

Artificial Intelligence

CS-401



Chapter # 03

Solving Problems by Searching

– Informed Searches

Dr. Hafeez Ur Rehman
(Email: hafeez.urrehman@nu.edu.pk)

Today's Outline

- Limitations of uninformed search methods
- **Informed (or heuristic)** search uses problem-specific **heuristics** to improve efficiency
 - Greedy Best-First Search
 - A* Search
 - RBFS (Recursive Best First Search)
 - SMA* (Simplified Memory Bounded A*)
- Can provide significant speed-ups in practice
 - e.g., on 8-puzzle
 - But can still have worst-case exponential time complexity

Limitations of uninformed search

- For 8-puzzle

- Avg. solution cost is about 22 steps

- branching factor ~ 3

- Exhaustive search to depth 22:

- **$3^{22} = 3.1 \times 10^{10}$ states**

- For example, at $d=12$, IDS expands 3.6 million states on average

- If we slightly complicate the problem to a 5x5 (24 puzzle) then the state space grows to **10^{24}** states. For such a problem, uninformed search strategies are the worst choice.

| | | |
|---|---|---|
| 7 | 2 | 4 |
| 5 | | 6 |
| 8 | 3 | 1 |

Start State

| | | |
|---|---|---|
| | 1 | 2 |
| 3 | 4 | 5 |
| 6 | 7 | 8 |

Goal State

Best-first search Class of algorithms

- Idea: use an **evaluation function** $f(n)$ for each node
 - estimate of "**desirability**"
 - Expand most desirable unexpanded node
- Implementation:
 - Order the nodes in fringe (queue) by $f(n)$ (i.e. by desirability, highest $f(n)$ first)
- **Special cases:**
 - 1. Uniform cost search** (from previous lecture): $f(n) = g(n)$
= path to n
 - 2. Greedy best-first search**
 - 3. A* search**
- Note: evaluation function $f(n)$ is an estimate of node quality

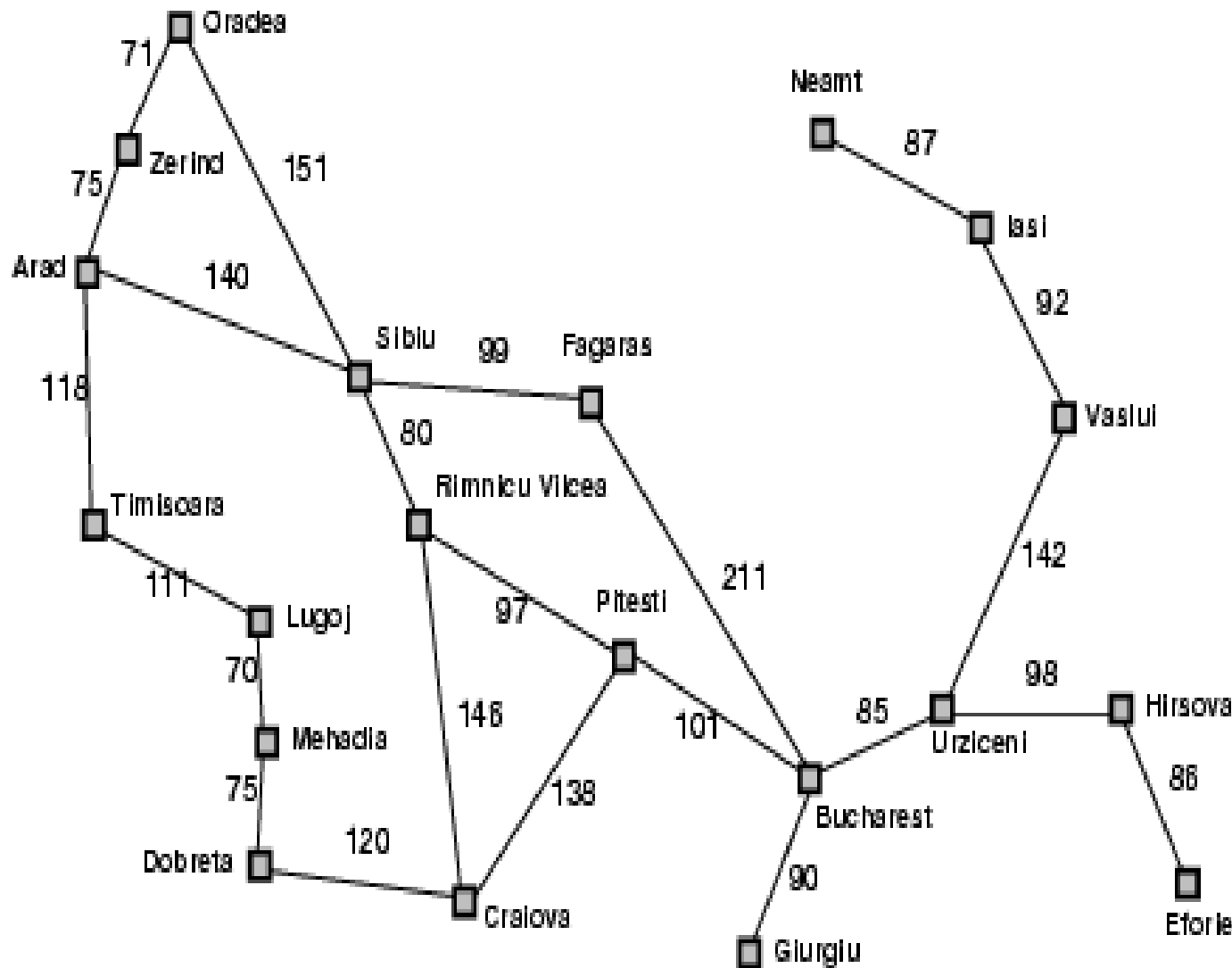
Heuristic function (for informed searches)

- Heuristic:
 - Definition: “using rules of thumb to find answers”.
These are clues in the form of additional information to help algorithm find solution quickly.
- Heuristic function $h(n)$
 - 1. Estimate of (optimal) cost from n to goal**
 2. $h(n) = 0$ if n is a goal node
 - 3. Example: straight line distance from n to Bucharest**
 - Note that this is not the true state-space distance
 - It is an estimate – actual state-space distance can be higher
 4. Provides problem-specific knowledge to the search algorithm

1. Greedy best-first search

- Special case of best-first search
 - Uses $h(n)$ = heuristic function as its evaluation function
 - Expand the node that appears closest to goal

