Classical Search: Informed Search

Joe Johnson, M.S., Ph.D., A.S.A.

Informed Search Algorithms

- Informed Search Algorithms
  - Definition
    - An informed search is one in which that agent possesses more information than that offered in the problem statement, typically expressed in the form of an evaluation function, f(n) for each node n in the search space.

Informed Search Function

- Evaluation function, f(n):
  - Construed as a cost estimate.
  - The node with the lowest value for f(n) among those on the search frontier is the most desirable.
  - Typically incorporates a heuristic function, h(n), which estimates the cost to the nearest goal node.

Informed Search Function

- Heuristic Function
  - h(n) = estimated cost of the cheapest path from the state at node n to a goal state.
  - Note that h(n) depends not on the node itself but on the *state of the node*, only.

Best First Search

- Best First Search
  - Definition
    - Best first search is a general informed search algorithm in which the agent traverses the search space according to the following rule:
      - Select the next node for exploration from the frontier which has a *minimum value for an evaluation function, f(n)*.

Best First Search

- Implementation
  - Identical to that for uniform cost search.
    - Except we use this evaluation function, f(n)…
    - And not the path cost function, g(n).
    - Arrange nodes in priority queue in increasing order of evaluation function value, f(n).

Best First Search

- Special Cases of Best First Search
  - Greedy Best First Search
  - A* Search

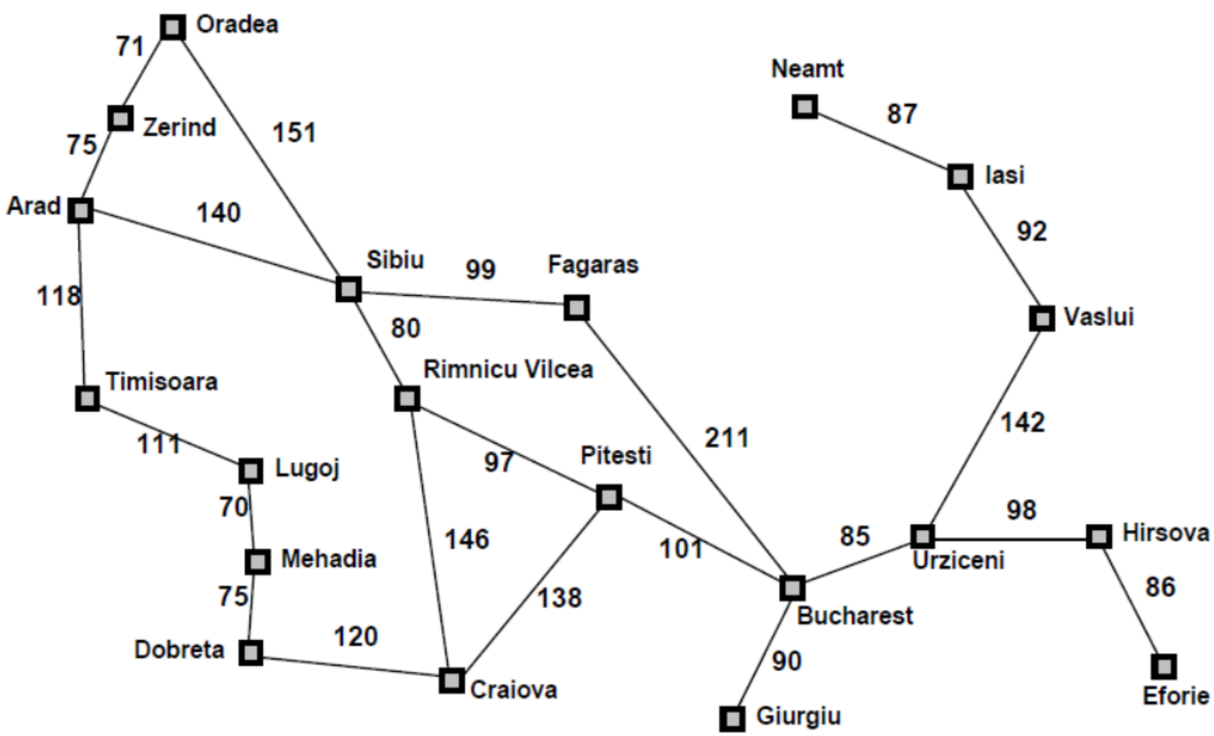
Greedy Best First Search

- Greedy Best First Search
  - Definition
    - A GBFS is a form of best first search in which f(n) = h(n)
      - Recall:
        - h(n) is the heuristic function estimating the cost of the cheapest path from the state of node n to the nearest goal node.
        - g(n), path cost function – the distance from the start node to the node, n, is not considered at all
  - Idea:
    - Expands the node that appears to be ***closest to the goal***, on the grounds that this is likely to lead to a solution quickly.

