Grundlagen der künstlichen Intelligenz – Informed Search

Matthias Althoff

TU München

October 25, 2019

Organization

1. Informed Search Strategies
2. Heuristic Functions

The content is covered in the AI book by the section "Solving Problems by Searching", Sec. 5-6.

Learning Outcomes

- You can describe the difference between informed and uninformed search.
- You can apply Greedy Best-First Search and A*.
- You can discuss the influence of a heuristic on the search result.
- You can decide whether a heuristic dominates another heuristic.
- You can explain methods how to systematically obtain heuristics for search problems.
- You can create simple heuristics to improve the performance of search problems.

Informed Search

- Requires problem-specific knowledge beyond the problem definition.
- Can find solutions more efficiently than uninformed search.
- Informed search uses indications whether a state is more promising than another to reach a goal.
- The choice of the next node is based on an **evaluation function** $f(n)$, which itself is often based on a **heuristic function** $h(n)$.
- $h(n)$ is problem specific with the only constraints that it is nonnegative and $h(\hat{n}) = 0$, where $\hat{n}$ is a goal node.

All presented informed search strategies are identical to **uniform-cost search**, except that f instead of g is used in the priority queue.

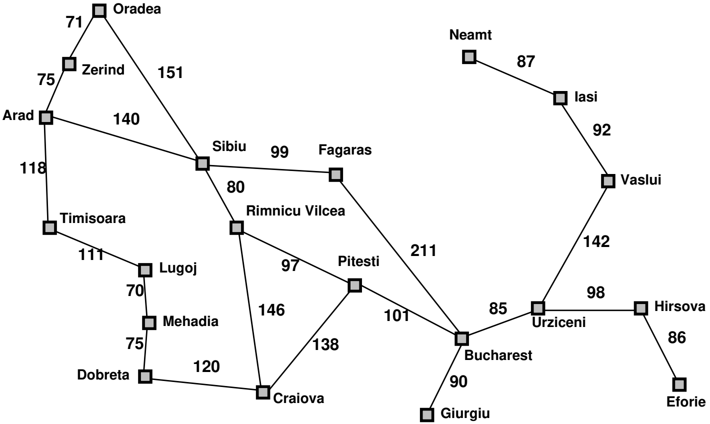
Heuristics

Heuristics refers to the art of achieving good solutions with limited knowledge and time based on experience.

Greedy Best-First Search: Idea (InformedSearch.ipynb)

Expands the node that is closest to the goal by using just the heuristic function so that $f(n) = h(n)$.

Example: Route-finding in Romania. We use the straight line distance to the goal as $h(n)$:

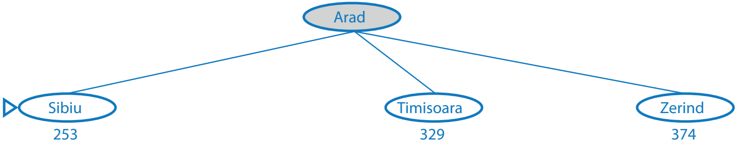


Straight−line distance to Bucharest

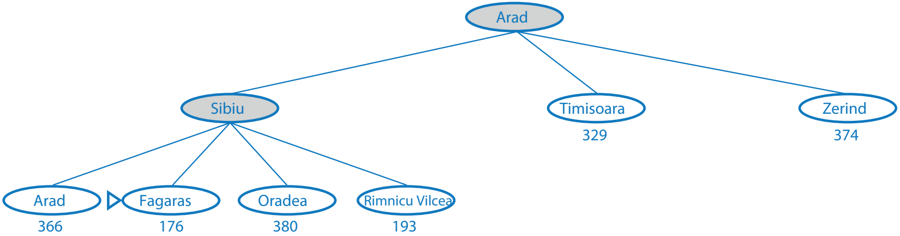
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy Best-First Search: Example (Step 1)

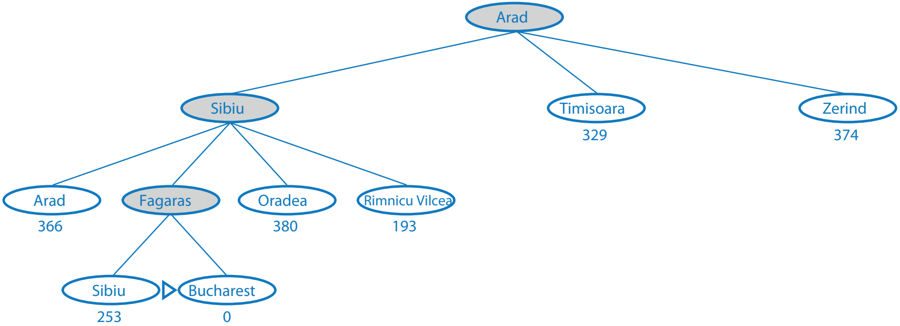
Greedy Best-First Search: Example (Step 2)



Greedy Best-First Search: Example (Step 3)



Greedy Best-First Search: Example (Step 4)



Note that the solution is not optimal since the path is 32 km longer than through Rimnicu Vilcea and Pitesti.

