COE206 – Principles of Artificial Intelligence

Mustafa MISIR

Istinye University, Department of Computer Engineering

mustafa.misir@istinye.edu.tr

http://mustafamisir.github.io
http://memoryrlab.github.io

ARTIFICIAL INTELLIGENCE IN MEDICINE RESEARCH CENTER
ISTINYE UNIVERSITY

MEMORY

L3-2: Problem Solving by Search

Informed (Heuristic) Search

Informed Search

An **informed search** strategy—one that uses problem-specific knowledge beyond the definition of the problem itself—can find solutions more efficiently than an **uninformed** strategy.

- operating thorough an evaluation function, $f(n)$

The general approach we consider is called **best-first search**[1].

- $f(n)$ is constructed as a cost estimate, so the node with the lowest evaluation is expanded first.
- $f(n)$ tends to utilize a heuristic function, e.g. $h(n) =$ estimated cost of the cheapest path from the state at node n to a goal state.

[1] https://en.wikipedia.org/wiki/Best-first_search

Outline

- Greedy Best-First Search
- A* Search
- Heuristic Functions

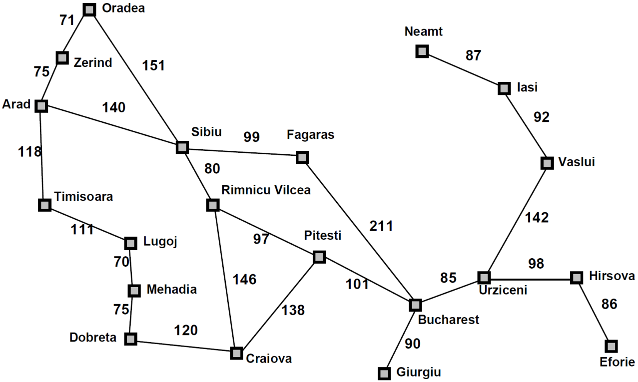
Greedy Best-first Search

Expand the node that is closest to the goal, likely to lead to a solution quickly.

- evaluates nodes by using just the heuristic function, i.e. $f(n) = h(n)$

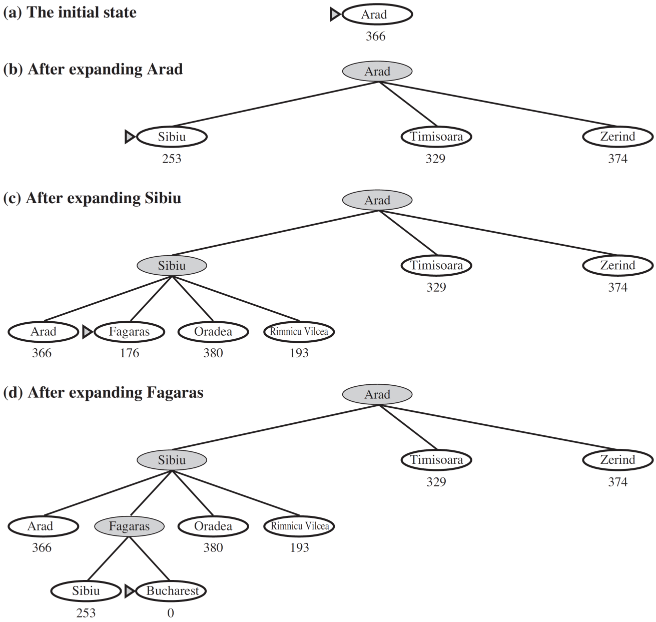
Greedy Best-first Search – Straight-line Distance Heuristic

If the goal is Bucharest, we need to know the straight-line distances to Bucharest.



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


(a) The initial state
Arad
366
(b) After expanding Arad
Arad
Sibiu
253
Timisoara
329
Zerind
374
(c) After expanding Sibiu
Arad
Sibiu
Timisoara
329
Zerind
374
Arad
366
Fagaras
176
Oradea
380
Rimnicu Vilcea
193
(d) After expanding Fagaras
Arad
Sibiu
Timisoara
329
Zerind
374
Arad
366
Fagaras
Oradea
380
Rimnicu Vilcea
193
Sibiu
253
Bucharest
0

Greedy Best-first Search – Properties

- **Completeness**[2]: No – can get stuck in loops; e.g. getting from Iasi to Fagaras: Neamt is expanded first because it is closest (straight-line) to Fagaras, but it is a dead end.
- **Time Complexity**[3]: $O(b^m)$
- **Space Complexity**[4]: $O(b^m)$
- **Optimality**[5]: No

---

[2] Is the algorithm guaranteed to find a solution when there is one?