Greedy Best First Search Example – Arad to Bucharest

- Heuristic function, h(n) = straight line distance from node n to goal
- Denote h(n) as $h_{SLD}(n)$ = straight-line distance from city *n* to Bucharest
- Greedy best-first search expands the node that appears to be closest to goal

Heuristic Function Example
straight line distance heuristic



straight-line distances to Bucharest

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

Greedy search example

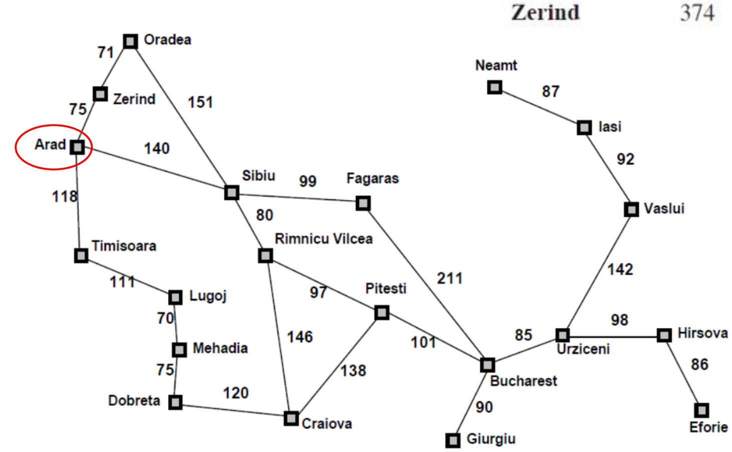
Node labels are h_{SLD} values

Arad is the initial state.

straight-line distances to Bucharest

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

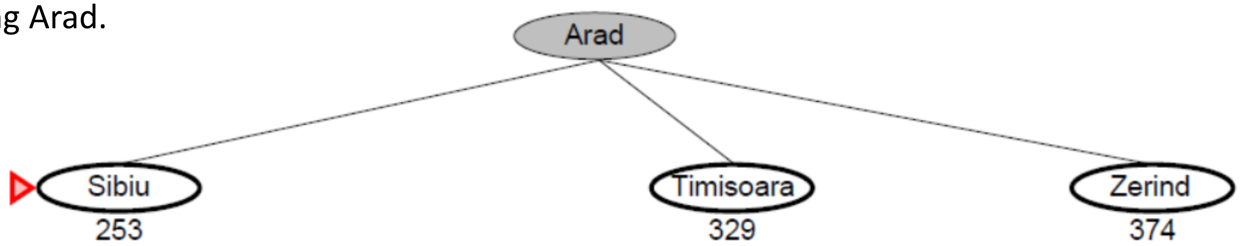
Frontier		Explored
Arad	366	



Greedy search example

Node labels are h_{SLD} values

After expanding Arad.



straight-line distances to Bucharest

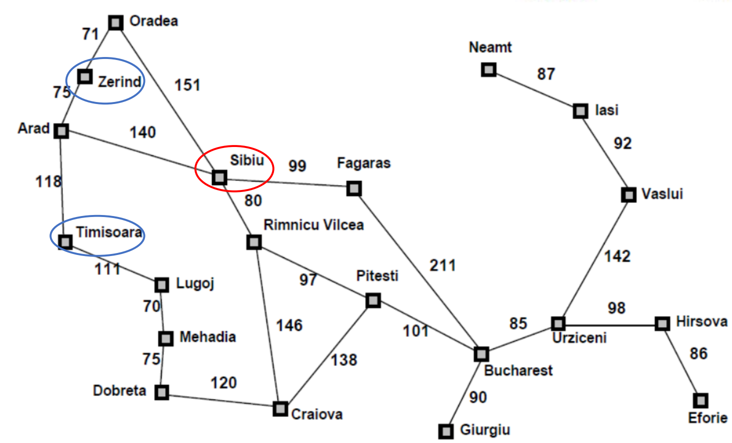
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Frontier

Sibiu	253
Timisoara	329
Zerind	374

Explored

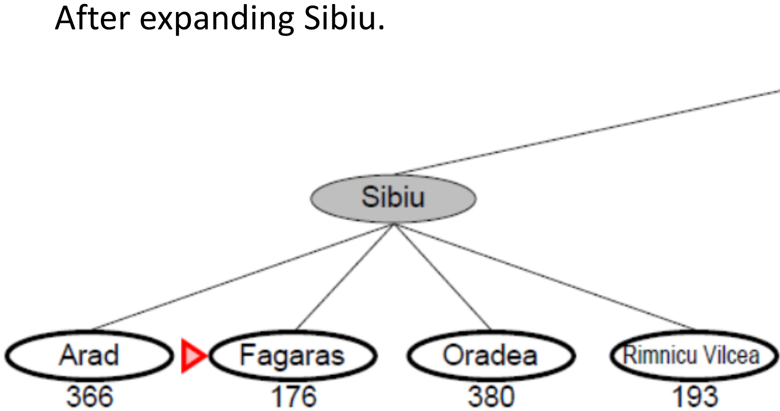
Arad



Greedy search example

Node labels are h_{SLD} values

After expanding Sibiu.



straight-line distances to Bucharest

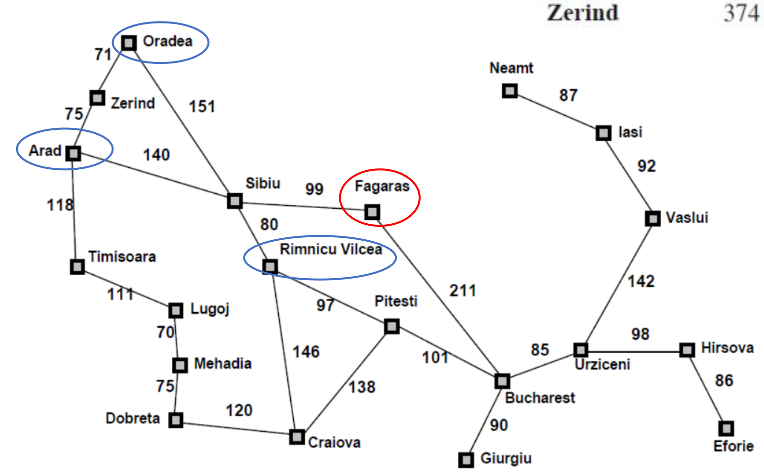
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Frontier

