

Solving Problems by Searching: Informed (Heuristic) Strategies

Kowndinya Boyalakuntla

Tuesday, September 8, 2026



Best-first Search

- Informed search strategies use problem-specific knowledge beyond the definition of the problem.
- General approach : Best-first Search, identical to uniform-cost search except for the use of a general **evaluation function** $f(n)$ instead of the **cost function** $g(n)$ for choosing nodes to expand.
- Evaluation function f often contains a component h known as the **heuristic function**.

Definition

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

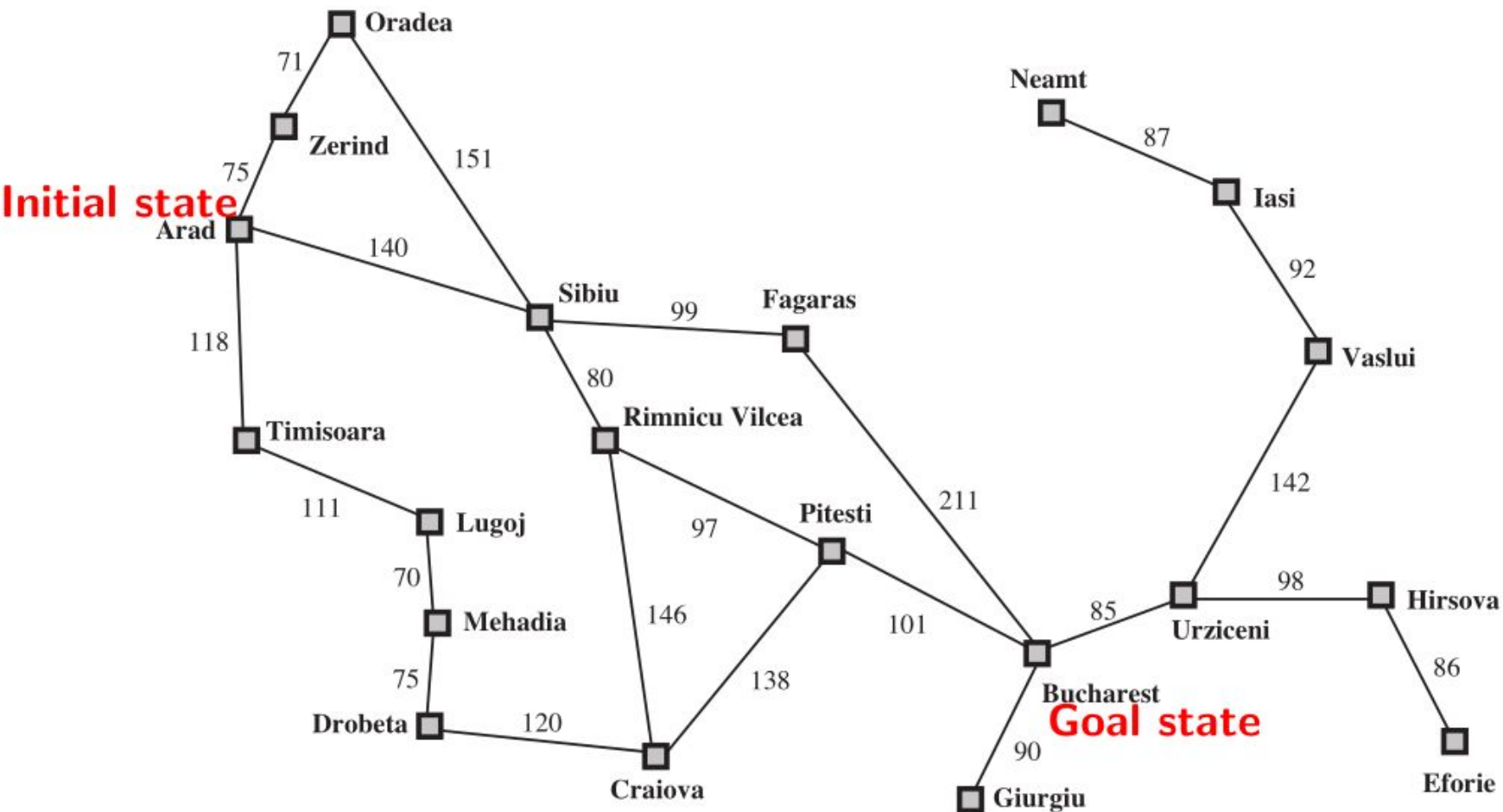
Greedy best-first search :

Set $f(n) = h(n)$. Consequently, the node that *seems* to be the closest to the goal is expanded first.

Route-finding problem using a simplified road map of Romania

Straight-Line Distance (SLD) heuristic

Choose $h(n)$ as the straight-line distance from node n to the goal state.

