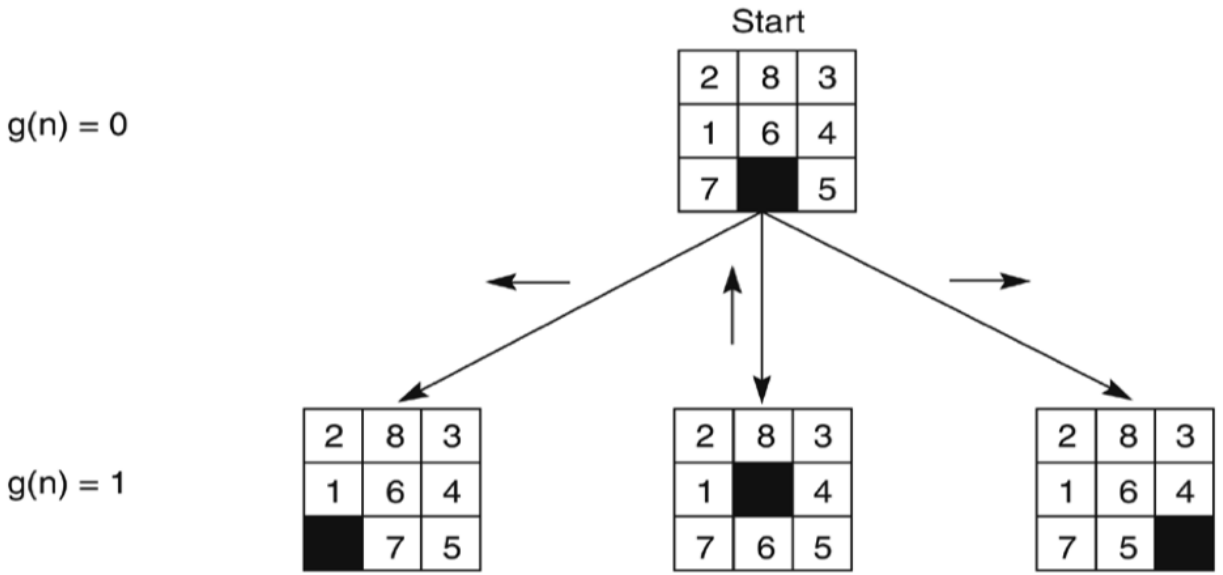
Example: 8-puzzle

	**5**	**6**	**0**
	**3**	**4**	**0**
	**5**	**6**	**0**
	Tiles out of place	Sum of distances out of place	2 x the number of direct tile reversals