Fagaras	176
RimnicuVil	193
Timisoara	329
Zerind	374
Oradea	380

Explored

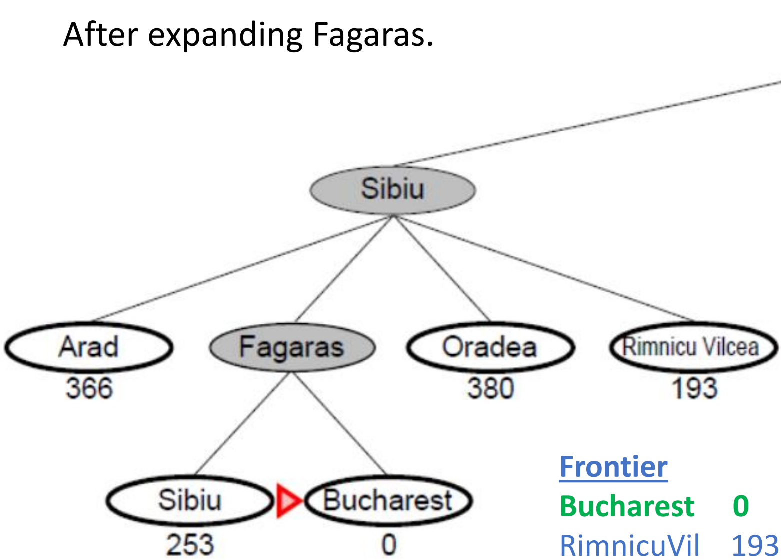
Arad
Sibiu



Greedy search example

Node labels are h_{SLD} values

After expanding Fagaras.



Frontier

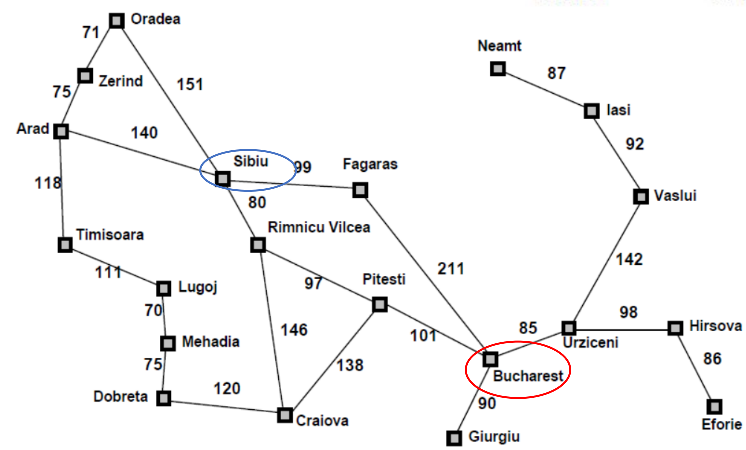
Bucharest	**0**
RimnicuVil	193
Timisoara	329
Zerind	374
Oradea	380