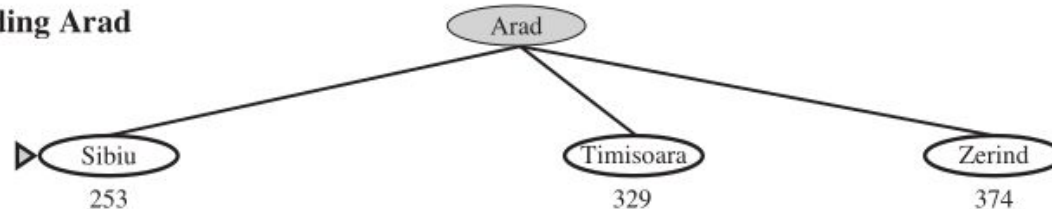


Stages in a greedy best-first tree search

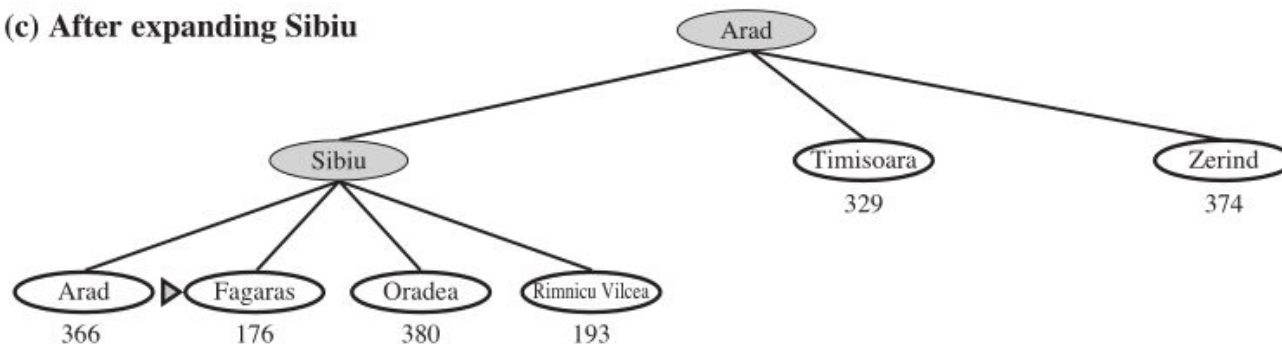
(a) The initial state



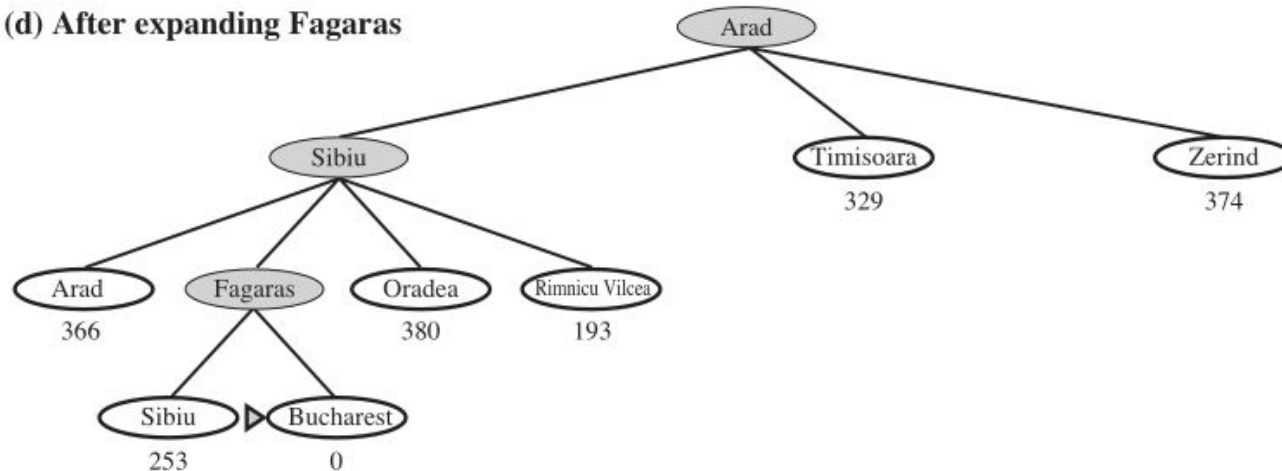
(b) After expanding Arad



(c) After expanding Sibiu



(d) After expanding Fagaras



Properties of Greedy Best-first Search (GBFS)

- **Complete** : No - can get stuck (think of an example)..
- **Optimal** : no.
- **Time complexity** : $O(b^m)$, where m is the length of the longest path along the tree. GBFS can potentially search the entire tree.
- **Space complexity** : $O(b^m)$, we may have to remember the whole tree to avoid redundant paths.
- The complexity can be reduced substantially with a good heuristic.