Goal

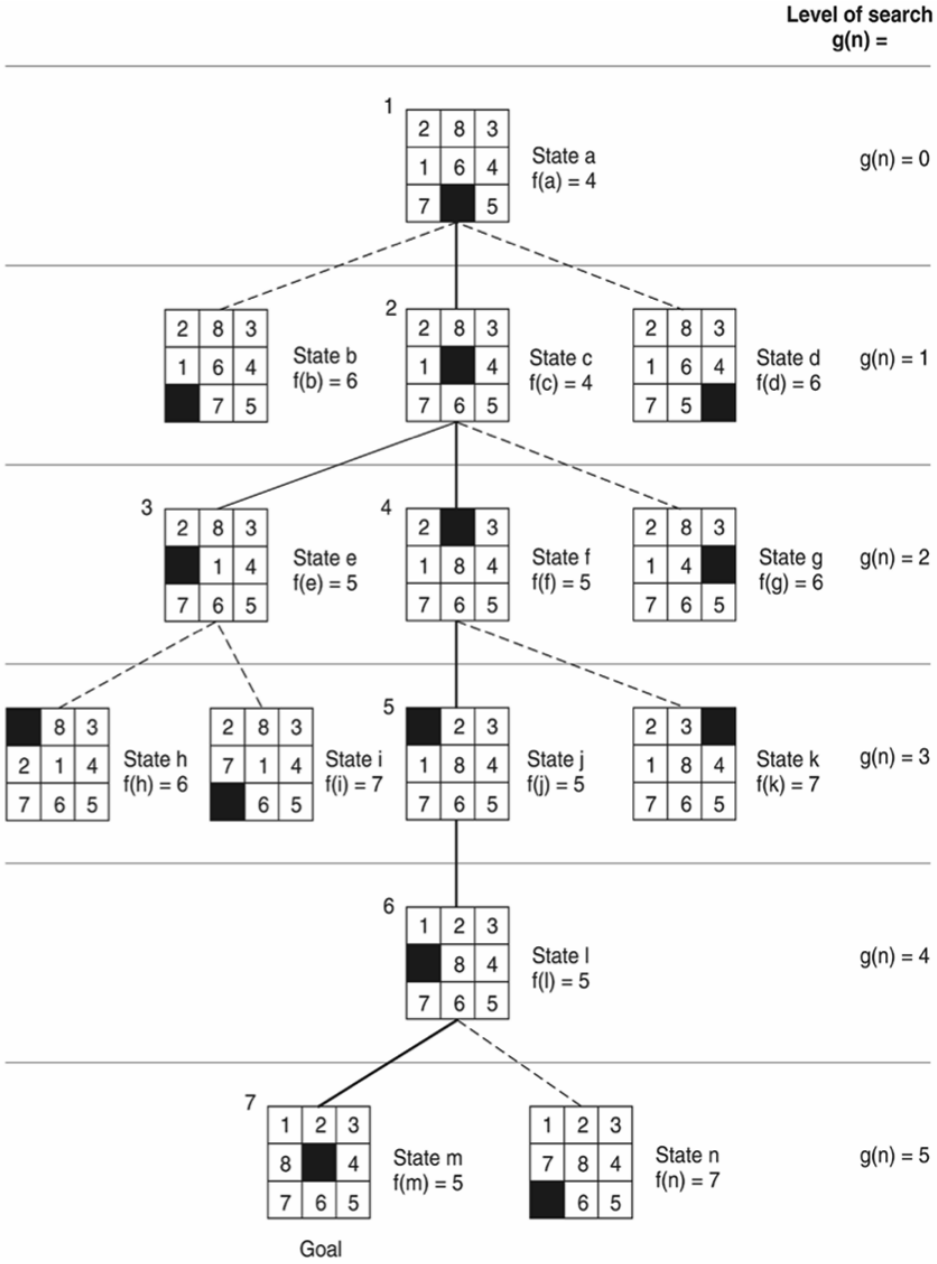
Example: 8-puzzle



Values of f(n) for each state, **6** **4** **6**

where:
f(n) = g(n) + h(n),
g(n) = actual distance from n to the start state, and
h(n) = number of tiles out of place.

Example: 8-puzzle



Example: 8-puzzle

	Search Cost (nodes generated)			Effective Branching Factor		
d	IDS	A*(h_1)	A*(h_2)	IDS	A*(h_1)	A*(h_2)
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	3644035	227	73	2.78	1.42	1.24
14	–	539	113	–	1.44	1.23
16	–	1301	211	–	1.45	1.25
18	–	3056	363	–	1.46	1.26
20	–	7276	676	–	1.47	1.27
22	–	18094	1219	–	1.48	1.28
24	–	39135	1641	–	1.48	1.26

Figure 3.29 Comparison of the search costs and effective branching factors for the ITERATIVE-DEEPENING-SEARCH and A* algorithms with h_1, h_2. Data are averaged over 100 instances of the 8-puzzle for each of various solution lengths d.

[Russell and Norvig]

Dominance

Let h_1 and h_2 be two admissible heuristics. if $h_2(n) \geq h_1(n)$ for all n, then h_2 dominates h_1.

Question

Is it possible for a computer to invent such a heuristic mechanically?

h_1 and h_2 are estimates of the remaining path length for the 8-puzzle, but they are also perfectly accurate path lengths for simplified versions of the puzzle.