Romania with step costs in km

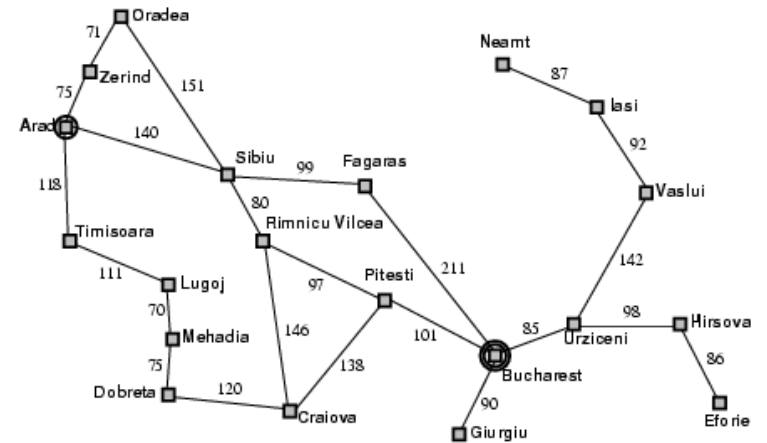


$h(n)$

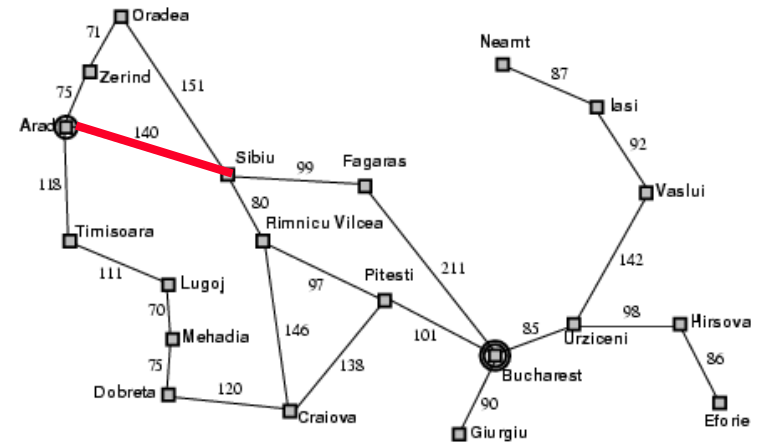
Straight-line distance
to Bucharest

| | |
|----------------|-----|
| Arad | 366 |
| Bucharest | 0 |
| Craiova | 160 |
| Dobreta | 242 |
| Eforie | 161 |
| Fagaras | 176 |
| Giurgiu | 77 |
| Hirsova | 151 |
| Iasi | 226 |
| Lugoj | 244 |
| Mehadia | 241 |
| Neamt | 234 |
| Oradea | 380 |
| Pitesti | 10 |
| Rimnicu Vilcea | 193 |
| Sibiu | 253 |
| Timisoara | 329 |
| Urziceni | 80 |
| Vaslui | 199 |
| Zerind | 374 |

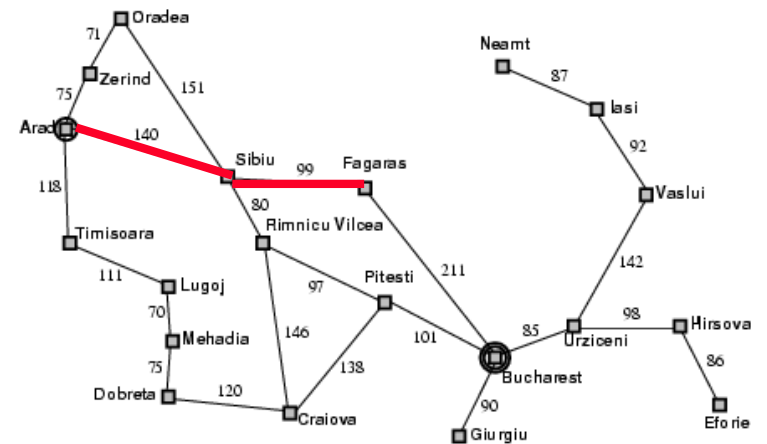
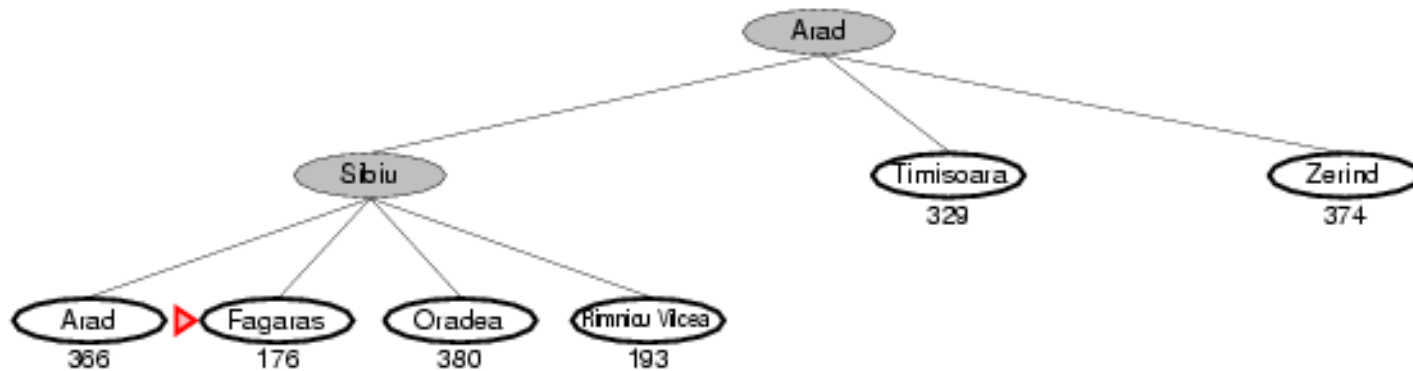
Greedy best-first search example



