Week 4-5: Informed Search Strategies

Dr. Ammar Masood

Department of Cyber Security,

Air University Islamabad

Contents

Quick Review

- Previous algorithms differed in how to select next node for expansion eg:
  - Breadth First
    - Frontier nodes sorted old -> new
  - Depth First
    - Frontier nodes sorted new -> old
  - Uniform cost
    - Frontier nodes sorted by path cost: small -> big
- Used little (no) “external” domain knowledge

Informed Strategies

- Informed search strategies use Informed search domain-specific hints about the location of goals. This hint comes in the form of heuristic functions denoted h(n) to guide the search process towards the goal more efficiently than uninformed strategies. These heuristics provide additional domain-specific knowledge to estimate the best path to a solution.

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

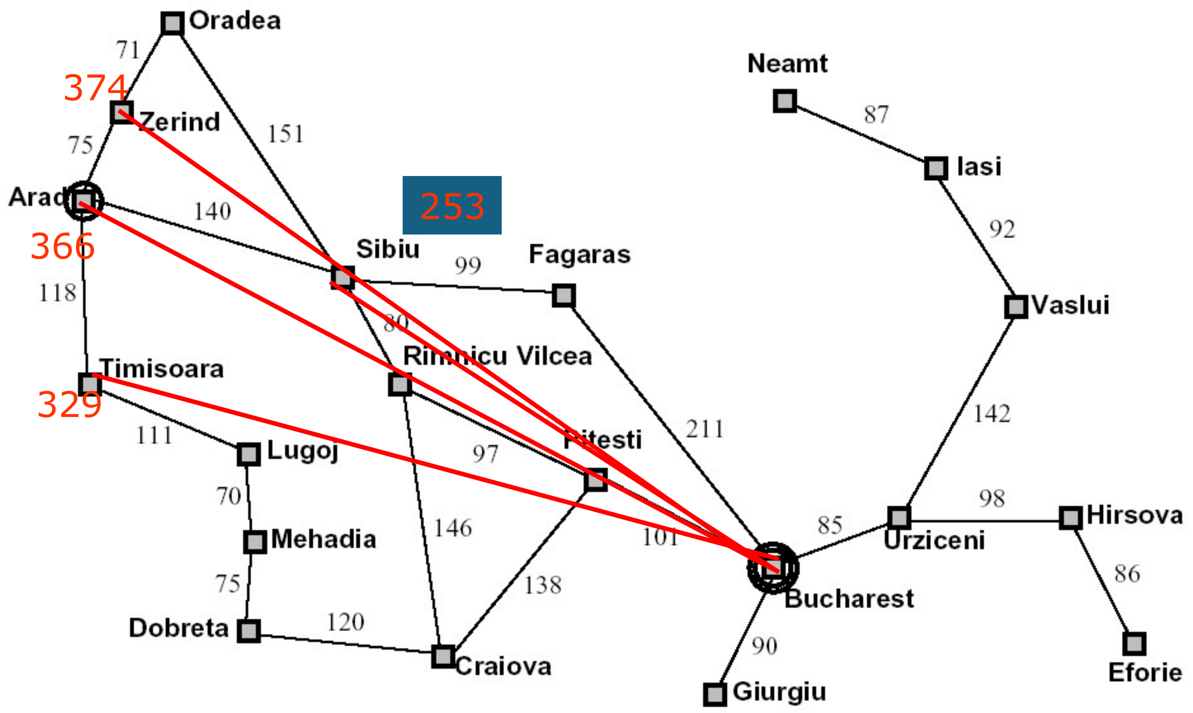
Greedy Best-First Search

- Greedy Best-First search tries to expand the node that is closest to the goal assuming it will lead to a solution quickly
  - $f(n) = h(n)$
  - aka "Greedy Search"

- Implementation
  - expand the "most desirable" node into the frontier queue
  - sort the queue in decreasing order of desirability

- Example: consider the **straight-line distance heuristic h_{SLD}**
  - Expand the node that appears to be closest to the goal
  - evaluation function $f(n) = h_{SLD}(n)$