Relaxation

- A problem with fewer restrictions on the actions is called a relaxed problem.
- The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem.

Relaxation

Relaxation

- A problem with fewer restrictions on the actions is called a relaxed problem.
- The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem.

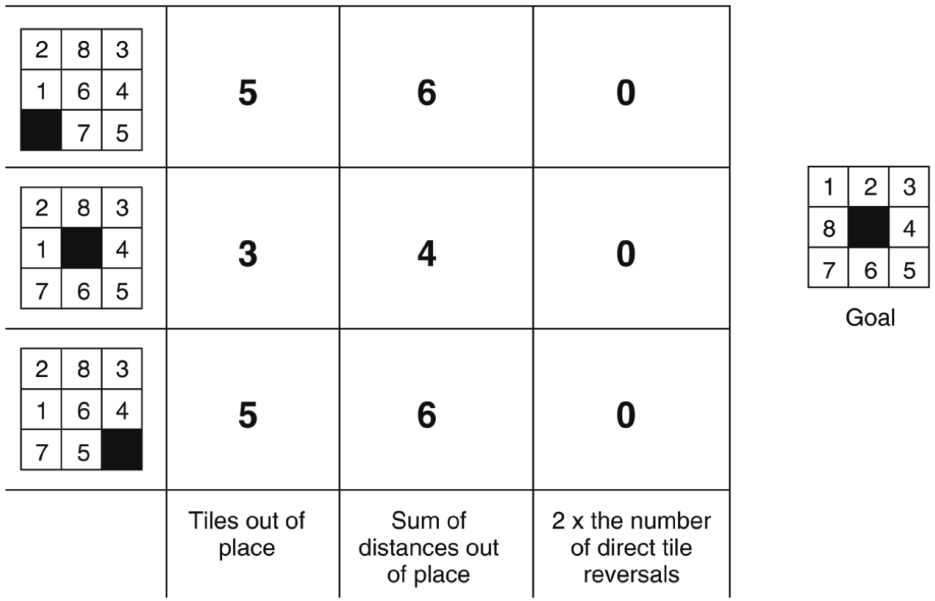
Because the relaxed problem adds edges to the state space, any optimal solution in the original problem is, by definition, also a solution in the relaxed problem; but the relaxed problem may have better solutions if the added edges provide short cuts. Hence, the cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem.

Relaxation

Idea: Constraints make life hard. Get rid of them.

"Due to TV network constraints, our 5 year mission has been reduced to 13 weeks, with a possible renewal."

Relaxation



- If the rules of the 8-puzzle are relaxed so that a tile can move anywhere, then $h_1(n)$ gives the exact solution!
- If the rules are relaxed so that a tile can move to any adjacent square, then $h_2(n)$ gives the exact solution!

Relaxation

If a problem is written down in a formal language, it is possible to construct heuristics automatically. Consider the following rule:

8-puzzle

A tile can move from square A to square B if A is horizontally or vertically adjacent to B and B is blank

We can generate three heuristics by removing one or both of the conditions from the above rule:

a) A tile can move from square A to square B

b) A tile can move from square A to square B if A is adjacent to B

c) A tile can move from square A to square B if B is blank

Relaxation

If a problem is written down in a formal language, it is possible to construct heuristics automatically. Consider the following rule:

8-puzzle

A tile can move from square A to square B if A is horizontally or vertically adjacent to B and B is blank

We can generate three heuristics by removing one or both of the conditions from the above rule:

a) A tile can move from square A to square B
b) A tile can move from square A to square B if A is adjacent to B
c) A tile can move from square A to square B if B is blank

From (b), we can derive h_2 (Manhattan distance). The reasoning is that h_2 would be the proper score if we moved each tile in turn to its destination. From (a), we can derive h_1 (misplaced tiles) because it would be the proper score if tiles could move to their intended destination in one step. Notice that it is crucial that the relaxed problems generated by this technique can be solved essentially with little search. If the relaxed problem is hard to solve, then the values of the corresponding heuristic will be expensive to obtain.