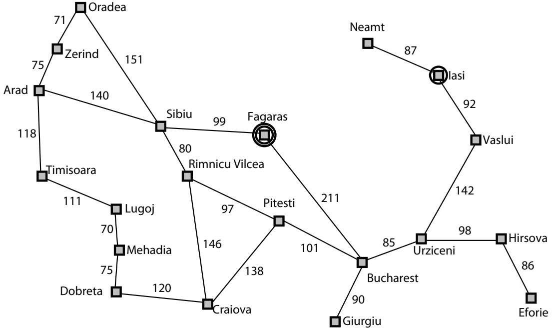
Tweedback Question

Greedy best-first search using **tree search**; start: Iasi, goal: Faragas.
Neamt is first expanded. Next,

A no further node expanded.

B Iasis is expanded. Afterwards, Neamt is again expanded due to the closest straight line distance.

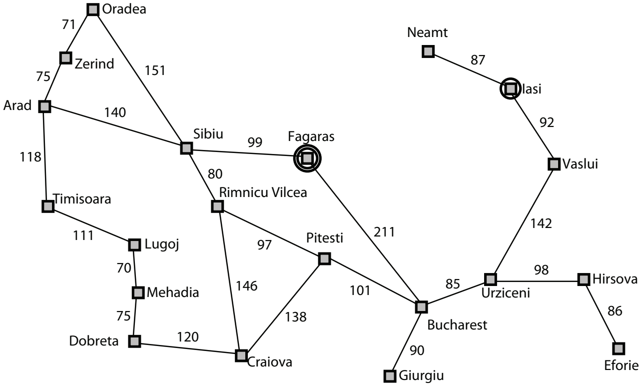
C Iasis is expanded. Afterwards, Vaslui is expanded, since Neamt has been expanded before.



Greedy Best-First Search: Another Example

Greedy best-first search is incomplete when not using an **explored set** (tree search), even on finite state spaces.

Example: From Iasi to Faragas: Neamt is first expanded due to closer straight line distance, but this is a dead end.



Tweedback Questions

- Is greedy best-first search optimal?
- What is the time complexity of greedy best-first search?
  - A $\mathcal{O}(b^m)$.
  - B $\mathcal{O}(b^d)$.

Reminder: Branching factor **b**, depth **d**, maximum length **m** of any path.

Greedy Best-First Search: Performance

Reminder: Branching factor **b**, depth **d**, maximum length **m** of any path.

- **Completeness**: Yes, if graph search is used, otherwise no (see previous example).
- **Optimality**: No (see previous example).
- **Time complexity**: The worst-case is that the heuristic is misleading the search such that the solution is found last: $\mathcal{O}(b^m)$. But a good heuristic can provide a dramatic improvement.
- **Space complexity**: Equals time complexity since all nodes are stored: $\mathcal{O}(b^m)$.

A* search: Idea

- The most widely known informed search is A* search (pronounced "A-star search").
- It evaluates nodes by combining the path cost $g(n)$ and the estimated cost to the goal $h(n)$:

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

  where $h(n)$ has to be **admissible**. An admissible heuristic is an underestimation, i.e., it has to be less than the actual cost.

  $\rightarrow f(n)$ never overestimates the cost to the goal and thus the algorithm keeps searching for paths that might have a lower cost to the goal than those found previously.

Consistent Heuristics

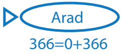
- A slightly stronger condition called **consistency** (or sometimes **monotonicity**) is required when applying A* to graph search.
- A heuristic is consistent if for given costs of transitions $c(n, a, n')$, we have that for all nodes n and its successors n'

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

- This is a form of the general **triangle inequality**.
- It is fairly easy to show that every consistent heuristic is also admissible.