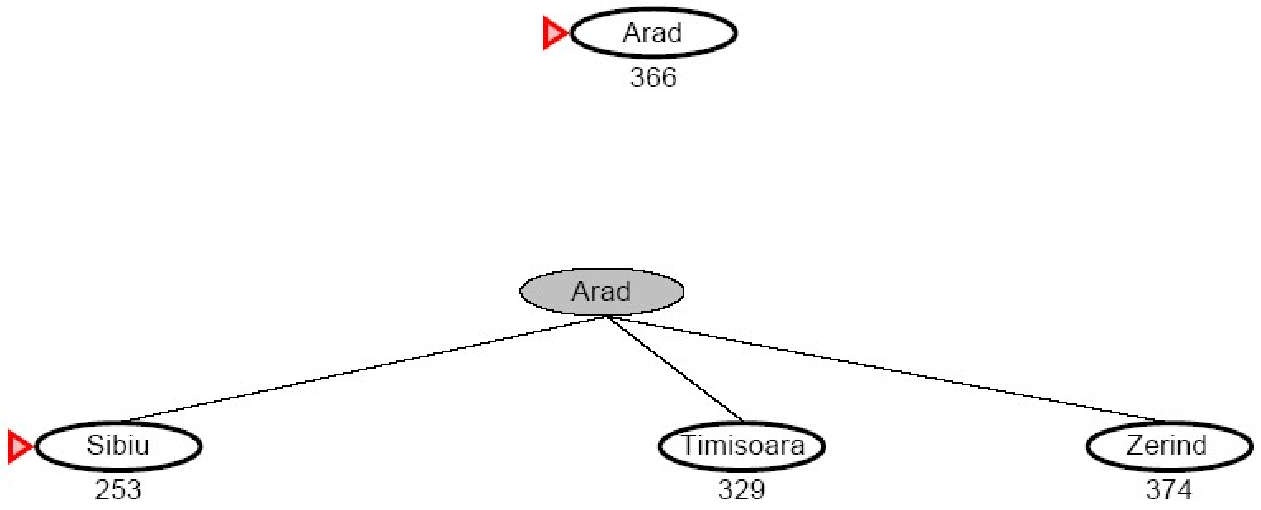
Route Finding



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

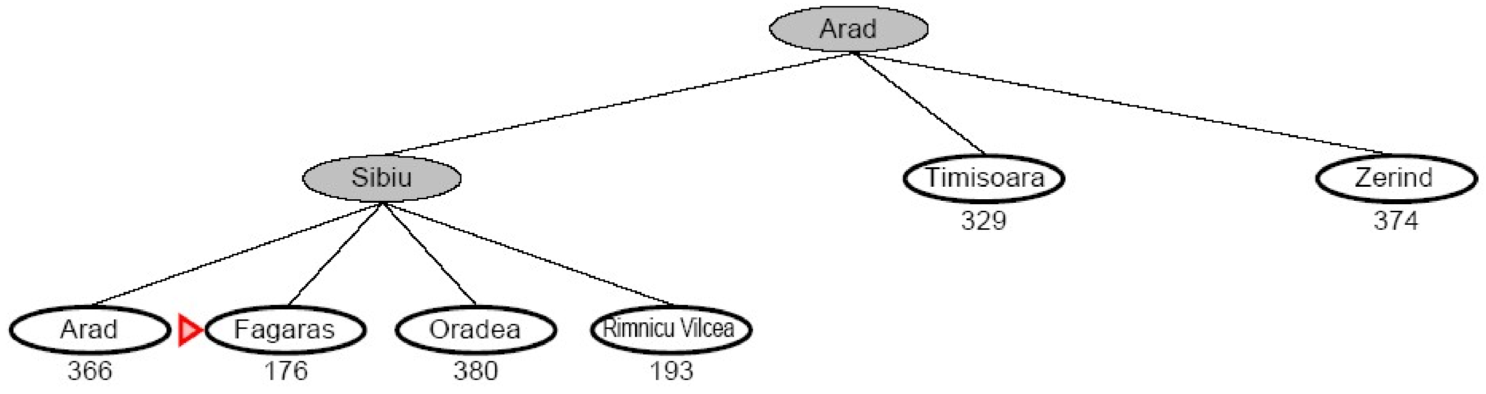
Greedy Best-First Search



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

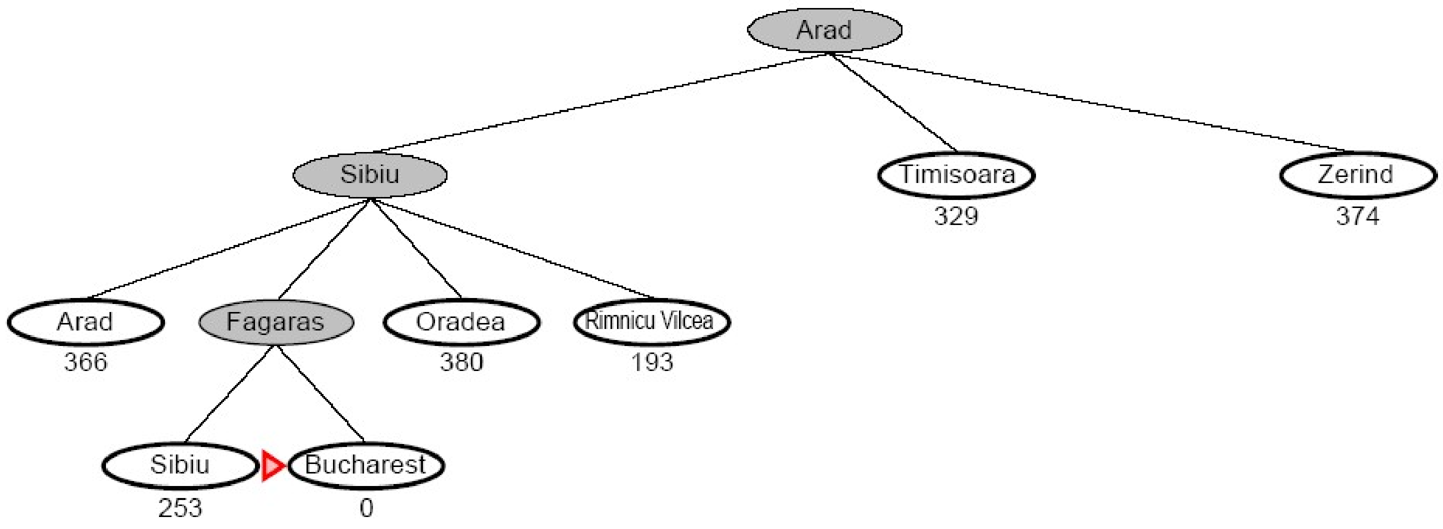
Greedy Best-First Search



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

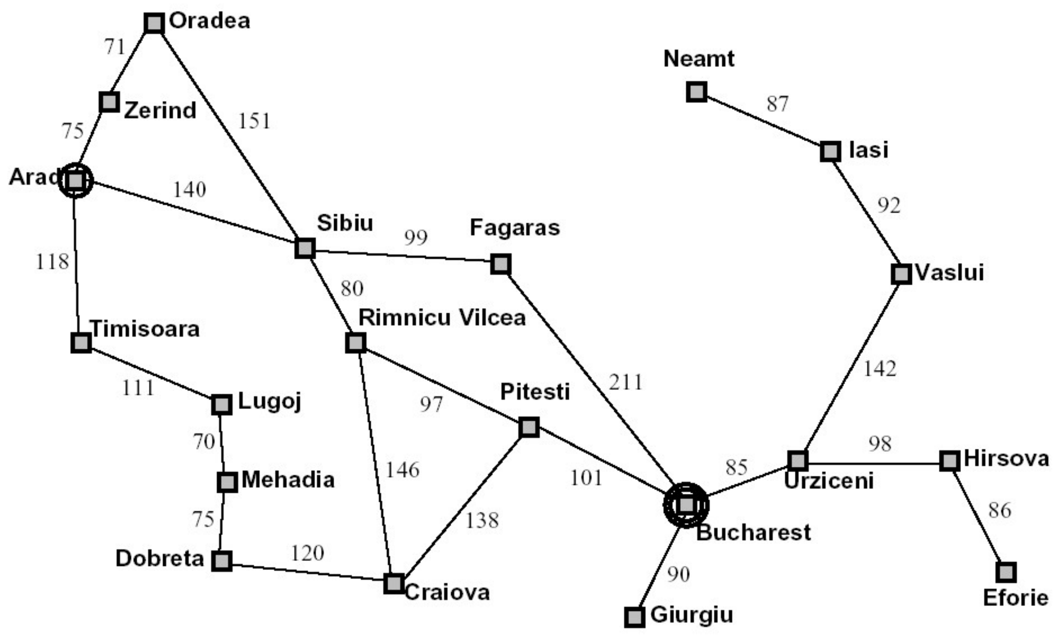
Greedy Best-First Search



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

Exercise



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

So is Arad->Sibiu->Fagaras->Bucharest optimal?

The path via Sibiu and Fagaras to Bucharest is 32 miles (450 miles) longer than the path through Rimnicu Vilcea and Pitesti (418 miles). This is why the algorithm is called “greedy”