Heuristics beyond relaxation

Subproblems: relax original problem into independent subproblems

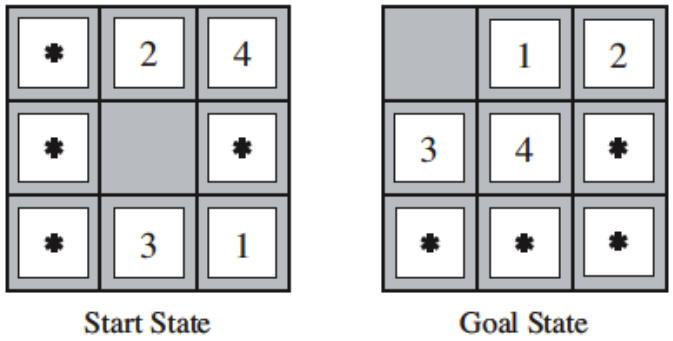


Figure 3.30 A subproblem of the 8-puzzle instance given in Figure 3.28. The task is to get tiles 1, 2, 3, and 4 into their correct positions, without worrying about what happens to the other tiles.

Learning from experience

E.g., solving lots of 8-puzzles $\rightarrow$ training data

Use ML to predict $h(n) \rightarrow h(n) = c_1 x_1(n) + c_2 x_2(n)$

[Russell and Norvig]

Heuristics beyond relaxation

Subproblems: relax original problem into independent subproblems

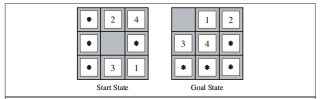


Figure 3.30 A subproblem of the 8-puzzle instance given in Figure 3.28. The task is to get tiles 1, 2, 3, and 4 into their correct positions, without worrying about what happens to the other tiles.

Learning from experience

E.g., solving lots of 8-puzzles $\rightarrow$ training data
Use ML to predict $h(n) \rightarrow h(n) = c_1 x_1(n) + c_2 x_2(n)$

[Russell and Norvig]

Admissible heuristics can also be derived from the solution cost of a subproblem of a given problem. The subproblem involves getting tiles 1, 2, 3, 4 into their correct positions. Clearly, the cost of the optimal solution of this subproblem is a lower bound on the cost of the complete problem. It turns out to be more accurate than Manhattan distance in some cases. Subproblems can be as easy as you like (think about just getting each individual tile into its correct position).

General framework

Relaxed search problem

A relaxation P' of a search problem P has costs that satisfy:

$$Cost'(n, a) \leq Cost(n, a)$$

Removing constraints $\rightarrow$ Reducing edge costs

Relaxed heuristic

Given a relaxed search problem P', define the relaxed heuristic $h(n) = h'^*(n)$, the minimum cost from n to a goal state using $Cost'(n, a)$.

General framework

Relaxed search problem

A relaxation P' of a search problem P has costs that satisfy:

$$Cost'(n, a) \leq Cost(n, a)$$

Removing constraints $\rightarrow$ Reducing edge costs

Relaxed heuristic

Given a relaxed search problem P', define the relaxed heuristic $h(n) = h'^*(n)$, the minimum cost from n to a goal state using $Cost'(n, a)$.

More formally, we define a relaxed search problem as one where the relaxed edge costs are no larger than the original edge costs. The relaxed heuristic is simply the future cost of the relaxed search problem, which by design should be efficiently computable.

Selecting heuristics

Question

If a collection of admissible heuristics $h_1, \ldots, h_m$ is available for a problem and none of them dominates any of the others, which should we choose?