A* Search: Example (Step 1) (InformedSearch.ipynb)

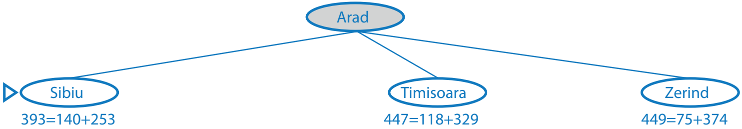
Straight line distance is an underestimation of the cost to the goal, which is used for $h(n)$.



Nodes are labeled with $f = g + h$

A* Search: Example (Step 2)

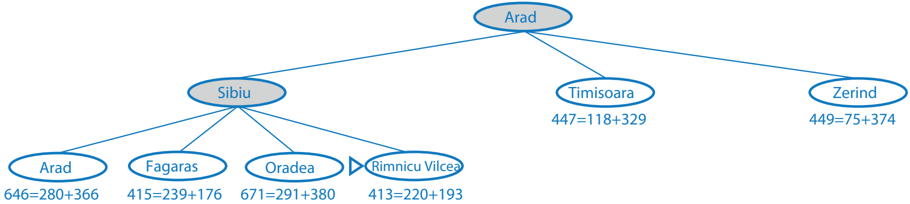
Straight line distance is an underestimation of the cost to the goal, which is used for $h(n)$.



Nodes are labeled with $f = g + h$

A* Search: Example (Step 3)

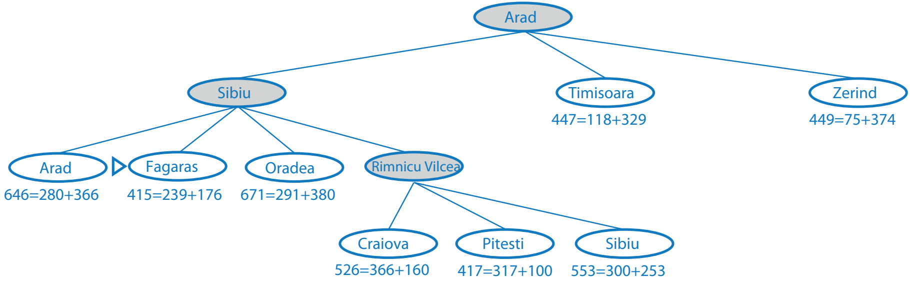
Straight line distance is an underestimation of the cost to the goal, which is used for $h(n)$.



Nodes are labeled with $f = g + h$

A* Search: Example (Step 4)

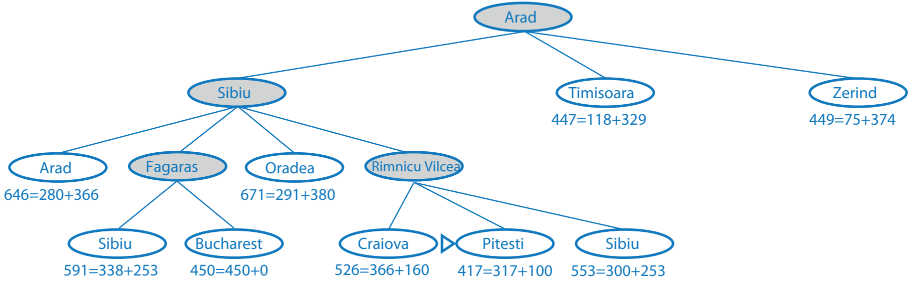
Straight line distance is an underestimation of the cost to the goal, which is used for $h(n)$.



Nodes are labeled with $f = g + h$

A* Search: Example (Step 5)

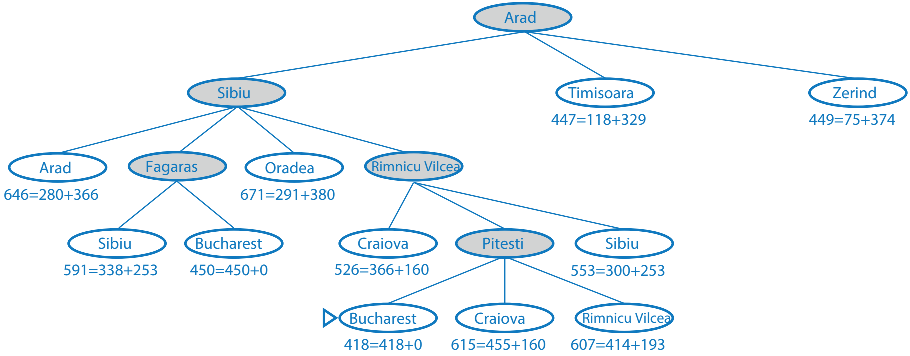
Straight line distance is an underestimation of the cost to the goal, which is used for $h(n)$.