Greedy Best-First Search

- Not optimal.
- Not complete.
  - Could go down a path and never return to try another.
  - e.g., Iasi → Neamt → Iasi → Neamt → …
- Space Complexity
  - $O(b^m)$ – keeps all nodes in memory
- Time Complexity
  - $O(b^m)$ (but a good heuristic can give a dramatic improvement)

A* search

A combination of Uniform-Cost Search and Greedy Best-First Search

Uses the evaluation function: $f(n)=g(n)+h(n)$

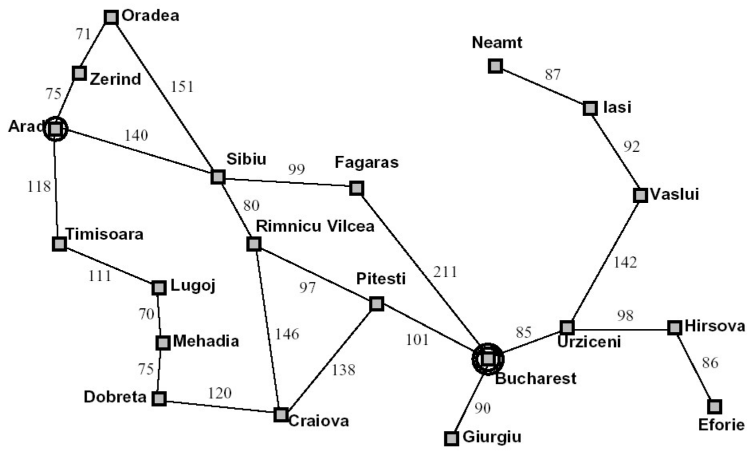
where:

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

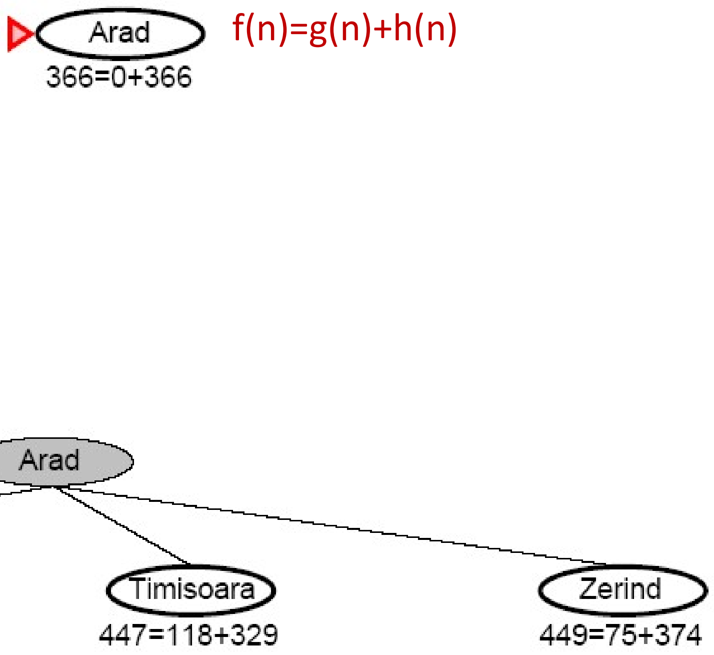
$h(n)$ is the estimated cost from n to the goal.