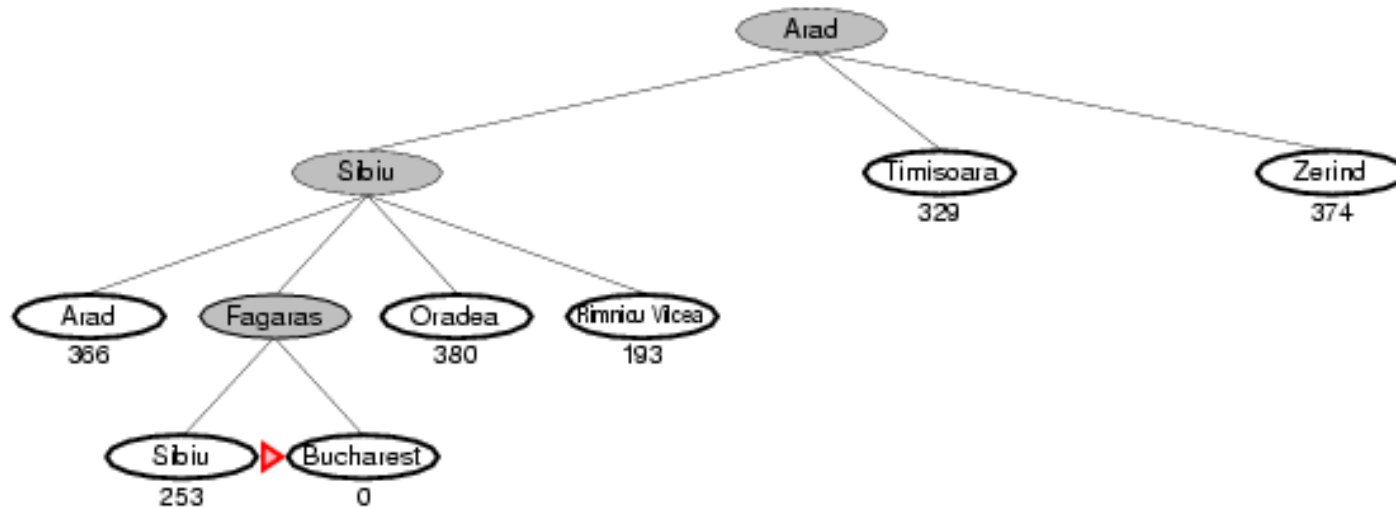
Greedy best-first search example



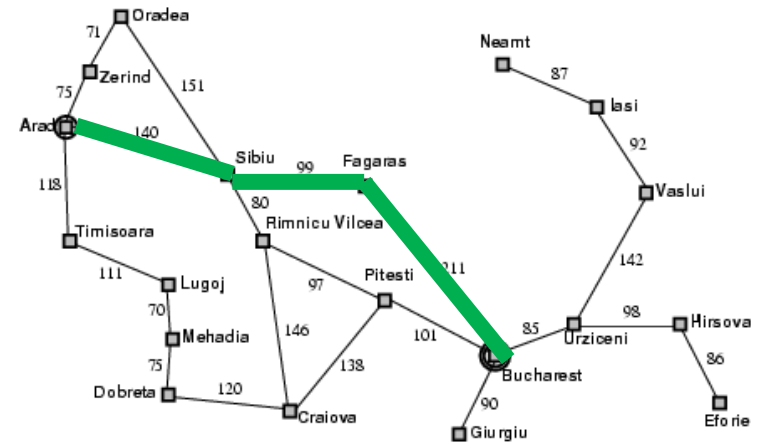
Greedy best-first search example



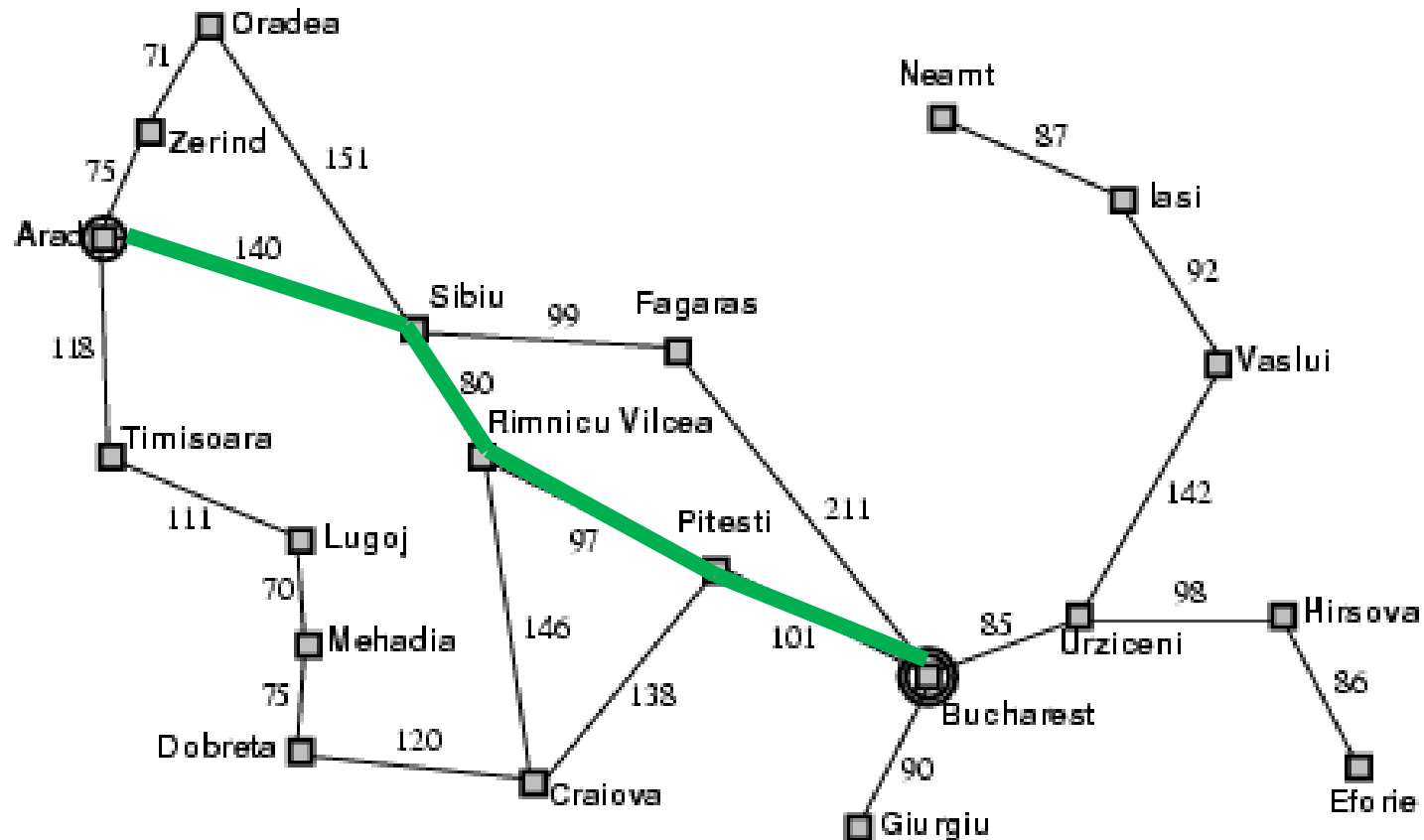
Greedy best-first search example



Attention: This is the non-optimal path



BUT Optimal Path



Conclusion: The Greedy Best First Search is non Optimal.