Nodes are labeled with $f = g + h$

A* Search: Example (Step 6)

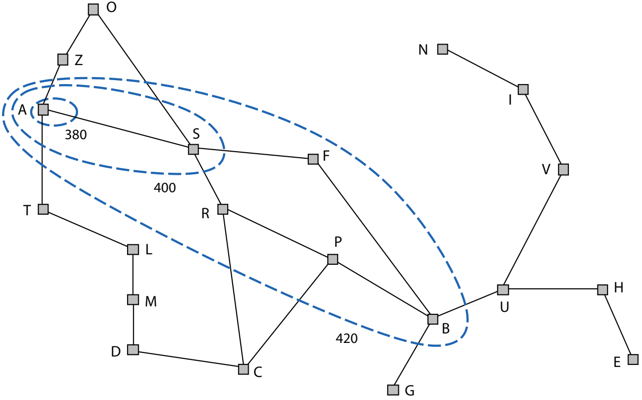
Straight line distance is an underestimation of the cost to the goal, which is used for $h(n)$.



Nodes are labeled with $f = g + h$

A* Search: Effects of the Heuristic

- The heuristic "steers" the search towards the goal.
- A* expands nodes in order of increasing f value, so that "f-contours" of nodes are gradually added.
- Each contour i includes all nodes with $f \leq f_i$, where $f_i < f_{i+1}$.



Tweedback Question

Given the cost C^* of the optimal path.

Does A* expand nodes, where $f(n) \geq C^*$?

A* Search: Pruning

- Given the cost C^* of the optimal path, it is obvious that only paths are expanded with $f(n) < C^*$.
  $\rightarrow$ A* never expands nodes, where $f(n) \geq C^*$.
- For instance, Timisoara is not expanded in the previous example. We say that the subtree below Timisoara is pruned.
- The concept of pruning – eliminating possibilities from consideration without having to examine them – brings enormous time savings and is similarly done in other areas of AI.

A* Search: Performance

Reminder: Branching factor **b**, depth **d**, maximum length **m** of any path.

Time and space complexity of A* is quite involved, and we will not derive the results. They are presented in terms of the **relative error** $\epsilon = (h^* - h)/h^*$, where h is the estimated and h^* is the actual cost from the root to the goal.

- **Completeness**: Yes, if costs are greater than 0 (otherwise infinite optimal paths of zero cost exist).
- **Optimality**: Yes (if cost are positive); heuristic has to be admissible for the tree-search version and consistent for the graph-search version.
- **Time complexity**: We only consider the easiest case: The state space has a single goal and all actions are reversible: $\mathcal{O}(b^{\epsilon d})$.
- **Space complexity**: Equals time complexity since all nodes are stored (Why does this also hold for the tree-search version?).

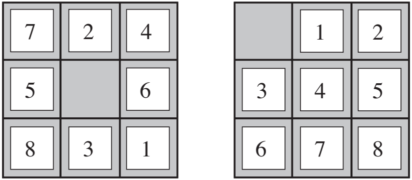
Alternatives of A* Search

One of the big disadvantages of A* search is the possibly huge space consumption. This can be alleviated by extensions, e.g.:

- **Iterative-deepening A***: adapts the idea of iterative deepening to A*. The main difference is that the f-cost $(g + h)$ is used for cutoff, rather than the depth.
- **Recursive best-first search**: a simple recursive algorithm with linear space complexity. Its structure is similar to the one of recursive-depth-first search, but keeps track of the f-value of the best alternative path.
- **Memory-bounded A*** and **simplified memory-bounded A***: These algorithms work just like A* until the memory is full. The algorithms drop less promising paths to free memory.

Heuristic Functions

For the shortest route in Romania, the straight line distance is an obvious underapproximating heuristic.



Start State   Goal State

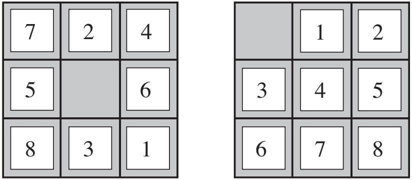
This is not always so easy, as we will show for the 8-puzzle.

Heuristic Functions for the 8-Puzzle

Two commonly used candidates that underestimate the costs to the goal:

- $\mathbf{h_1}$: the number of misplaced tiles (e.g., in the figure $h_1 = 8$).
  Why admissible? Misplaced tile has to be moved at least once.
- $\mathbf{h_2}$: the sum of the distances to the goal positions using horizontal and vertical movement.
  Why admissible? All a move can do is bring the tile one step closer.
  In the figure below:

tile	1	2	3	4	5	6	7	8	**sum**
steps	3	1	2	2	2	3	3	2	18



(the true cost is 26)

Effective Branching Factor

One way of characterizing the quality of a heuristic is the effective branching factor b^*.