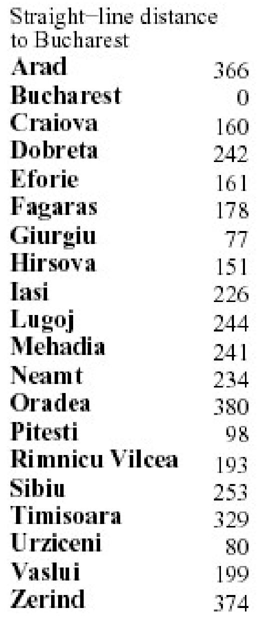
$f(n)$ = estimated cost of the best path that continues from n to a goal.

A* Search



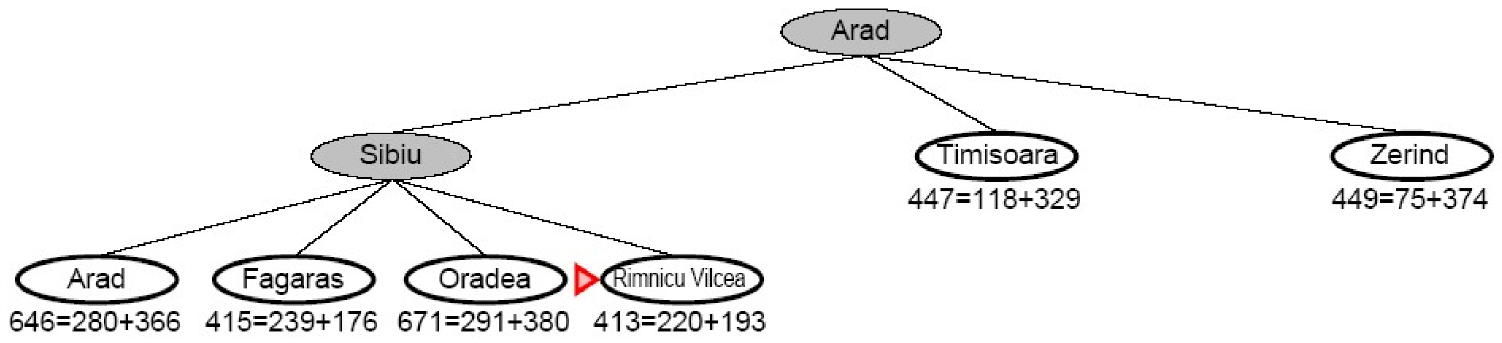
f(n)=g(n)+h(n)