[3] How long does it take to find a solution?

[4] How much memory is needed to perform the search?

[5] Does the strategy find the optimal solution?

A* Search

The most widely known form of **best-first search** is called **A* search**.

- It evaluates nodes by combining $g(n)$, the cost to reach the node, and $h(n)$, the cost to get from the node to the goal:

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

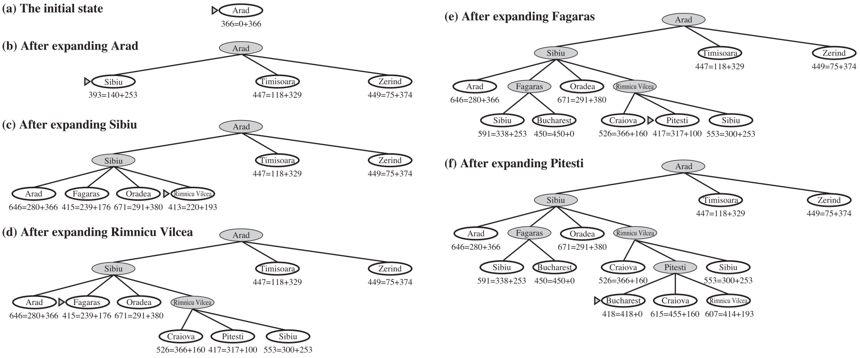
Since $g(n)$ gives the path cost from the start node to node n, and $h(n)$ is the estimated cost of the cheapest path from n to the goal, we have

$$f(n) = \text{estimated cost of the cheapest solution through } n$$

The algorithm is identical to **Uniform-Cost Search** (UCS) except that A* uses $g + h$ instead of g.

A* Search

The h values are the straight-line distances to Bucharest



A* Search – Optimality (Admissibility)

An admissible heuristic is one that never overestimates the cost to reach the goal (always less than or equal to the actual cost).

$h(n) \leq h^*(n)$ where $h^*(n)$ is the true cost from n

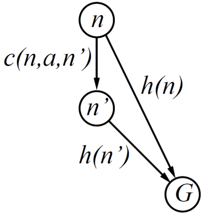
Admissible heuristics are by nature optimistic because they think the cost of solving the problem is less than it actually is.

A* Search – Optimality (Consistency)

A heuristic $h(n)$ is consistent (monotone) if

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

Triangle inequality, a side of a triangle cannot be longer than the sum of the other two sides.



A consistent heuristic[6] is also admissible.

[6] https://en.wikipedia.org/wiki/Consistent_heuristic

A^* Search – Optimality (Consistency)

Define two nodes, n and n', where n' is a successor of n;
$g(n') = g(n) + c(n, a, n')$ considering that n' is a successor of n and $h(n) \leq c(n, a, n') + h(n')$

If h is consistent, we have

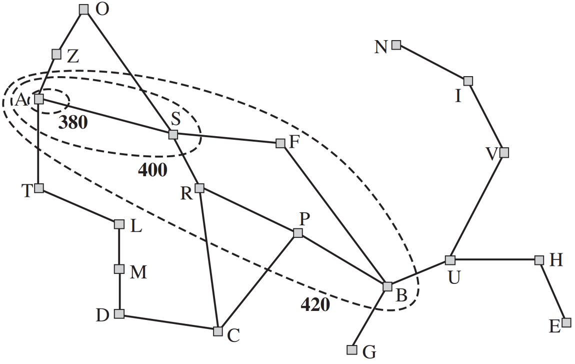
$$
\begin{aligned}
f(n') &= g(n') + h(n') \\
&= g(n) + c(n, a, n') + h(n') \\
&\geq g(n) + h(n) \\
&= f(n)
\end{aligned}
$$

As $f(n') \geq f(n)$, the values of $f(n)$ are monotonically non-decreasing along that path

A* Search – Optimality

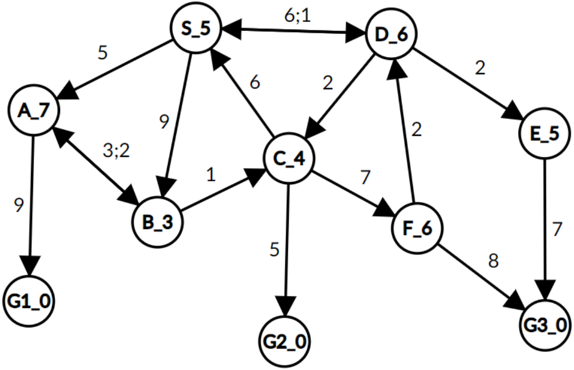
With a monotonic heuristic, we can interpret A* as searching through contours.