Explored

Arad
Sibiu
Fagaras

straight-line distances to Bucharest

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

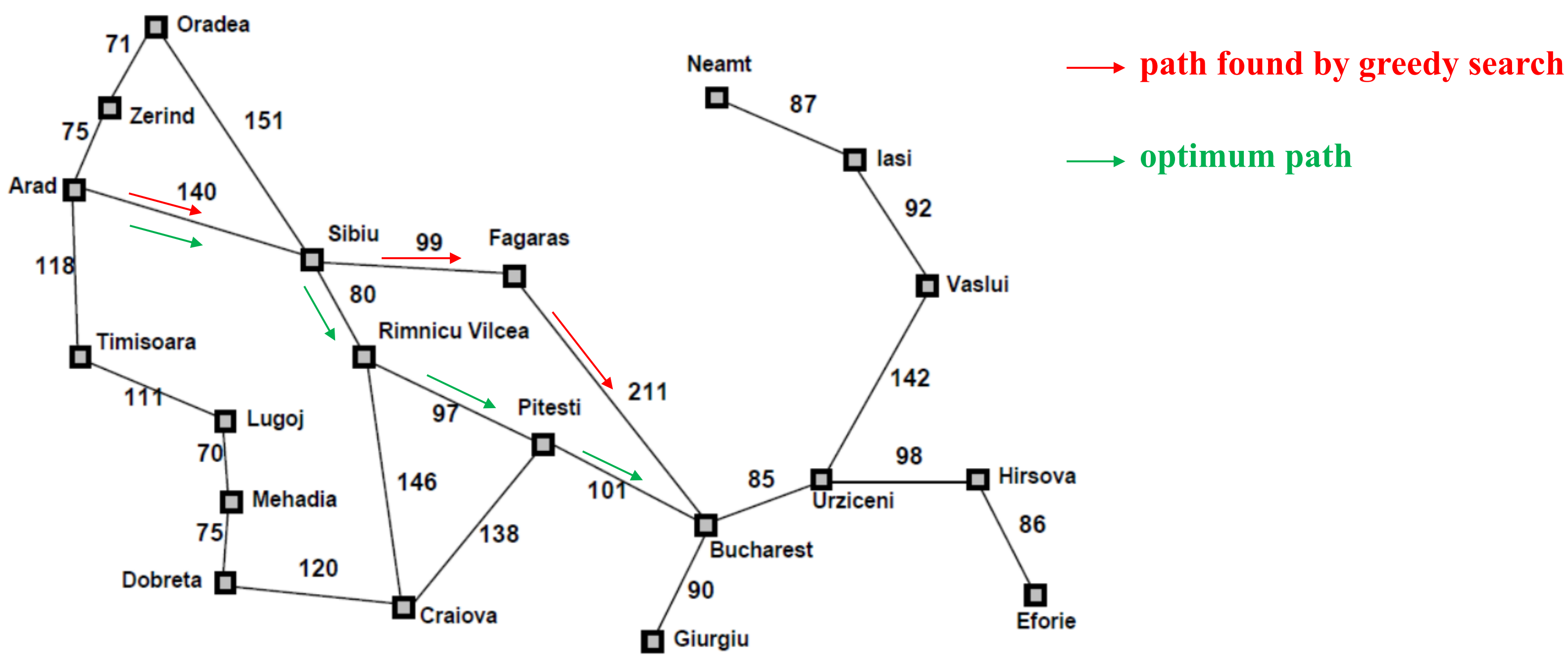


GBFS - Implementation

```
gbfs(Graph g = (V,E), Node start_node):
    start_node.predecessor = None                                      // start node has no predecessor
    frontier = []                                                      // set up priority queue for frontier, initially empty
    frontier.put(start_node):                                          // put start node in frontier

    explored_nodes = []                                                // set up list of explored nodes, initially empty

    while not frontier.empty():                                        // repeat while frontier not empty
      current_node = frontier.get().                                   // get node from front of frontier (priority queue)
      explored_nodes.add(current_node)                                 // add node to list of explored nodes
      if current_node.is_goal_state():                                 // is node a goal state?
        return path: start_node→neighbor_node.                         // if so, we are done – return path
      for neighbor_node in current_node.unexplored_adjacent_nodes():   // visit each unexplored neighbor
        neighbor_node.predecessor =  current_node                      // maintain a trail of bread crumbs
        frontier.put(neighbor_node, f(n))                              // put neighbor node in search frontier, f(n) score
        if frontier.count(neighbor_node) > 1:                          // keep only 1 copy of neighbor node with min f(n)
          keep neighbor_node with min(f(n)), remove others             // f(n) = h(n) is score for priority queue
    return None                                                        // if we made it here, no path to goal found
```

Properties of greedy search

Optimal? NO the path via Sibiu and Fagaras to Bucharest is 32 kilometers longer than the path through Rimnicu Vilcea and Pitesti.



Properties of greedy best-first search

- Complete
  - Complete in finite space with repeated state checking
- Optimal
  - No
- Time
  - $O(b^m)$, where m is the maximum depth of the search tree
  - But a good heuristic can give dramatic improvement
- Space
  - $O(b^m)$ - keeps all nodes in memory

A* Search

- *g(n)* = cost so far to reach *n*
- *h(n)* = estimated cost from *n* to goal
- *f(n)* = estimated total cost of path through *n* to goal

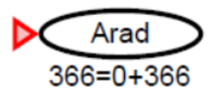
A* Search

- Definition – a form of best first search whose evaluation function is $f(n) = g(n) + h(n)$
- Idea – avoid expanding nodes that are already expensive
- Next node, n, selected from frontier is the one lying along the path from start to goal with minimum total path cost.

A* Search Example

Node labels are $f(n) = g(n) + h_{SLD}(n)$

Arad is the initial state.



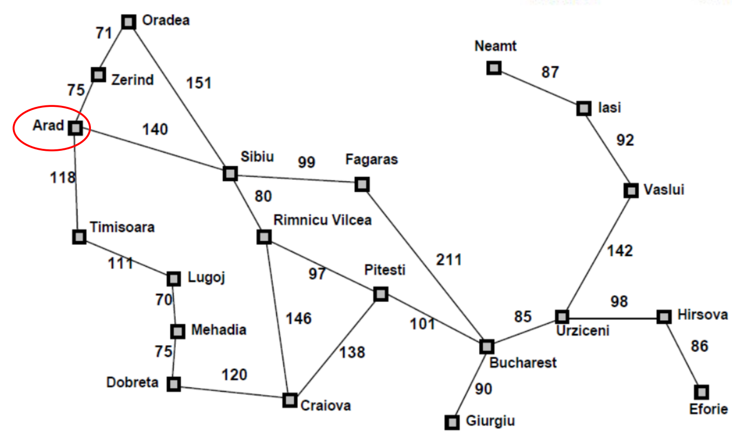
straight-line distances to Bucharest

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

Frontier

Arad 366

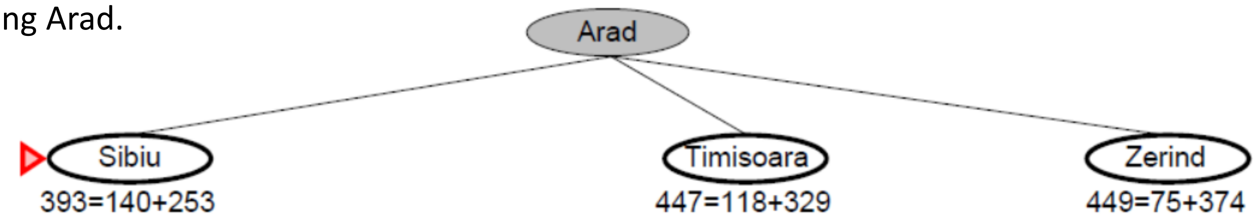
Explored



A* Search Example

Node labels are $f(n) = g(n) + h_{SLD}(n)$

After expanding Arad.