A^* search

- A^* is the most popular search algorithm.
- A^* is similar to the Greedy Best-first Search algorithm (and also to uniform-cost search), but it also takes the distance already traveled into account.
- The evaluation function f is defined as

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

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

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

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

Stages in an A^* search

Heuristic function $h(n)$: straight-line distances to Bucharest starting from any city.

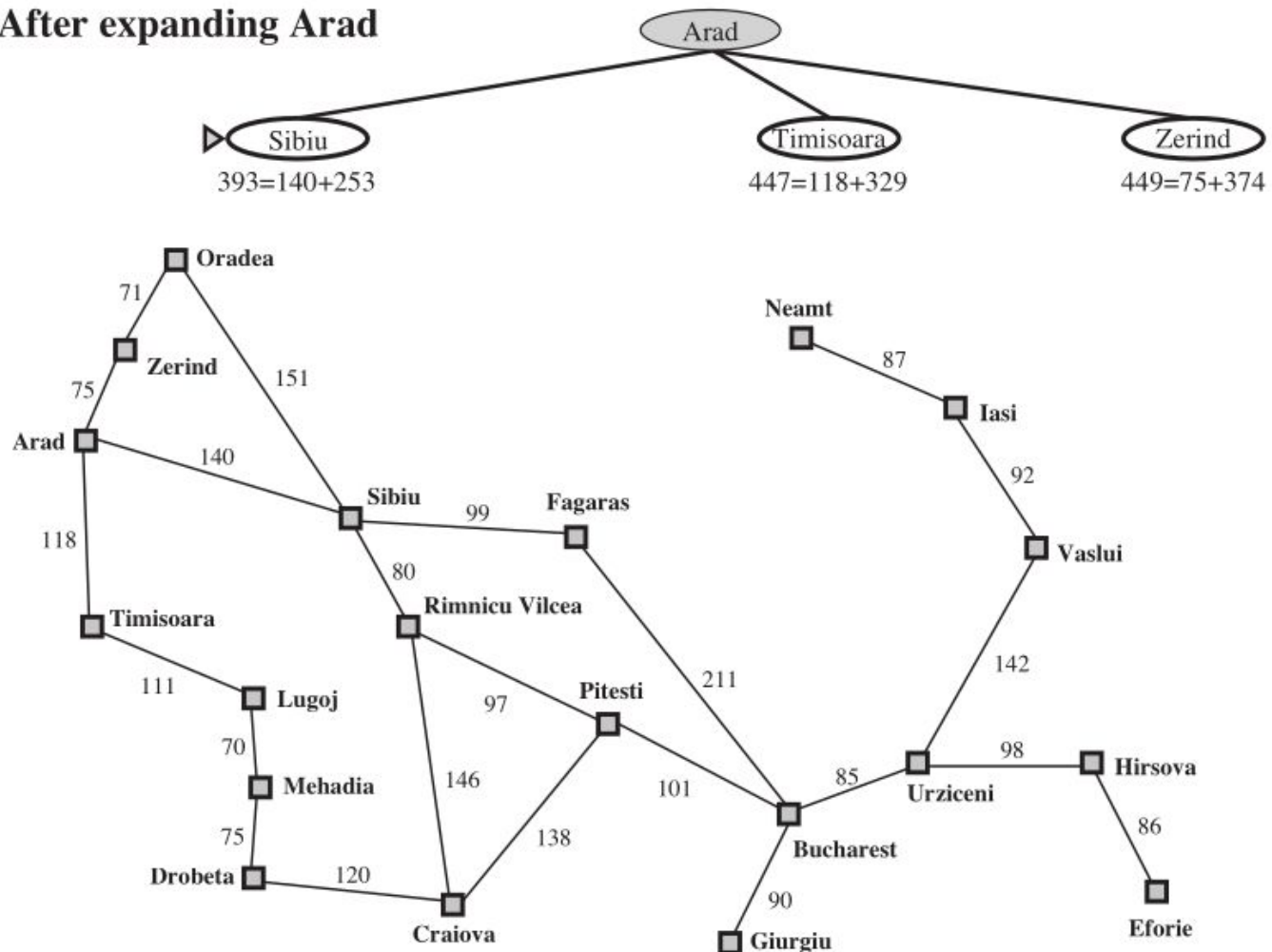
Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