Selecting heuristics

Question

If a collection of admissible heuristics $h_1, \ldots, h_m$ is available for a problem and none of them dominates any of the others, which should we choose?

$$h(n) = max\{h_1(n), \ldots, h_m(n)\}$$

Selecting heuristics

Question

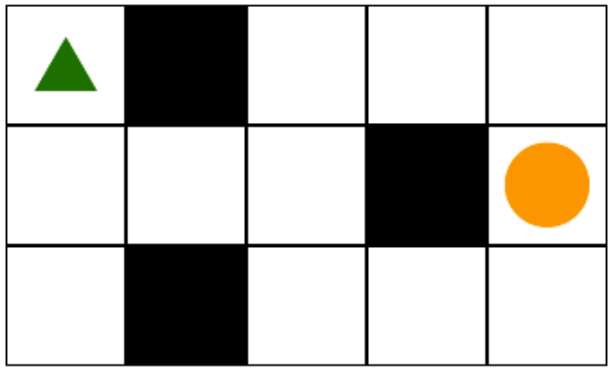
If a collection of admissible heuristics $h_1, \ldots, h_m$ is available for a problem and none of them dominates any of the others, which should we choose?

$$h(n) = max\{h_1(n), \ldots, h_m(n)\}$$

The key point is the max of two consistent heuristics is consistent. Why max? Remember that we want heuristic values to be larger. And furthermore, we can prove that taking the max leads to a consistent heuristic. Stepping back a bit, there is a deeper takeaway with A^* and relaxation here. The idea is that when you are confronted with a difficult problem, it is wise to start by solving easier versions of the problem (being an optimist). The result of solving these easier problems can then be a useful guide in solving the original problem.

Problem

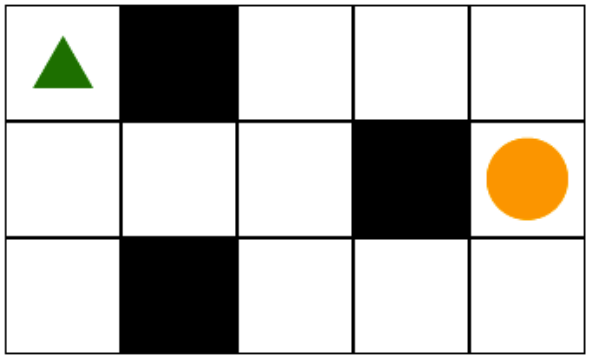
Goal: move from triangle to circle

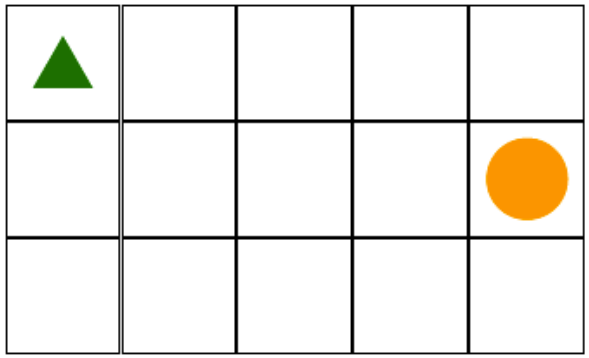


Find a good heuristic!

Problem

Goal: move from triangle to circle





$$h(n) = ManhattanDistance(n, (2, 5))$$

$$e.g., h((1, 1)) = 5$$

Problem

Goal: move from triangle to circle

$h(n) = ManhattanDistance(n, (2, 5))$

$e.g., h((1, 1)) = 5$

Here is a simple problem. Suppose states are positions (r, c) on the grid. Possible actions are moving up, down, left, or right, provided they don't move you into a wall or off the grid; and all edge costs are 1. The start state is at the triangle at $(1, 1)$, and the goal state is the circle at position $(2, 5)$. By removing the walls, then we can compute the solution in closed form: it's just the Manhattan distance between the start state and goal state. Specifically, $ManhattanDistance((r_1, c_1), (r_2, c_2)) = |r_1 - r_2| + |c_1 - c_2|$.

Summary

- Informed search: A^* expands nodes with minimal $f(n) = g(n) + h(n)$.
- Consistent and admissible heuristics.
- How to construct heuristics?
  - Relaxation
  - Subproblems
  - Learning

Next lecture

Intro to Graphical Models!