straight-line distances to Bucharest

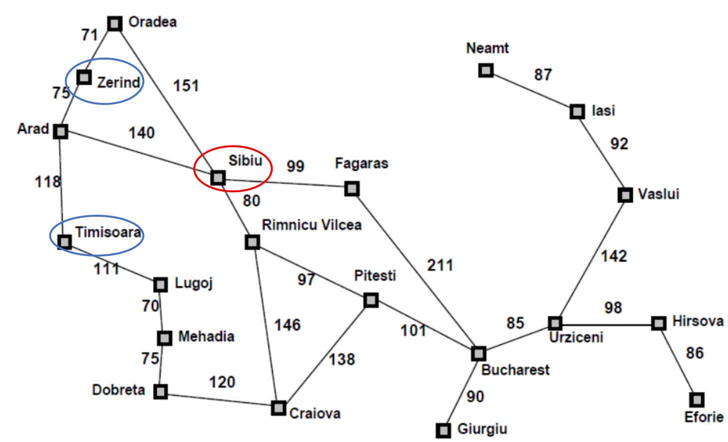
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Frontier

Sibiu	393
Timisoara	447
Zerind	449

Explored

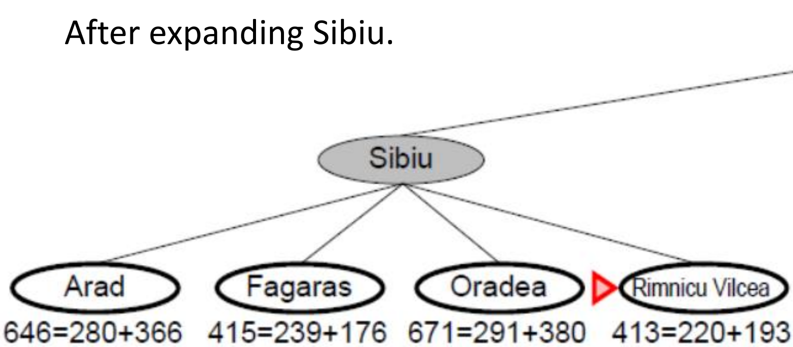
Arad



A* Search Example

Node labels are $f(n) = g(n) + h_{SLD}(n)$

After expanding Sibiu.



straight-line distances to Bucharest

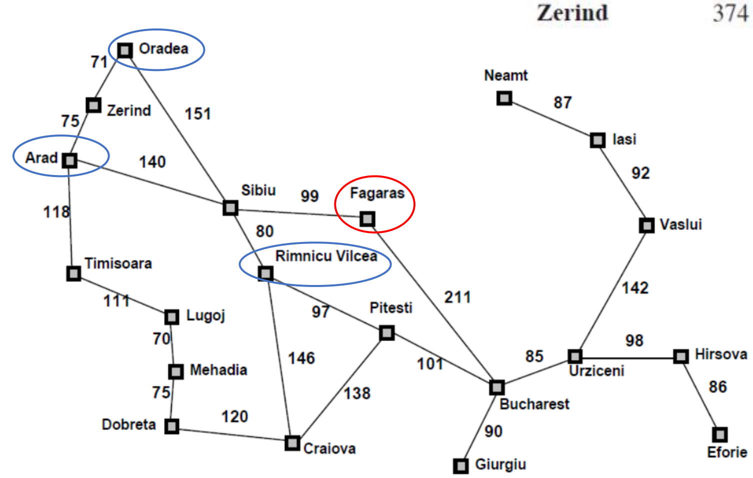
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Frontier

RimnicuVil	413
Fagaras	415
Timisoara	447
Zerind	449
Oradea	671

Explored

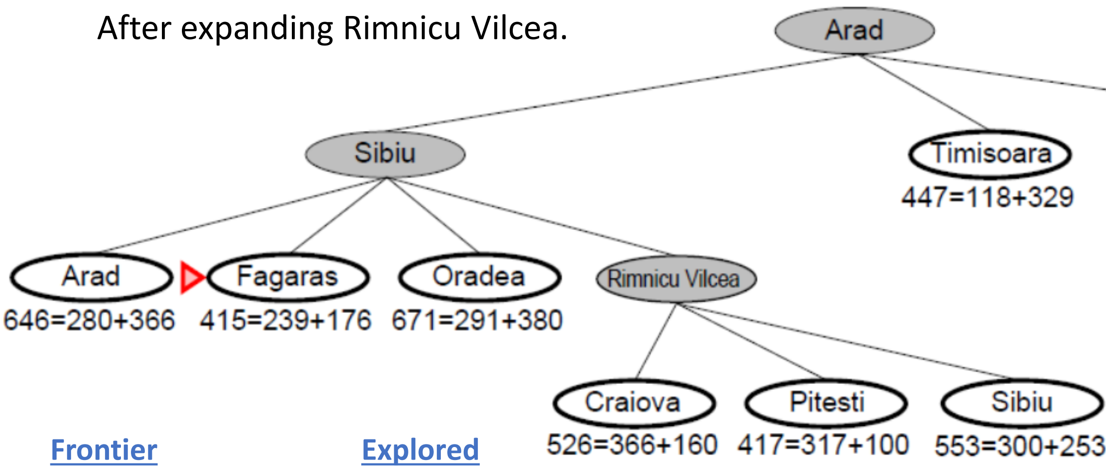
Arad
Sibiu



A* Search Example

Node labels are $f(n) = g(n) + h_{SLD}(n)$

After expanding Rimnicu Vilcea.