Stages in an A^* search

(a) The initial state

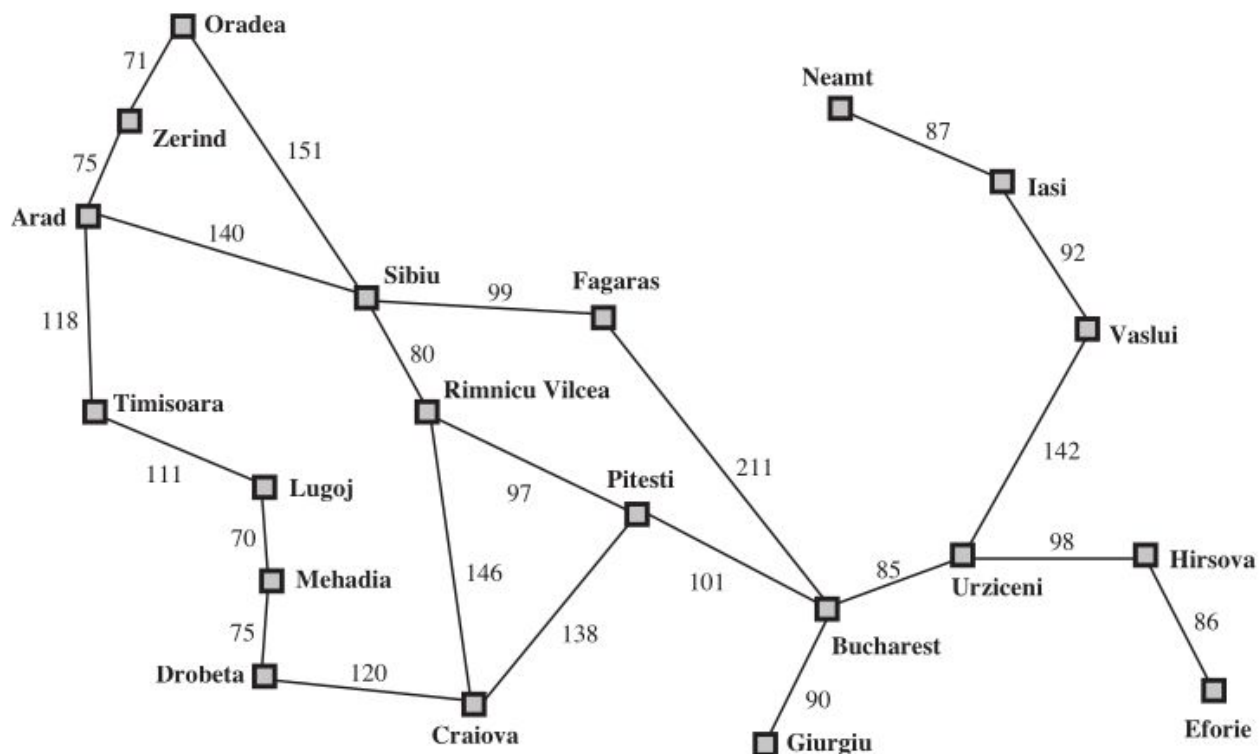
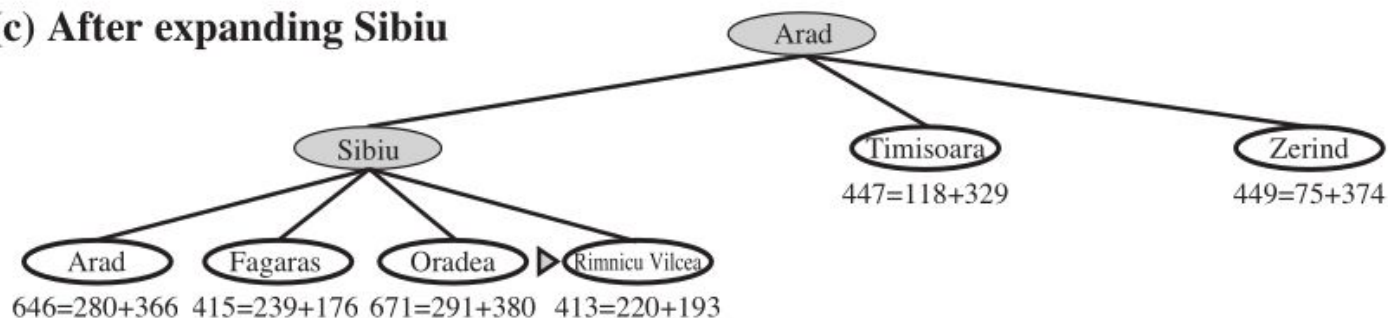