Completeness of Greedy BFS

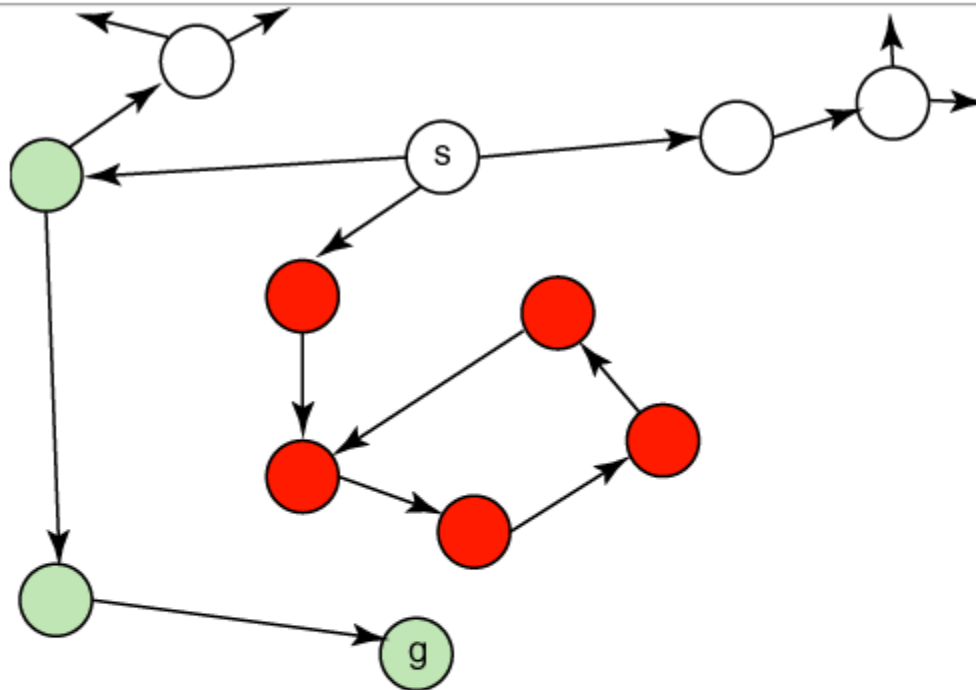


Figure 3.8: A graph that is bad for best-first search

Example 3.14: Consider the graph shown in [Figure 3.8](#), where the cost of an arc is its length. The aim is to find the shortest path from s to g . Suppose the Euclidean distance to the goal g is used as the heuristic function. A heuristic depth-first search will select the node below s and will never terminate. Similarly, because all of the nodes below s look good, a best-first search will cycle between them, never trying an alternate route from s .

Properties of greedy best-first search

- **Complete?**

- **No.** unless it keeps track of all states visited
 - Otherwise can get stuck in loops (just like DFS)

- **Optimal?**

- **No.** We just saw a counter-example (Romania map)

- **Time?**

- **Exponential** i.e., $O(b^m)$, can generate all nodes at depth m before finding solution
- Can be much better with a good heuristic

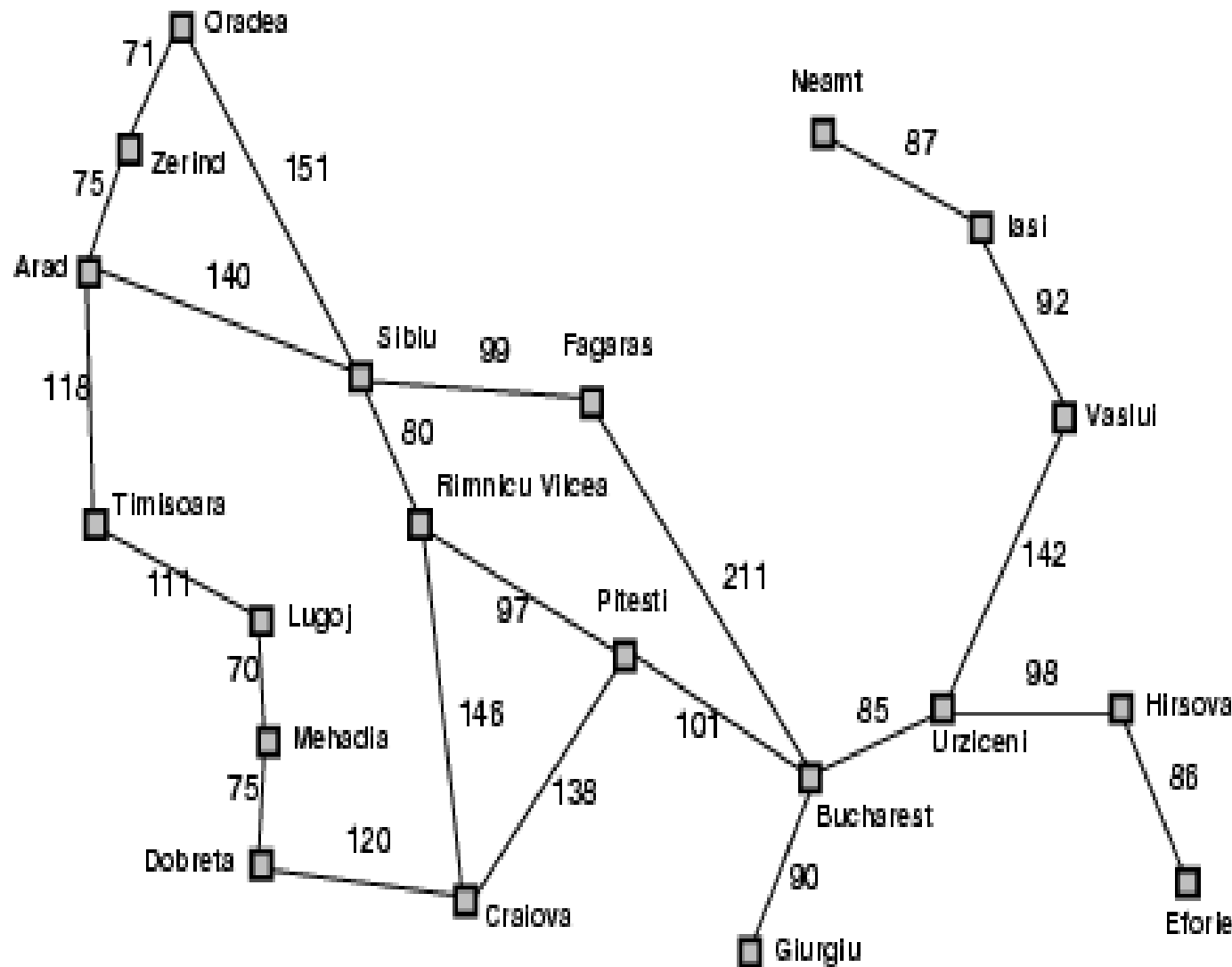
- **Space?**

- **Exponential** i.e., $O(b^m)$ – again, worst case, can generate all nodes at depth m before finding solution