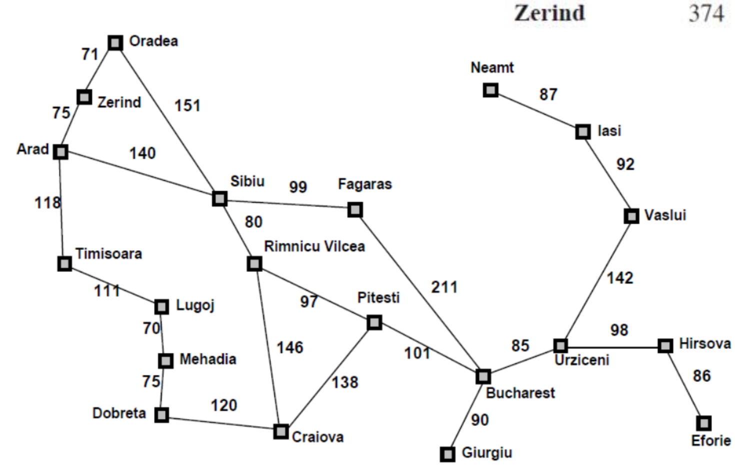
Frontier	
Fagaras	415
Pitesti	417
Timisoara	447
Zerind	449
Craiova	526
Oradea	671

Explored

Arad
Sibiu
RimnicuVil

straight-line distances to Bucharest

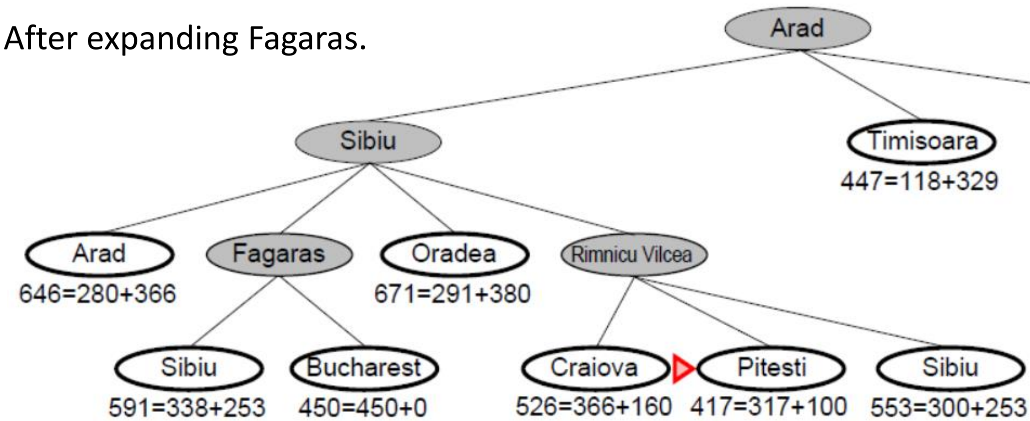
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



A* Search Example

Node labels are $f(n) = g(n) + h_{SLD}(n)$

After expanding Fagaras.



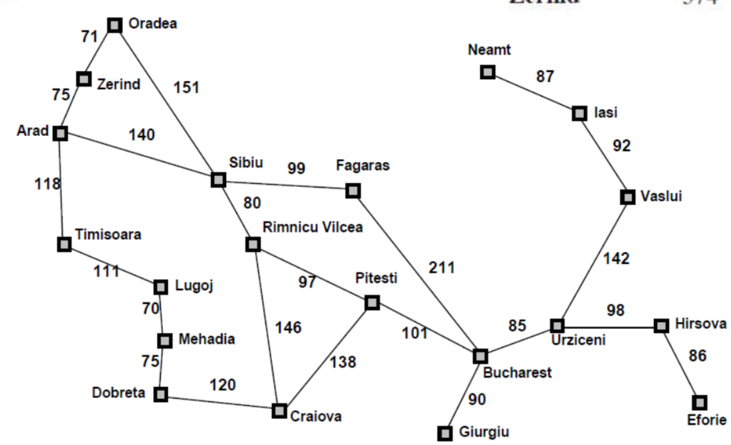
Frontier	
Pitesti	417
Timisoara	447
Zerind	449
Bucharest	450
Craiova	526
Oradea	671

Explored

Arad
Sibiu
RimnicuVil
Fagaras

straight-line distances to Bucharest

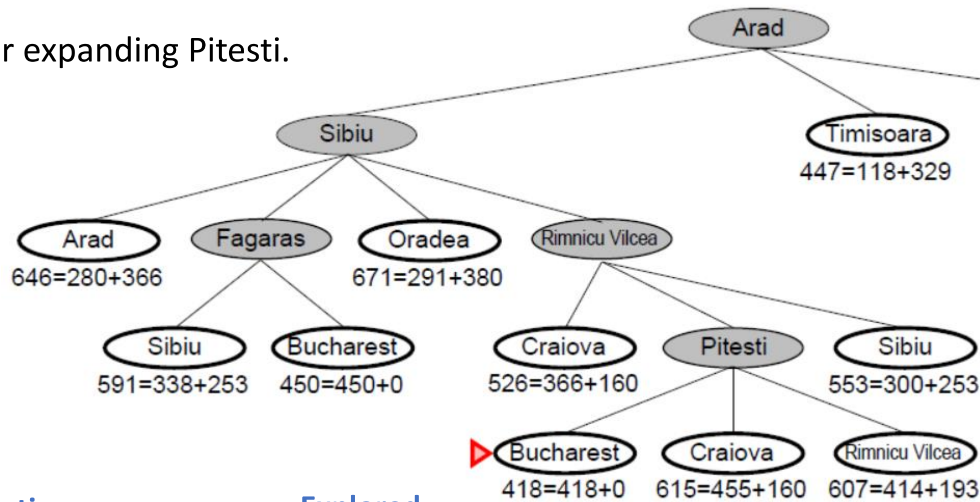
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