(b) After expanding Arad



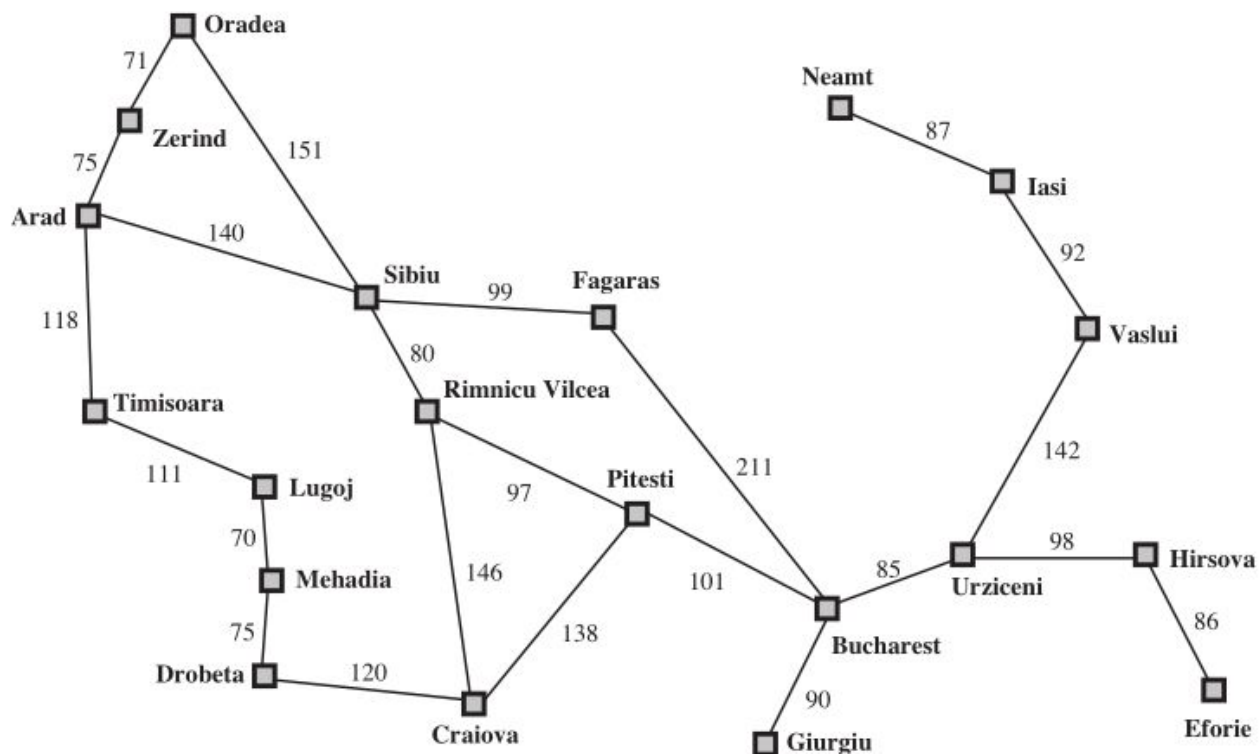
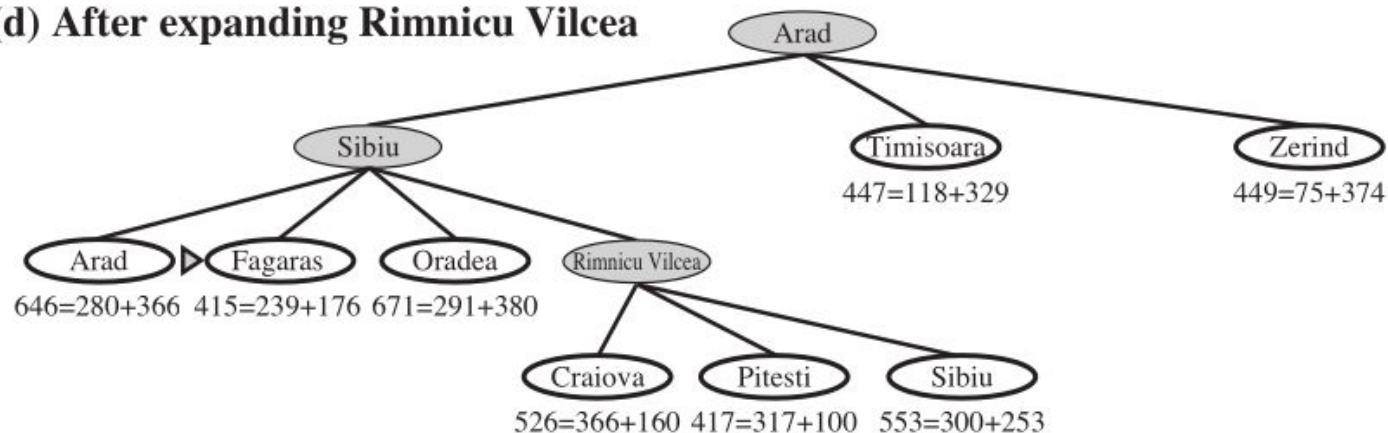
Stages in an A^* search