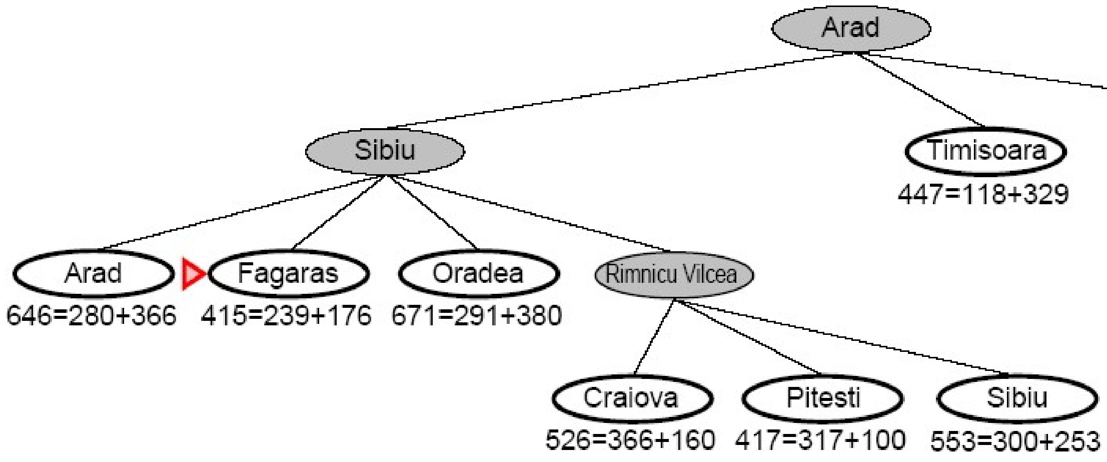


Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

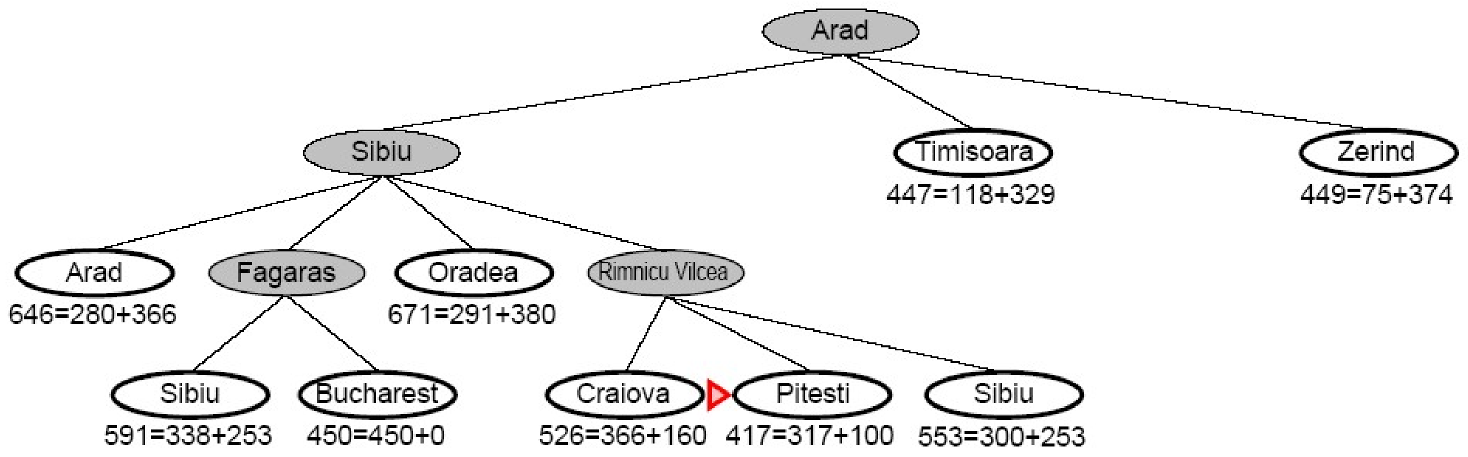
A* Search


Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
415=239+176
Oradea
671=291+380
Rimnicu Vilcea
413=220+193

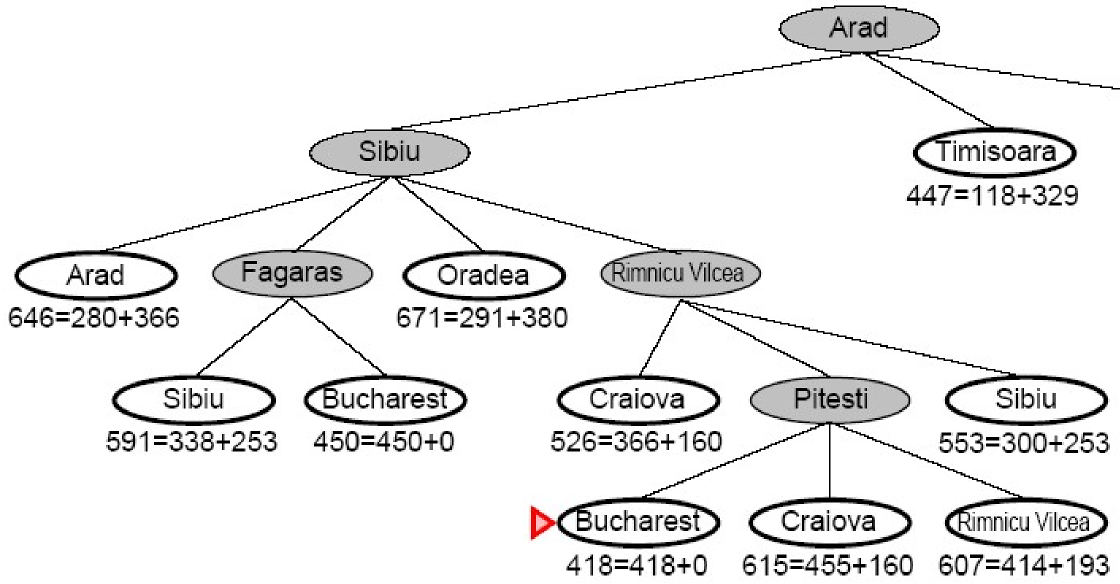
A* Search


Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
415=239+176
Oradea
671=291+380
Rimnicu Vilcea
Craiova
526=366+160
Pitesti
417=317+100
Sibiu
553=300+253

A* Search


Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
Oradea
671=291+380
Rimnicu Vilcea
Sibiu
591=338+253
Bucharest
450=450+0
Craiova
526=366+160
Pitesti
417=317+100
Sibiu
553=300+253

A* Search


Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
Oradea
671=291+380
Rimnicu Vilcea
Sibiu
591=338+253
Bucharest
450=450+0
Craiova
526=366+160
Pitesti
Sibiu
553=300+253
Bucharest
418=418+0
Craiova
615=455+160
Rimnicu Vilcea
607=414+193

A* Search

- When h(n) = actual cost to goal
  - Only nodes in the correct path are expanded
  - Optimal solution is found
- When h(n) < actual cost to goal ➔ Admissible Heurstic
  - Additional nodes are expanded
  - Optimal solution is found
- When h(n) > actual cost to goal
  - Optimal solution can be overlooked

A* Search

- A* is optimal if it uses an ***admissible heuristic***
  - h(n) <= h*(n) the true cost from node n
  - if h(n) *never overestimates* the cost to reach the goal