2. A* Search

- Expand node based on estimate of **total path cost** through node
- Evaluation function $f(n) = g(n) + h(n)$
 - $g(n)$ = 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
- Efficiency of search will depend on **quality of heuristic $h(n)$**

A* search example [Arad to Bucharest]

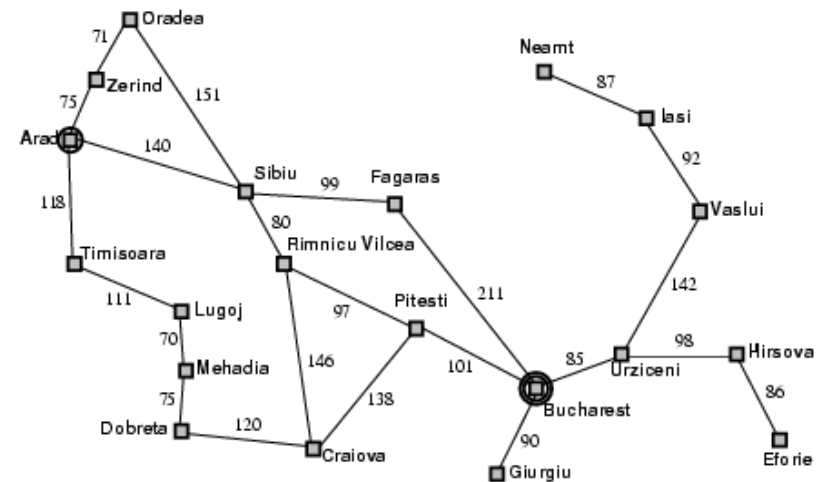
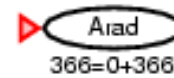


h(n)

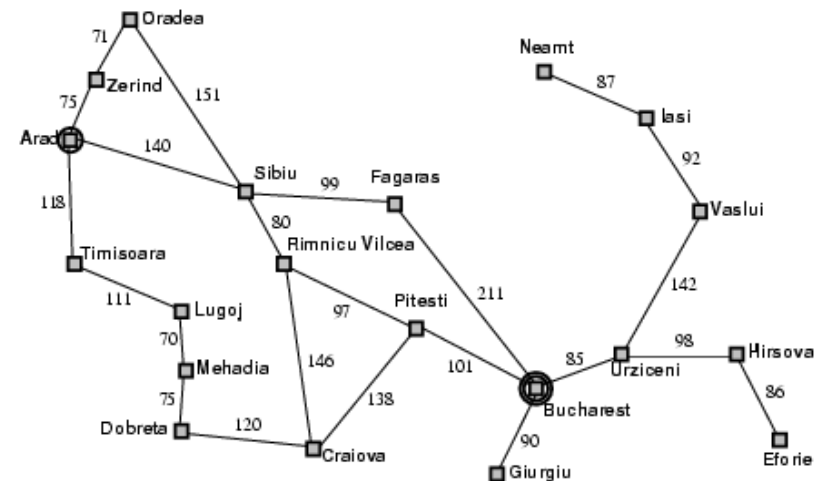
Straight-line distance
to Bucharest

| | |
|----------------|-----|
| Arad | 366 |
| Bucharest | 0 |
| Craiova | 160 |
| Dobreta | 242 |
| Eforie | 161 |
| Fagaras | 176 |
| Giurgiu | 77 |
| Hirsova | 151 |
| Iasi | 226 |
| Lugoj | 244 |
| Mehadia | 241 |
| Neamt | 234 |
| Oradea | 380 |
| Pitesti | 10 |
| Rimnicu Vilcea | 193 |
| Sibiu | 253 |
| Timisoara | 329 |
| Urziceni | 80 |
| Vaslui | 199 |
| Zerind | 374 |

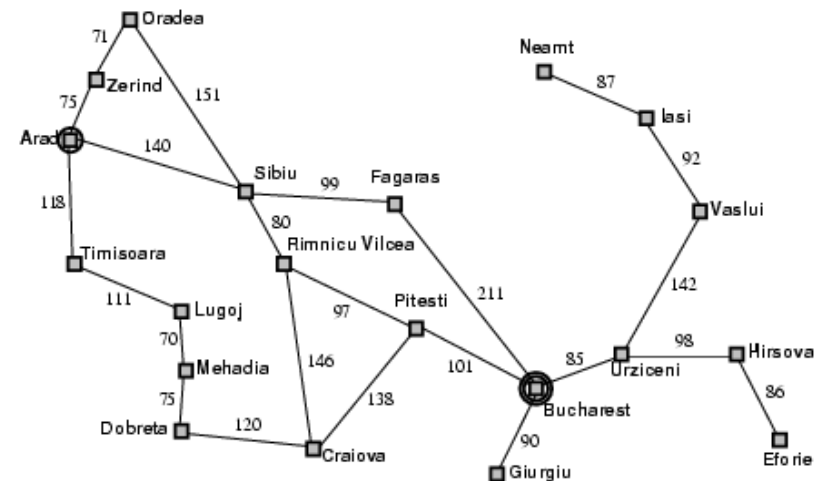
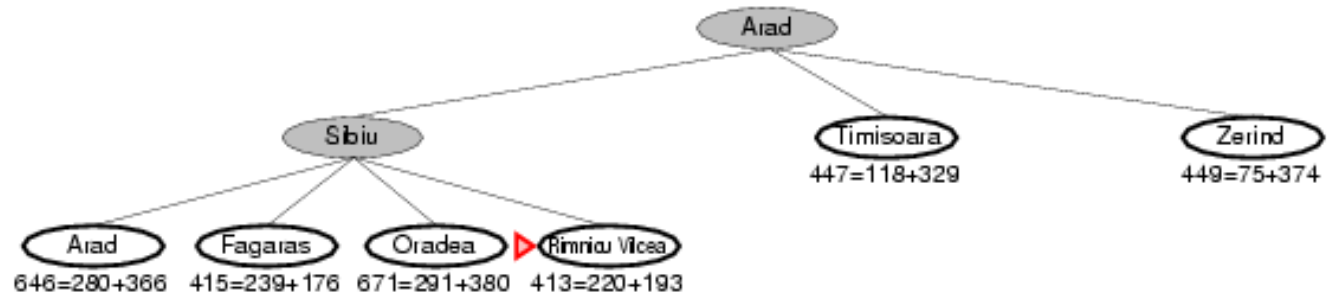
A* search example [Arad to Bucharest]