A* Search Example

Node labels are $f(n) = g(n) + h_{SLD}(n)$

After expanding Pitesti.



straight-line distances to Bucharest

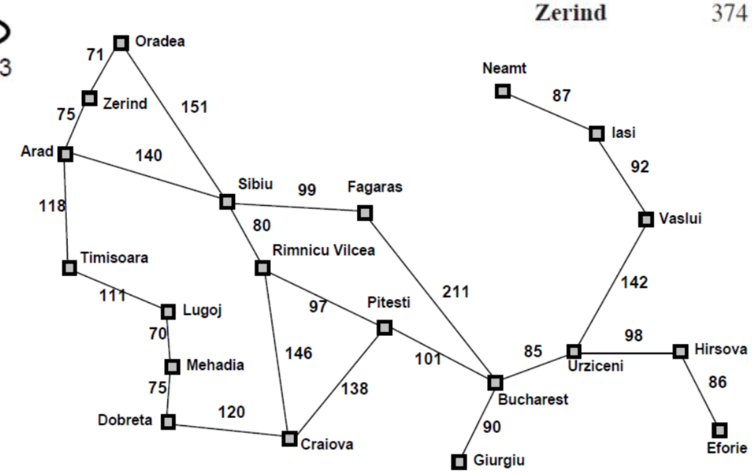
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Frontier

Bucharest	~~450~~ 418
Timisoara	447
Zerind	449
Craiova	526
Oradea	671

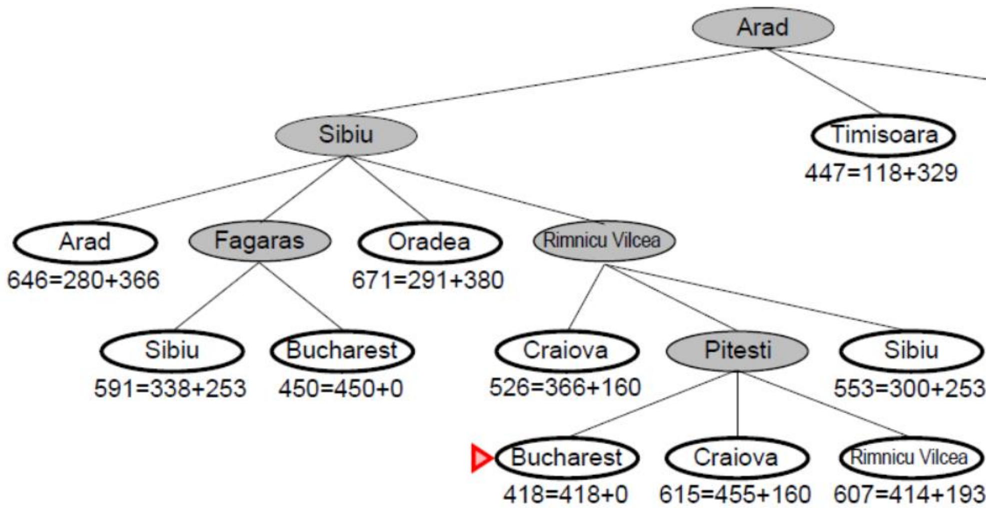
Explored

Arad
Sibiu
RimnicuVil
Fagaras
Pitesti



A* Search Example

Node labels are $f(n) = g(n) + h_{SLD}(n)$



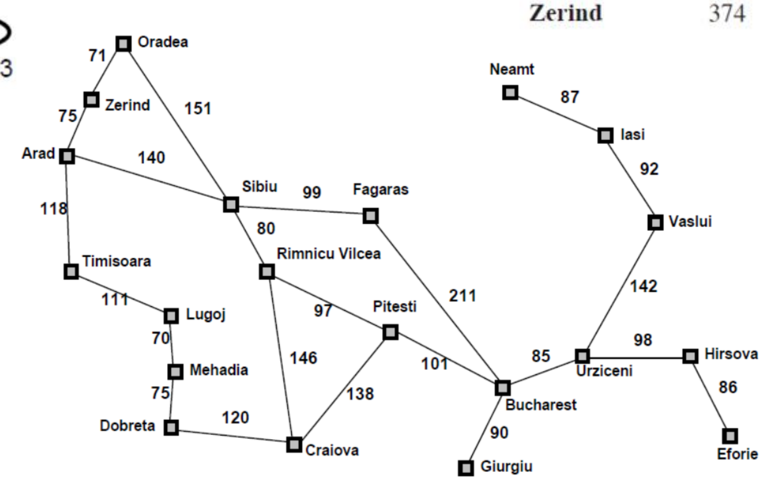
straight-line distances to Bucharest

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

Path found by A*: Arad, Sibiu, Rimnicu Vilcea, Pitesti, Bucharest
A* Path Cost: 140+80+97+101 = 418

Optimum Path Cost: 418

A* finds an optimum path.