- Example
  - h_{SLD} never overestimates the actual road distance

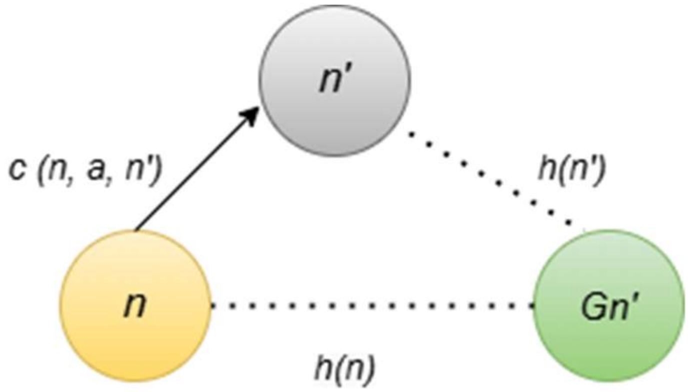
Consistency

- A slightly stronger property from admissibility is called consistency.
- A heuristic h(n) is consistent if, for every node and every successor of generated by an action we have:

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

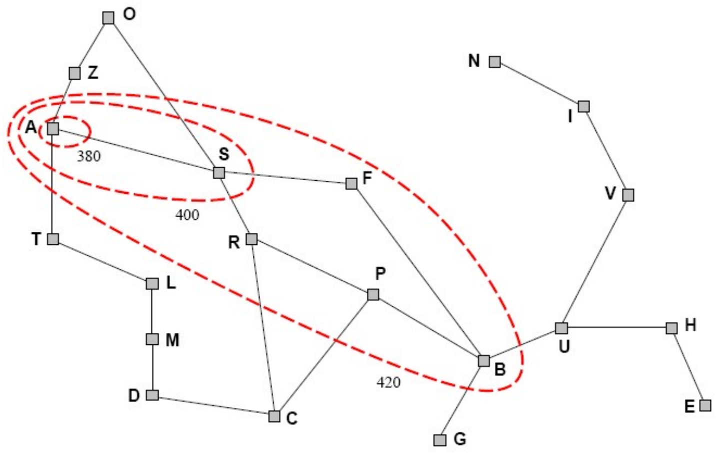
This is a form of the triangle inequality, which stipulates that a side of a triangle cannot be longer than the sum of the other two sides

An example of a consistent heuristic is the straight-line distance h_{SLD} that we used in getting to Bucharest.
Every consistent heuristic is admissible (but not vice versa), so with a consistent heuristic, A∗ is cost-optimal.



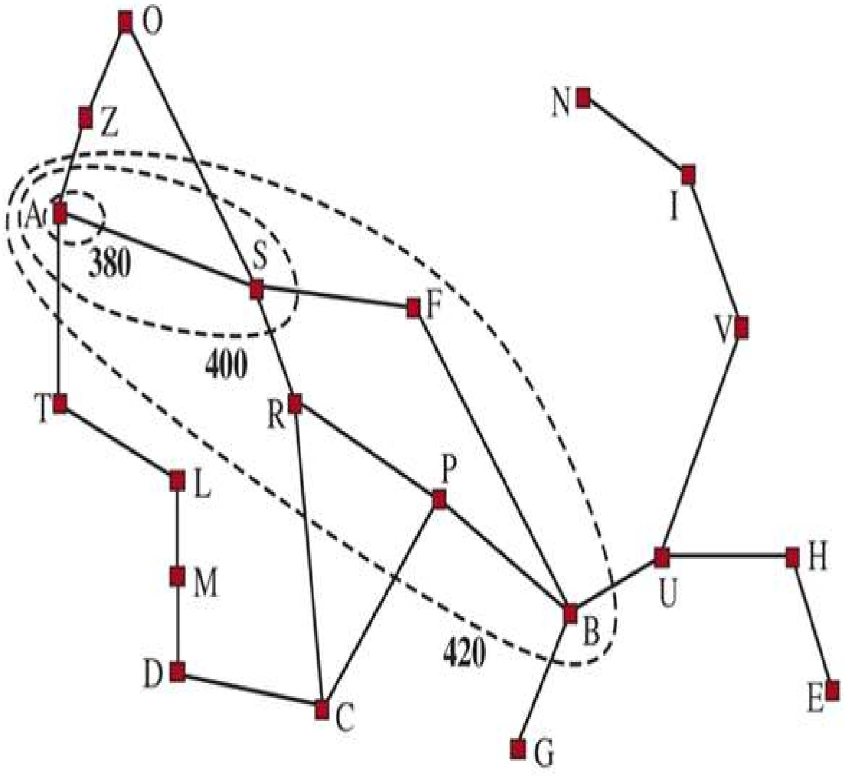
A* Search

- A* expands nodes in increasing f value
  - Gradually adds f-contours of nodes (like breadth-first search adding layers)
  - Contour i has all nodes $f=f_i$ where $f_i < f_{i+1}$



Contours

- A useful way to visualize a search is to draw contours in the state space, just like the contours in a topographic map.
- Inside the contour labeled 400, all nodes have f(n) = g(n)+h(n) ≤ 400, f and so on. because A* expands the frontier node of lowest –cost, we can see that an A* search fans out from the start node, adding nodes in concentric bands of increasing -cost.



Map of Romania showing contours at inside a given contour have f =g+h f =380, f = 400, and f =420, with Arad as the start state. Nodes costs less than or equal to the contour value.

Monotonic

- With uniform-cost search, we also have contours, but of cost g, not g +h.
- The contours with uniform-cost search will be “circular” around the start state, spreading out equally in all directions with no preference towards the goal.
- With A* search using a good heuristic, the g +h bands will stretch toward a goal state and become more narrowly focused around an optimal path.