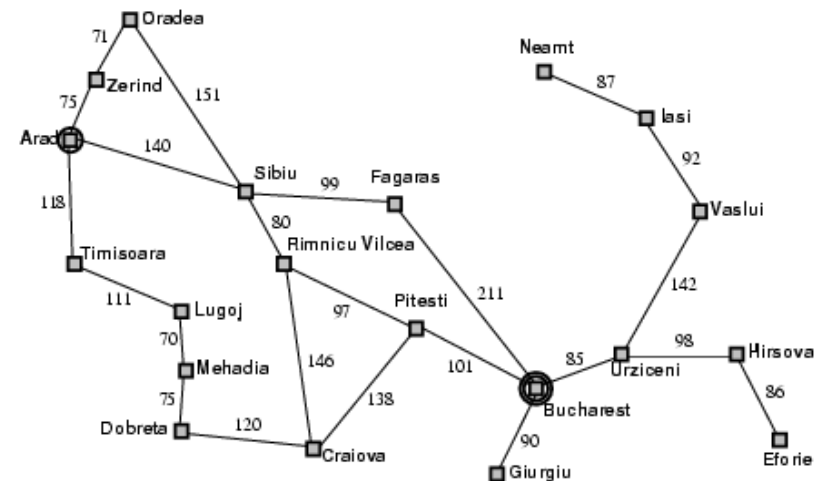
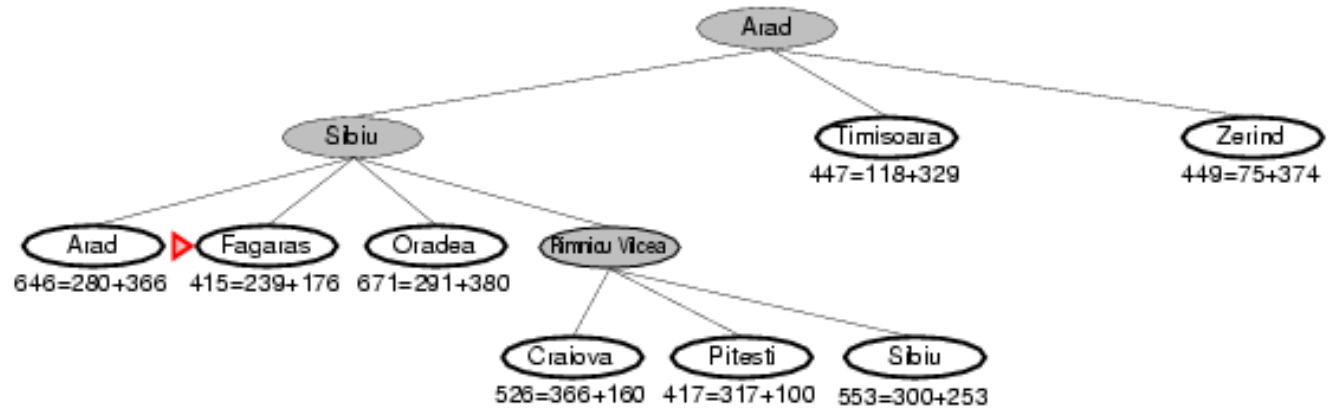
A* search example [Arad to Bucharest]



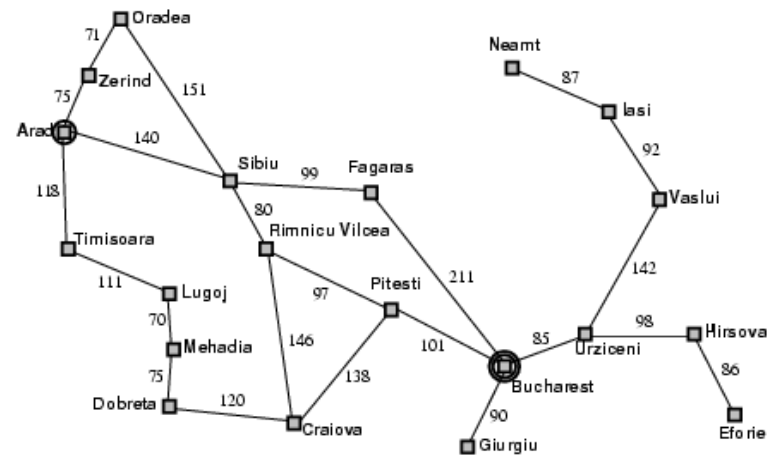
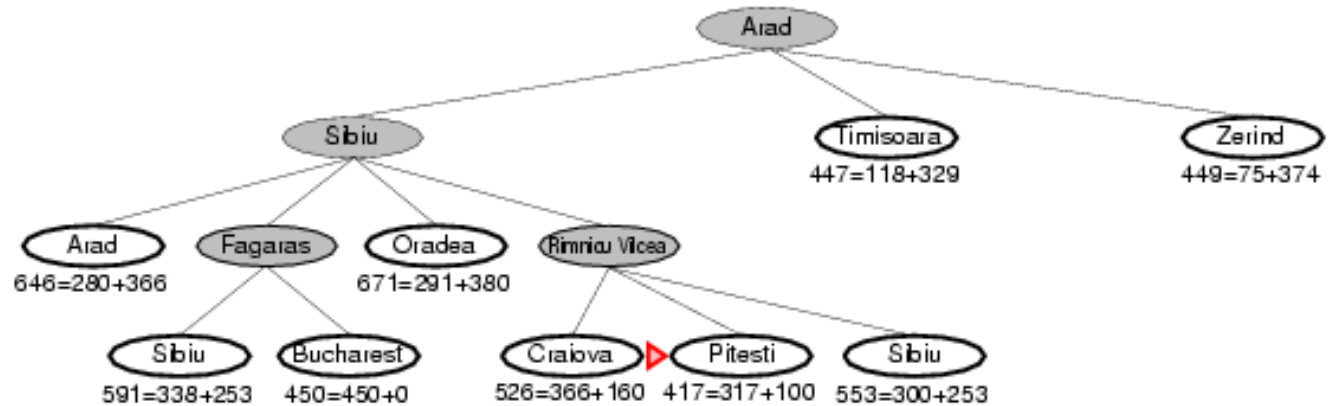
A* search example [Arad to Bucharest]