(c) After expanding Sibiu



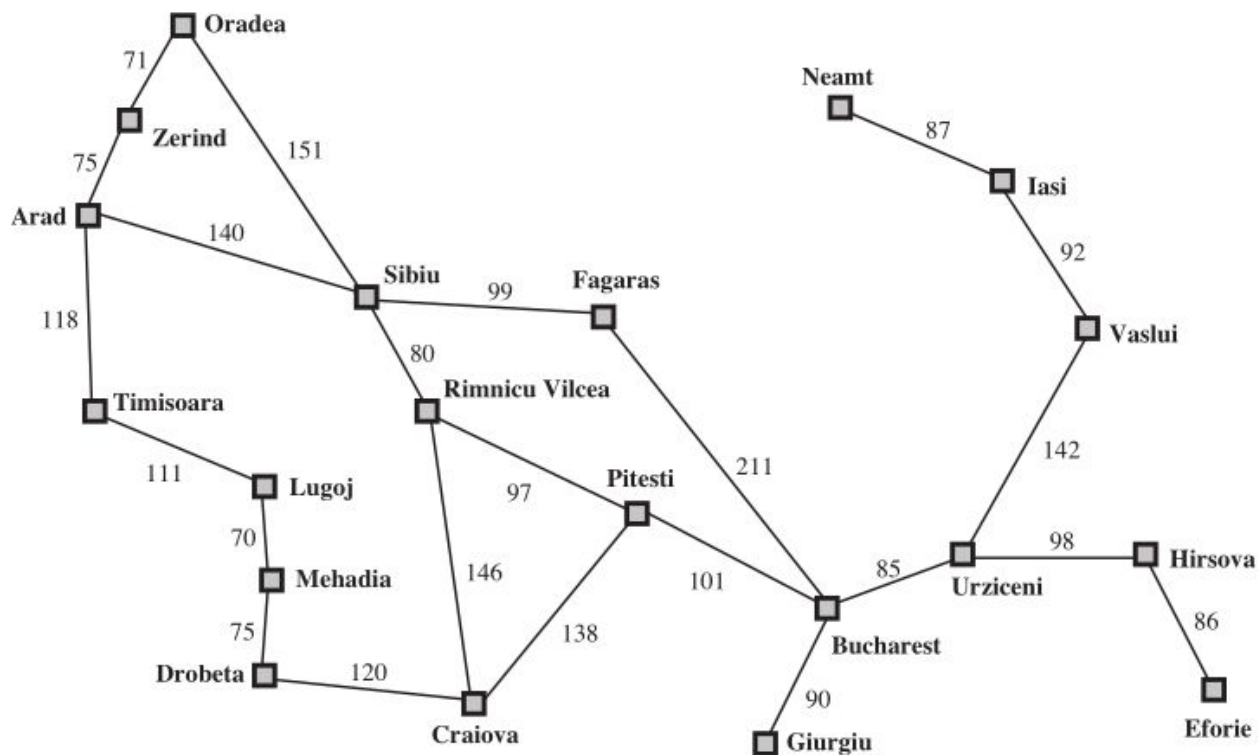
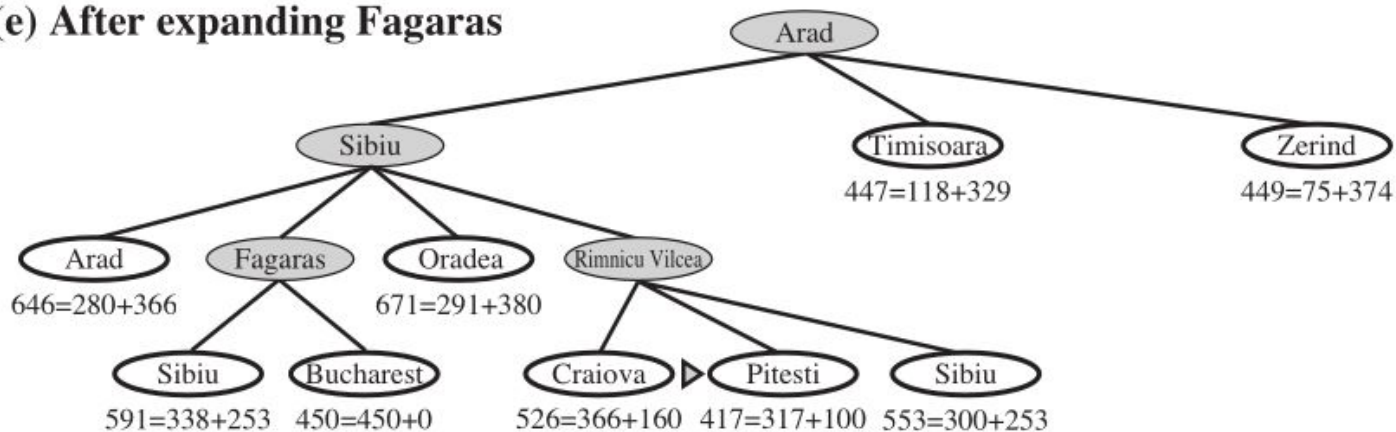
Stages in an A^* search