It should be clear that as you extend a path, the costs are monotonic

Key points

- If C* is the cost of the optimal solution path, A* expands all nodes that can be reached from the initial state on a path where every node on the path has f(n) < C*. We say these are surely expanded nodes.
- A* might then expand some of the nodes right on the "goal contour" where f(n) = C* before selecting a goal node.
- A* expands no nodes with f(n) > C*

A* Search

- Complete
  - Yes, unless there are infinitely many nodes with f <= f(G)
- Time
  - The better the heuristic, the better the time
    - Best case h is perfect, O(d)
    - Worst case h = 0, $O(b^d)$ same as BFS
- Space
  - Keeps all nodes in memory and save in case of repetition
  - This is $O(b^d)$ or worse
  - A* usually runs out of space before it runs out of time
- Optimal
  - Yes, cannot expand f_{i+1} unless f_i is finished

Satisficing Search

Satisficing search aims to find a "good enough" solution rather than the optimal one. It trades off accuracy for efficiency, making it useful in scenarios where finding the absolute best solution is computationally expensive or unnecessary.

Inadmissible Heuristics

- An **inadmissible heuristic** is one that may **overestimate** the actual cost to reach the goal, meaning it does not guarantee optimal solutions.
- While admissible heuristics (like in A*) ensure optimality by never overestimating, inadmissible heuristics allow faster but potentially suboptimal solutions.
- **Example**: Overestimating straight-line distance between two points in measuring road distance to make search more aggressive.

Weighted A*

- **Weighted A*** is a variation of A* that introduces a weight factor to prioritize heuristic guidance more heavily.
- It modifies the standard A* evaluation function:

$$f(n)=g(n)+w\cdot h(n)$$

where:

g(n) is the cost from the start node to n.

h(n) is the heuristic estimate of the cost from n to the goal.

w (weight) is a tuning parameter ($w>1$) that increases heuristic influence.

Comparison

Search Strategy	**Cost Function**	**Weight (W)**
A* Search	$g(n)+h(n)$	$W = 1$
Uniform-Cost Search	$g(n)$	$W = 0$
Greedy Best-First Search	$h(n)$	$W = \infty$
Weighted A* Search	$g(n)+W\times h(n)$	$1 < W < \infty$

Memory-Bounded Heuristic Search

- Iterative Deepening A* (IDA*)
  - Similar to Iterative Deepening Search, but cut off at f(n)=g(n)+h(n)) > *max* instead of depth > *max*
  - At each iteration, cutoff is the first f-cost that exceeds the cost of the node at the previous iteration