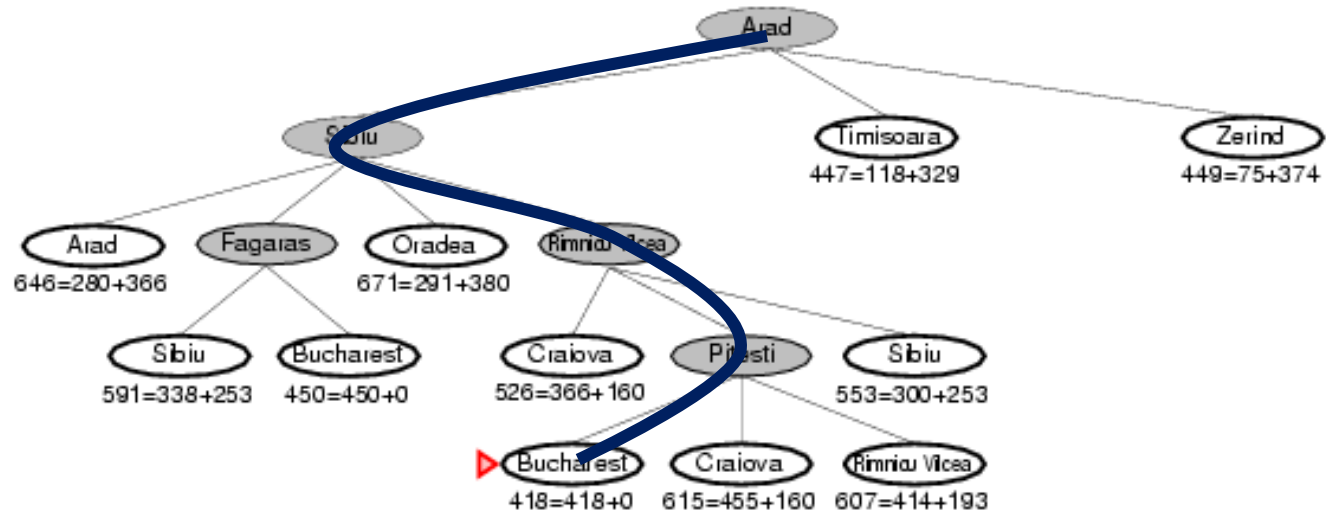
A* search example [Arad to Bucharest]



A* search example [Arad to Bucharest]

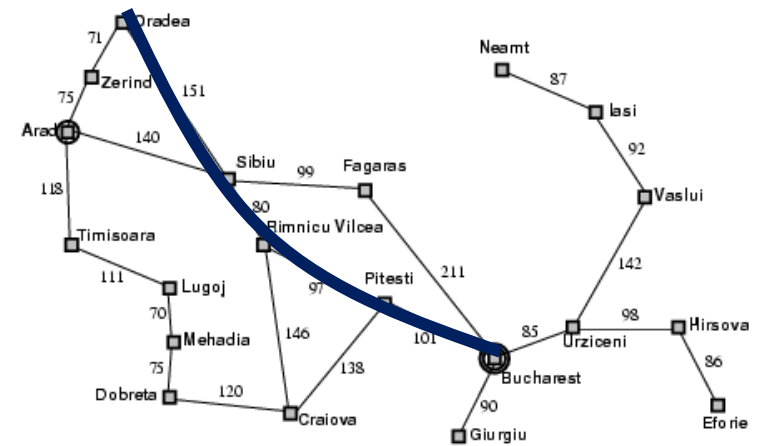


A* search example [Arad to Bucharest]



Finds Optimal Path

But Optimality depends
on what???



Admissible heuristics

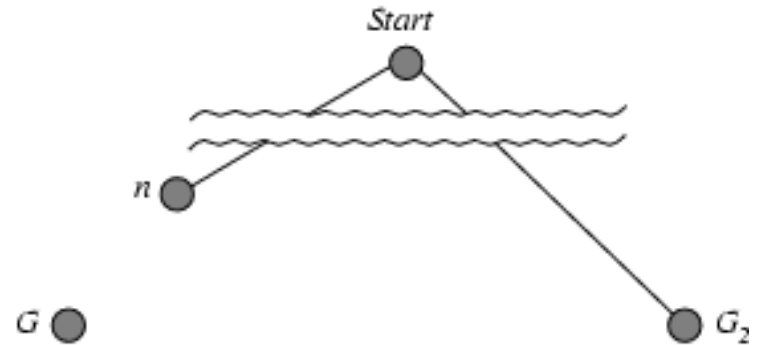
- A heuristic $h(n)$ is **admissible** if for every node n , $h(n) \leq h^*(n)$, where $h^*(n)$ is the **true** cost to reach the goal state from n .
- An admissible heuristic **never overestimates** the cost to reach the goal, i.e., it is **optimistic**
- Example: **$h_{SLD}(n)$** *is admissible*
 - never overestimates the actual road distance
- **Theorem:**

If $h(n)$ is admissible, A^* using TREE-SEARCH is optimal

Optimality of A* (proof)

- Suppose some suboptimal goal G_2 has been generated and is in the fringe. Let n be an unexpanded node in the fringe such that n is on a shortest path to an optimal goal G .

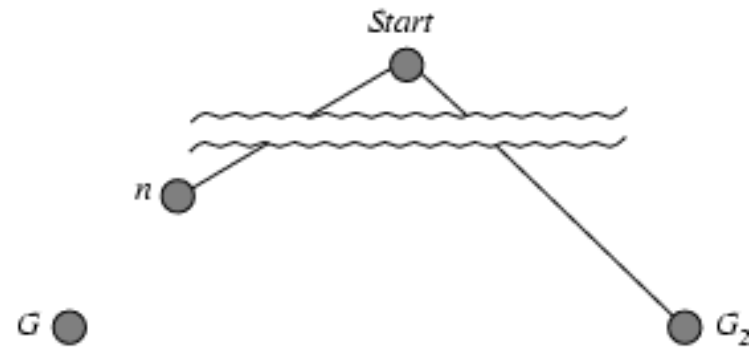
- **Need to prove that**
- $f(G_2) > f(G) > f(n)$
- and A* will never select G_2 for expansion



- $f(G_2) = g(G_2)$ since $h(G_2) = 0$
- $g(G_2) > g(G)$ since G_2 is suboptimal
- $f(G) = g(G)$ since $h(G) = 0$
- **$f(G_2) > f(G)$** from above

Optimality of A* (proof)

- Suppose some suboptimal goal G_2 has been generated and is in the fringe. Let n be an unexpanded node in the fringe such that n is on a shortest path to an optimal goal G .