- Showing contours at $f = 380$, $f = 400$, and $f = 420$, with Arad as the start state.
- Nodes inside a given contour have f-costs less than or equal to the contour value.



A* Search, e.g. Graph Search[7]

Underscore values in the nodes refer to the estimated distance to one of the goal states from the current node, i.e. $h(n)$



[7] A* search example by John Levine (U. Strathclyde): https://www.youtube.com/watch?v=6TsL96NAZCo

A* Search – Properties

- **Completeness**[8]: Yes
- **Time Complexity**[9]: Exponential (depending on the heuristic function)
- **Space Complexity**[10]: Keeps all the nodes in memory
- **Optimality**[11]: Yes

8 Is the algorithm guaranteed to find a solution when there is one?

9 How long does it take to find a solution?

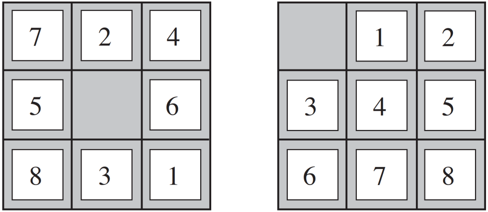
10 How much memory is needed to perform the search?

11 Does the strategy find the optimal solution?

Heuristic Functions – 8-Puzzle

The objective is to slide the tiles horizontally or vertically into the empty space until the configuration matches the goal configuration

A solution with 26 steps long:

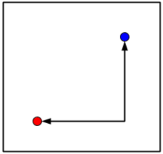


Start State Goal State

Heuristic Functions – 8-Puzzle[14]

2 heuristic function suggestions for **A* search**:

- h_1: the number of misplaced tiles
  - admissible as it is clear that any tile that is out of place must be moved at least once, i.e. hamming distance[12]
- h_2: the sum of the distances (horizontal and vertical) of the tiles from their goal positions, i.e. Manhattan distance[13]
  - admissible as all any move can do is move one tile one step closer to the goal.

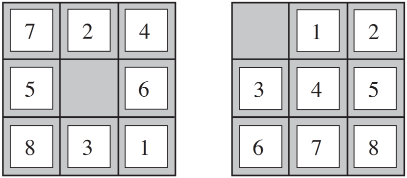


[12] https://en.wikipedia.org/wiki/Hamming_distance

[13] https://en.wikipedia.org/wiki/Taxicab_geometry

[14] image source: https://subscription.packtpub.com/book/big_data_and_business_intelligence/9781785882104/6/ch06lvl1sec40/measuring-distance-or-similarity

Heuristic Functions – 8-Puzzle



Start State Goal State

For the given example with the solution of 26 steps long[15]:

- $h_1 = 8$, as all the tiles are misplaced at the start state
- $h_2 = 3 + 1 + 2 + 2 + 2 + 3 + 3 + 2 = 18$, considering the number of moves to the goal state as the distance

Due to admissibility, neither of these overestimates the true solution cost, which is 26.

[15] $h_2 = 18$ is closer to 26 than $h_1 = 8$

Heuristic Functions – Accuracy on Performance[17]

One way to characterize the quality of a heuristic is the **effective branching factor**[16] b^*.

- If the total number of nodes generated by A^* for a particular problem is N and the solution depth is d,
- then b^* is the branching factor that a uniform tree of depth d would have to have in order to contain $N+1$ nodes.

$$N + 1 = 1 + b^* + (b^*)^2 + \ldots + (b^*)^d$$
$$= (1 - (b^*)^{n+1})/(1 - b^*)$$

e.g. if A^* finds a solution at depth 5 using 52 nodes, the effective branching factor is 1.92.

- The effective branching factor can vary across problem instances, but usually it is fairly constant for sufficiently hard problems.

[16] the number of successors generated by a *typical* node for a given search problem:
http://ozark.hendrix.edu/~ferrer/courses/335/f11/lectures/effective-branching.html

[17] the sum of the geometric progression series: https://en.wikipedia.org/wiki/Geometric_progression

Heuristic Functions – Accuracy on Performance

Average performances of **iterative deepening search (IDS)** and with A^* tree search using both h_1 and h_2 on 100 randomly generated problems

- h_2 is better[18] than h_1, and A^* is far better than **IDS**.

	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

[18] as expected from the earlier calculation of the solution with the step size 26, where $h_1 = 8$ and $h_2 = 18$ – for this specific example h_2 is more reasonable since $h_1 < h_2 < h^*$