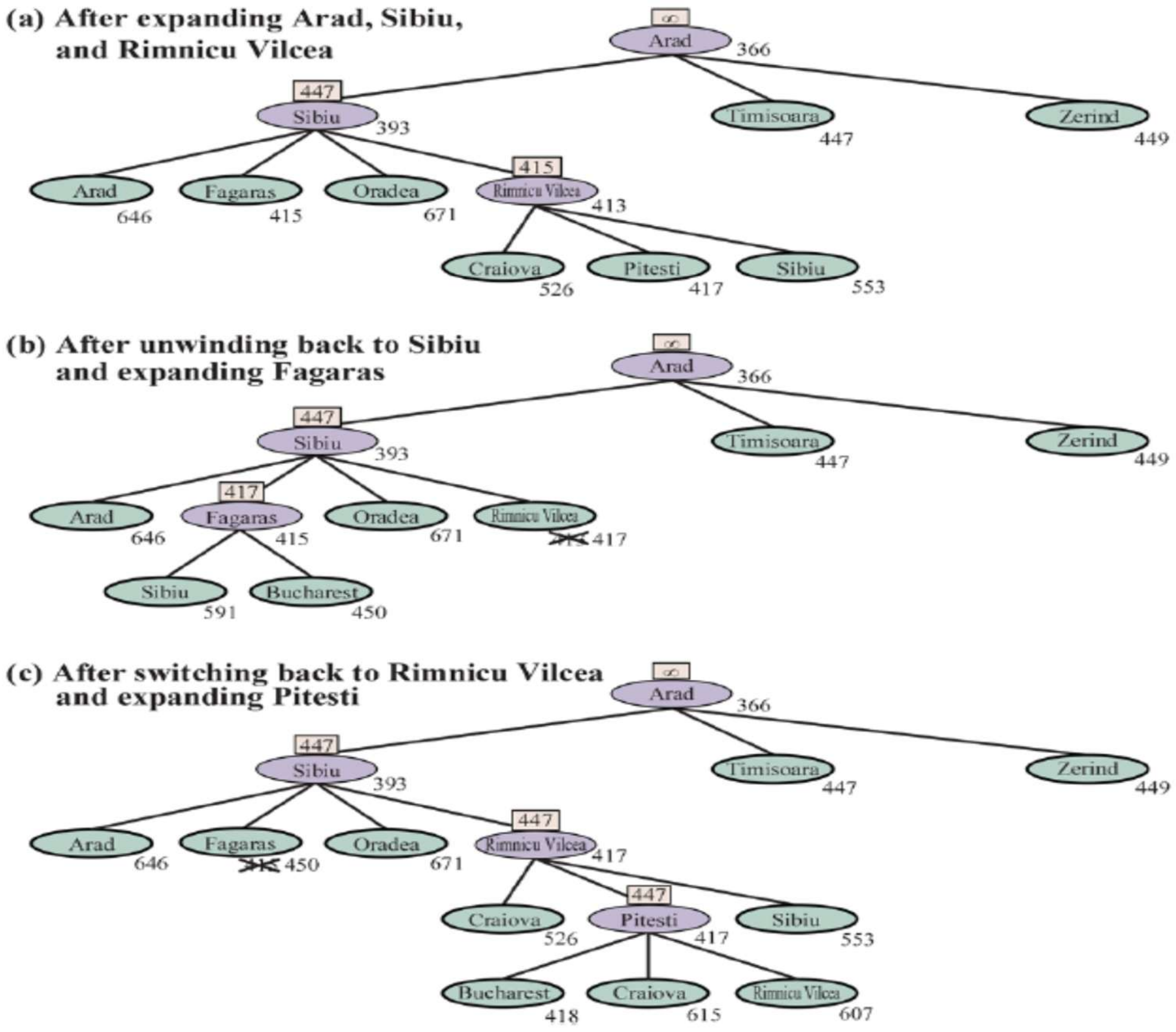
- Recursive Best First Search

- Simple Memory Bounded A* (SMA*)
  - Set max to some memory bound
  - If the memory is full, to add a node drop the worst f=g+h node that is already stored
  - Expands newest best leaf, deletes oldest worst leaf

Recursive Best First Search

- RBFS resembles a recursive depth first search
- Rather than continuing indefinitely down the current path, it uses the **f_limit** variable to keep track of the **f-value** of the best alternative path available from any ancestor
- If the current node exceeds this limit, the recursion unwinds back to the alternative path.
- As the recursion unwinds, RBFS replaces the **f-value** of each node along the path with a backed-up value—the best **f-value** of its children

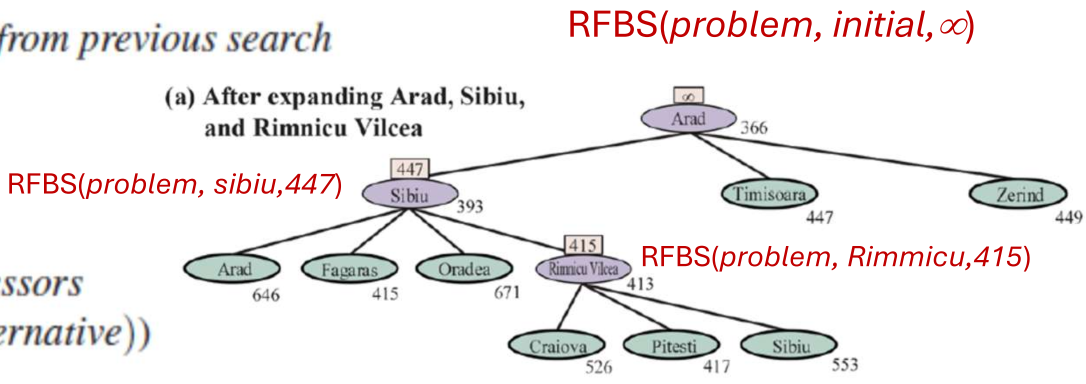
- Rather than continuing indefinitely down the current path, it uses the **f_limit** variable to keep track of the **f-value** of the best alternative path available from any ancestor
- If the current node exceeds this limit, the recursion unwinds back to the alternative path.
- As the recursion unwinds, RBFS replaces the **f-value** of each node along the path with a backed-up value—the best **f-value** of its children



Algorithm

```
function RECURSIVE-BEST-FIRST-SEARCH(problem) returns a solution or failure
    solution, fvalue ← RBFS(problem, NODE(problem.INITIAL), ∞)
  return solution

function RBFS(problem, node, f_limit) returns a solution or failure, and a new f-cost limit
   if problem.IS-GOAL(node.STATE) then return node
   successors ← LIST(EXPAND(node))
   if successors is empty then return failure, ∞
   for each s in  successors do        // update f with value from previous search
      s.f ← max(s.PATH-COST + h(s), node.f))
   while true do
      best ← the node in successors with lowest f-value
      if best.f > f_limit then return failure, best.f
      alternative ← the second-lowest f-value among successors
      result, best.f ← RBFS(problem, best, min(f_limit, alternative))
      if result ≠ failure then return result, best.f
```



(a) After expanding Arad, Sibiu, and Rimnicu Vilcea

Working

- Unlike A* search, which stores all explored nodes, RBFS only keeps track of a limited number of nodes using recursion.
- If a node exceeds current ***f_limit***, RBFS backtracks to explore second best path
- Requires less memory than A* as It uses backtracking when necessary

Applications

Search Algorithm	Classification	Real-World Application
Uniform Cost Search (UCS)	Uninformed	Finding the least costly path in transportation networks, such as determining the most fuel-efficient route for delivery trucks.
Bidirectional Search	Uninformed	Network routing protocols, where simultaneous searches from source and destination optimize data packet transmission.
Greedy Best-First Search	Informed	Web crawling and information retrieval by quickly locating relevant web pages based on keyword heuristics.
A* Search	Informed	Robotics navigation, enabling autonomous drones to determine the shortest and safest path to a target location using terrain data.
Weighted A* Search	Informed	Game development pathfinding, allowing non-player characters (NPCs) to find paths that balance speed and safety by adjusting the weight parameter.
Recursive Best-First Search	Informed	Solving complex puzzles and games, such as strategizing moves in chess where memory efficiency is crucial due to the vast search space.