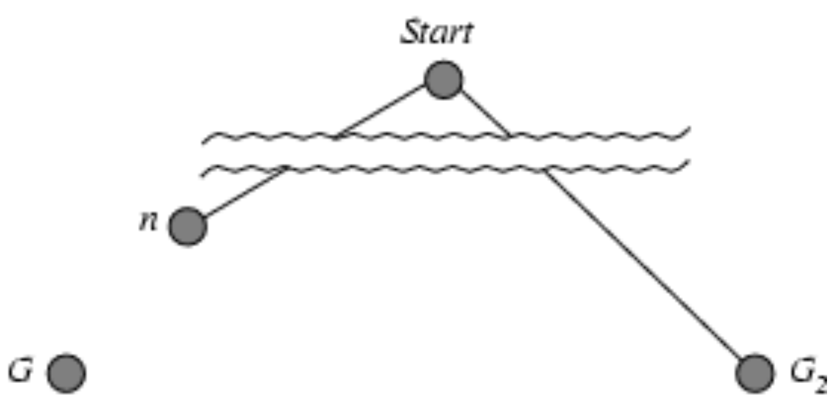
A* - Implementation

```
a_star(Graph g = (V,E), Node start_node):
    start_node.predecessor = None                          // start node has no predecessor
    frontier = []                                          // set up priority queue for frontier, initially empty
    frontier.put(start_node):                              // put start node in frontier

    explored_nodes = []                                    // set up list of explored nodes, initially empty

    while not frontier.empty():                            // repeat while frontier not empty
        current_node = frontier.get().                     // get node from front of frontier (priority queue)
        explored_nodes.add(current_node)                   // add node to list of explored nodes
        if current_node.is_goal_state():                   // is node a goal state?
            return path: start_node→neighbor_node.         // if so, we are done – return path
        for neighbor_node in current_node.unexplored_adjacent_nodes():   // visit each unexplored neighbor
            neighbor_node.predecessor =  current_node      // maintain a trail of bread crumbs
            frontier.put(neighbor_node, f(n))              // put neighbor node in search frontier, f(n) score
            if frontier.count(neighbor_node) > 1:          // keep only 1 copy of neighbor node with min f(n)
                keep neighbor_node with min(f(n)), remove others  // f(n) = g(n) + h(n) is score for priority queue
    return None                                            // if we made it here, no path to goal found
```

Condition for Optimality

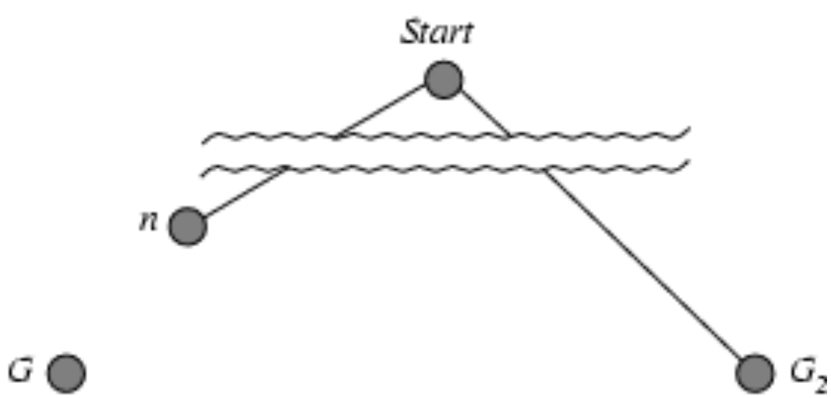
- Condition for Optimality:
  - Admissibility
    - *h(n) never over-estimates* the cost to reach the goal.
    - Since g(n) is the actual cost from the start node to node, n, f(n) = g(n) + h(n) never overestimates total path cost of the route through node n, as a consequence.
    - *h(n)* is considered optimistic
      - Example – straight line distance heuristic
    - Formally: *h(n)* <= *h*(n)* where *h*(n)* is the actual cost of the cheapest path from n to the nearest goal node.

Optimality of A* (proof)



- Suppose we have a search space with initial state given by the node, Start.
- Both G and G2 are goal nodes, but G is optimal and G2 is not optimal.
- Suppose we are at point in the search where both node, n, and G2 are unexpanded nodes in the frontier, where n is on the shortest path to G.
- We seek to show that **A* will select node n** for expansion **and not select G2** for expansion.
- That is, we need to show that $f(n) < f(G_2)$,
  - Recall A* chooses the next node, n_f, along the frontier such that $f(n_f)$ is the minimum value of f for all nodes in the frontier.

Optimality of A^* (proof – contd.)



- Proof Sketch:
- $f(G_2) = g(G_2)$ since $h(G_2) = 0$
- $g(G_2) > g(G)$ since G_2 is suboptimal
- $f(G) = g(G)$ since $h(G) = 0$
- $f(G_2) = g(G_2) > g(G) = f(G)$
- Thus, $f(G_2) > f(G)$, i.e., $f(G) < f(G_2)$

- $h(n) \leq h^*(n)$ since h is admissible
- $g(n) + h(n) \leq g(n) + h^*(n)$
- $f(n) = g(n) + h(n)$
- $f(G) = g(n) + h^*(n)$
- Thus, $f(n) \leq f(G)$, by substitution of $f(n)$ and $f(G)$ into our inequality

- Since $f(n) \leq f(G)$ and $f(G) < f(G_2)$ (from above), we have $f(n) < f(G_2)$
- Thus, A^* will select n for expansion and not G_2.

Properties of A*

- Complete
  - Yes (unless there are infinitely many nodes with $f \leq f(G)$)
- Optimal
  - Yes (unless there are infinitely many nodes with $f \leq f(G)$)
- Time
  - Exponential
- Space
  - $O(b^m)$Exponential - Keeps all nodes in memory