Given:

- Number of nodes N generated by the A* search.
- A uniform tree with depth d (each node has the same fractional number b^* of children)

Thus,

$$N + 1 = 1 + b^* + (b^*)^2 + \ldots + (b^*)^d.$$

E.g., if A* generates a solution at depth 5 using 52 nodes, $b^* = 1.92$ since

$$53 \approx 1 + 1.92 + (1.92)^2 + \ldots + (1.92)^5.$$

The branching factor makes it possible to compare heuristic applied to problems of different size. Why?

Comparison of the Heuristic for the 8-Puzzle

100 random problems for each depth d:

	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

Tweedback Question

Is h_2 always better or equally good as h_1?

Reminder:

- $\mathbf{h_1}$: the number of misplaced tiles.
- $\mathbf{h_2}$: the sum of the distances to the goal positions using horizontal and vertical movement.

Domination of a Heuristic

Question: Is h_2 always better or equally good as h_1?

Answer: Yes.

Reason:

- For every node, we have that $h_2(n) \geq h_1(n)$. We say that h_2 dominates h_1.
- A* using h_2 will never expand more nodes than with h_1 (except possibly for some nodes with $f(n) = C^*$):

  A* expands all nodes with

  $$f(n) < C^* \leftrightarrow h(n) < C^* - g(n),$$

  where $g(n)$ is fixed. Since $h_2(n) > h_1(n)$, fewer nodes are expanded.

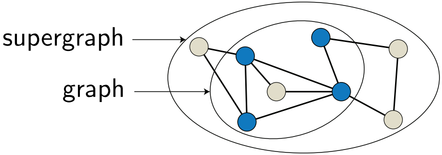
Heuristics from Relaxed Problems

Idea: The previous heuristics h_1 and h_2 are perfectly accurate for **simplified** versions of the 8-puzzle.

Method: Formalize a problem definition and remove restrictions.

Result:

- One obtains a **relaxed** problem, i.e., a problem with more freedom, whose state-space graph is a supergraph of the original one (see figure).
- An optimal solution in the original problem is automatically a solution in the relaxed problem, but the relaxed problem might have better solutions due to added edges.
- Hence, the cost of an optimal solution in the relaxed problem is underapproximative.



Heuristics from Relaxed Problems: 8-Puzzle Example

A tile can move from square A to B if $\Phi_1 \wedge \Phi_2$, where

- Φ_1: B is blank
- Φ_2: A is adjacent to B

We generate three relaxed problems by removing one or two conditions:

1. remove Φ_1: A tile can move from square A to B if A is adjacent to B.
2. remove Φ_2: A tile can move from square A to B if B is blank.
3. remove Φ_1 and Φ_2: A tile can move from square A to B.

- From the first relaxed problem, we can generate h_2 and from the third relaxed problem, we can derive h_1.
- If one is not sure which heuristic is better, one can apply in each step

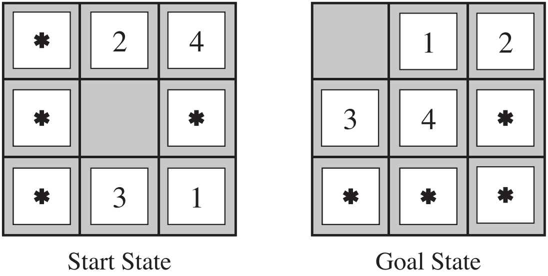
$$h(n) = \max\{h_1(n), h_2(n), \dots, h_M(n)\}.$$

Why?

Heuristics from Pattern Databases

- Underapproximative heuristics can also be obtained from subproblems.
- Those solution costs are underapproximations and can be stored in a database.
- The number of subproblems has to be much less than the original problems to not exceed storage capacities.

The figure shows a subproblem of an 8-puzzle, where only the first 4 tiles have to be brought into a goal position:



Summary

- Informed search methods require a heuristic function that estimates the cost $h(n)$ from a node n to the goal:
  - **Greedy best-first search** expands nodes with minimal $h(n)$, which is not always optimal.
  - **A* search** expands nodes with minimal $f(n) = g(n) + h(n)$. A* is complete and optimal for underapproximative $h(n)$.
- The performance of informed search depends on the quality of the heuristic. Possibilities to obtain good heuristics are **relaxed problems** and **pattern databases**.