(d) After expanding Rimnicu Vilcea



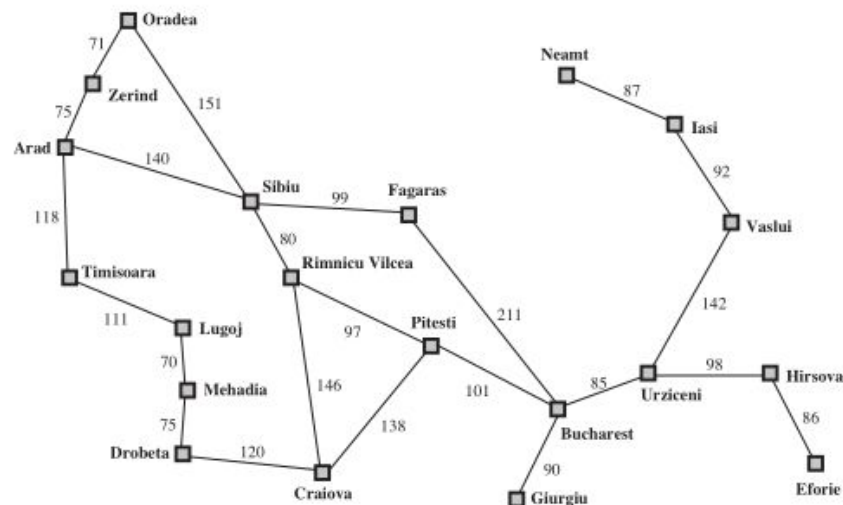
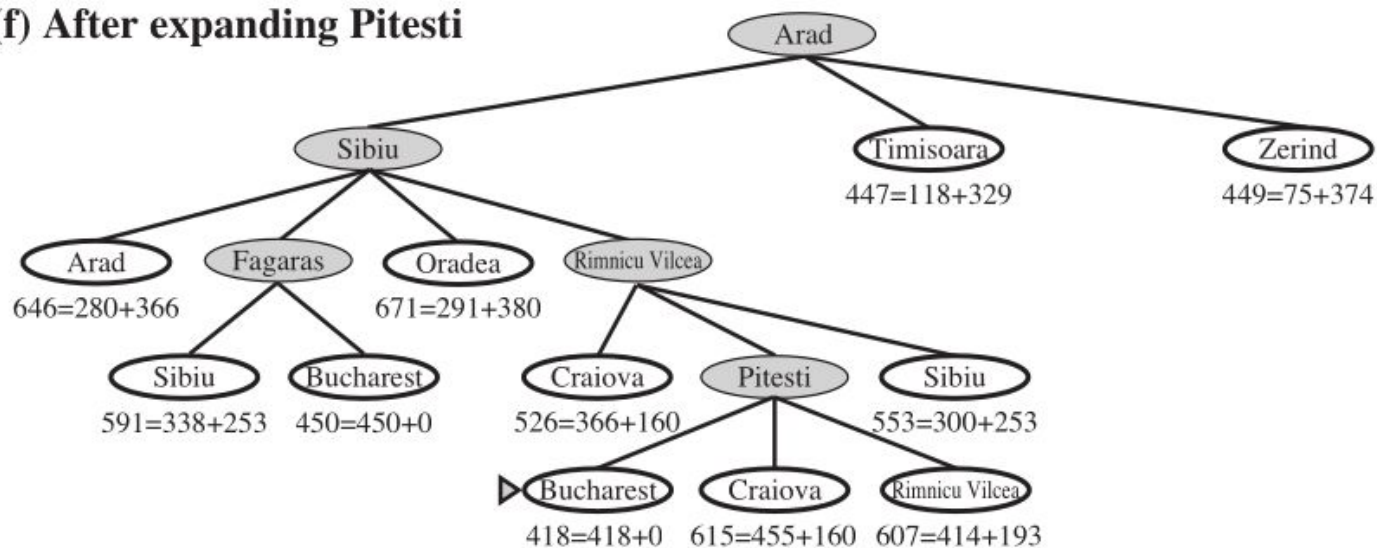
Stages in an A^* search

(e) After expanding Fagaras



Stages in an A^* search

(f) After expanding Pitesti



Optimality of A^*

Definition

A heuristic function h is said to be **admissible** if it *never overestimates* the cost to reach the goal, i.e.

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

where $h^*(n)$ is the true cost of the shortest path from n to the goal.

Theorem

If h is admissible then the tree search A^ is optimal.*

Optimality of A^*

The proof follows directly from the following lemma

Lemma

Assume h is admissible.

Let n be a node such that $n.State = goal$ and n is not on the optimal path to the goal,

Let n' be the first node on the optimal path that has not yet been expanded.

Then A^ expands n' before n .*

Optimality of A^*

The proof follows directly from the following lemma

Lemma

Assume h is admissible.

Let n be a node such that $n.State = goal$ and n is not on the optimal path to the goal,

Let n' be a node on the optimal path,

Then A^ expands n' before n .*

- It follows that A^* will always expand all the nodes on the optimal path before expanding any non-optimal node that contains the goal.
- Therefore, A^* is optimal.

Optimality of A^*

We now prove the lemma.

- Let us denote the cost of the optimal path by C^*
- When the non-optimal node n (that has $n.State = goal$) is created and added to the open-list, we compute its value $f(n)$.
- Since $n.State = goal$, then $f(n) = g(n)$ (real cost from root to n).
- The first optimal node n' that has not yet been expanded has a value

$$f(n') = g(n') + h(n') \leq C^* < g(n) = f(n).$$

- The first inequality follows from the admissibility of h ($C^* = g(n') + h^*(n')$ where $h(n') \leq h^*(n')$) and the second one follows from the definition of the minimum cost.
- Therefore, n' should be expanded before n .

Optimality of A^*

For graph search A^* , we need a stronger requirement on h because repeated states are not allowed.

Definition

A heuristic function h is said to be **consistent** if

$$\begin{aligned} \forall(n, a, n') : \quad & h(n) \leq c(n, a, n') + h(n'), \\ & \text{and } (h(n) = 0 \text{ if } n.State = goal), \end{aligned}$$

where $c(n, a, n')$ is the step cost for going from n to n' using action a .

Theorem

If h is consistent then the graph search A^ is optimal.*

Designing Heuristics

- For many problems other than route-finding, the procedure to come up with a good heuristic is not trivial.
- Let's reconsider the 8-puzzle problem.

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

Sliding-block puzzle

Two possible ideas for a heuristic are the following :

- h_1 : the number of tiles that has to be moved so that the current state looks like the goal state,
- h_2 : the sum of the Manhattan distances for all tiles between the current state and the goal one.

Designing Heuristics

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

Sliding-block puzzle

In our example :

- $h_1 = 8$.
- $h_2 = 18$.
- The true number of steps that it takes to reach the goal from this state is 26.
- Both heuristics are truly admissible and consistent.
- However, how can we evaluate which heuristic is the most appropriate to solve the problem with A^* ?
- How can we come up with heuristics in an automated way ?

Evaluating Heuristics

- We prefer heuristics that will make the A^* algorithm to expand as few nodes as possible.
- One way to evaluate the number of nodes expanded by a search technique is the effective branching factor b^* , defined as

$$N = 1 + b^* + (b^*)^2 + \dots + (b^*)^d,$$

where N is the number of nodes generated by A^* .

- e.g., if $d = 5$ and $N = 52$ then the branching factor is 1.92. This means that the algorithm on average branched out 1.92 times out of each node.
- The optimum and desired performance is for the effective branching factor to be 1.

Evaluating Heuristics

- If we experiment with the two possible heuristics for the 8-puzzle, we will discover that the second heuristic h_2 has on average a lower effective branching factor than h_1 .
- This is expected since $h_2(n) \geq h_1(n), \forall n$.
- We say that h_2 dominates h_1 .

Automatic Generation of Heuristics

- Notice that both h_1 and h_2 are optimal solutions to a simpler version of the original 8-puzzle.
- In general, one can find good admissible heuristics by **relaxing** the original problems (using fewer restrictions).

Automatic Generation of Heuristics Using Subproblems

One way to come up with relaxed versions of the original problem is to use subproblems that can be quickly solved.

*	2	4
*		*
*	3	1

Start State

	1	2
3	4	*
*	*	*

Goal State

Finding a heuristic for the 8-puzzle

One can also create a database of pre-computed solutions to different patterns (used a lot in chess games).

Combining Heuristics

- In many cases we can multiple valid heuristics and the question that arises is which one should we select to use.
- If we have a list of heuristics, $h_1, h_2, \dots, h_n$, and there is not one that dominates every other heuristic all of the time, then we can use at each state the heuristic that dominates the others in that specific state.
- We get a new heuristic $h_{max}(n) = \max\{h_1(n), h_2(n), \dots, h_n(n)\}$.
- If $h_1, h_2, \dots, h_n$ are admissible then h_{max} is also admissible.
- If $h_1, h_2, \dots, h_n$ are consistent then h_{max} is also consistent.
- h_{max} is a dominant heuristic over this set.

It happens that for the 8-puzzle the best heuristic is to use the maximum over multiple heuristics defined over subproblems.