- $f(G_2) > f(G)$ from above
- $h(n) \leq h^*(n)$ **since h is admissible**
- $g(n) + h(n) \leq g(n) + h^*(n)$
- $f(n) \leq f(G)$

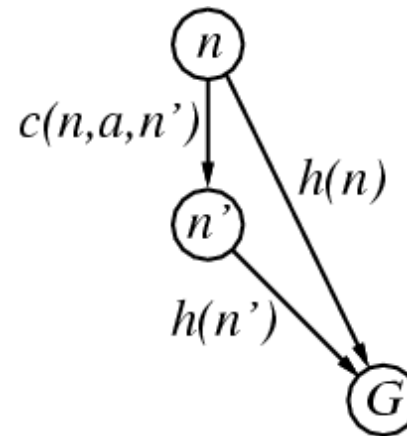
Hence $f(G_2) > f(G) > f(n)$, and A* will never select G_2 for expansion

Optimality for graphs?

- Admissibility is not sufficient for graph search
 - In graph search, the optimal path to a repeated state could be discarded if it is not the first one generated
 - Can fix problem by requiring consistency property for $h(n)$
- A heuristic is **consistent** if for every successor n' of a node n generated by any action a ,

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

(*aka "monotonic"*)



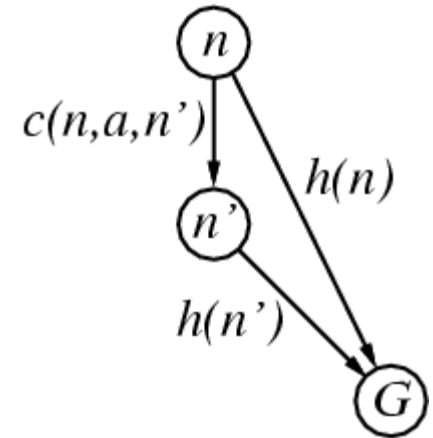
- admissible heuristics are generally consistent

A* is optimal with consistent heuristics

- If h is consistent, we have

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

i.e., $f(n)$ is non-decreasing along any path.

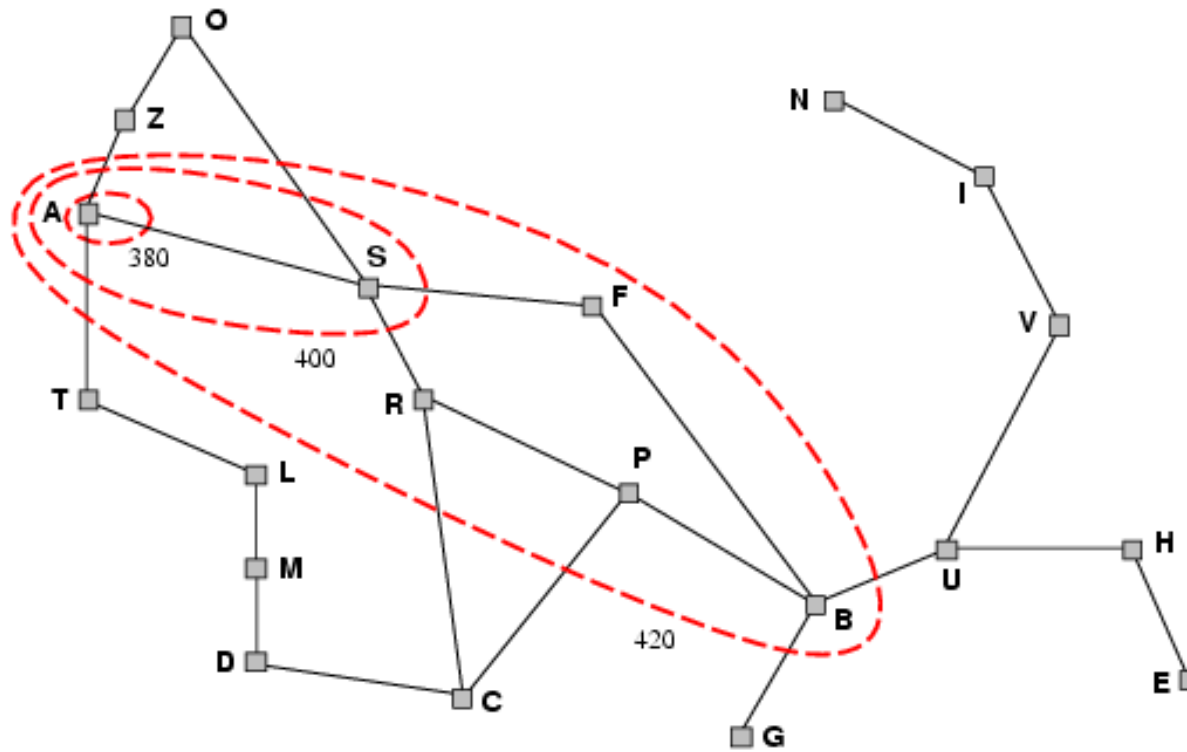


Thus, first goal-state selected for expansion must be optimal

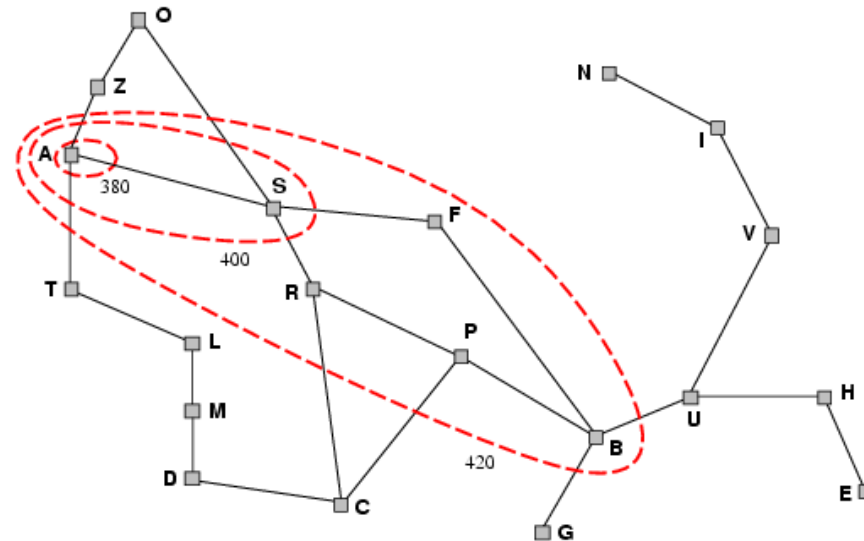
- Theorem:
 - If $h(n)$ is consistent, A* using GRAPH-SEARCH is optimal

Contours of A* Search

- A* expands nodes in order of increasing f value
- Gradually adds " f -contours" of nodes
- Contour i has all nodes with $f=f_i$, where $f_i < f_{i+1}$



Contours of A* Search



- With uniform-cost $h(n) = 0$, contours will be circular
- A* with good heuristics, contours will be focused around optimal path
- A* will expand all nodes with cost $f(n) < C^*$

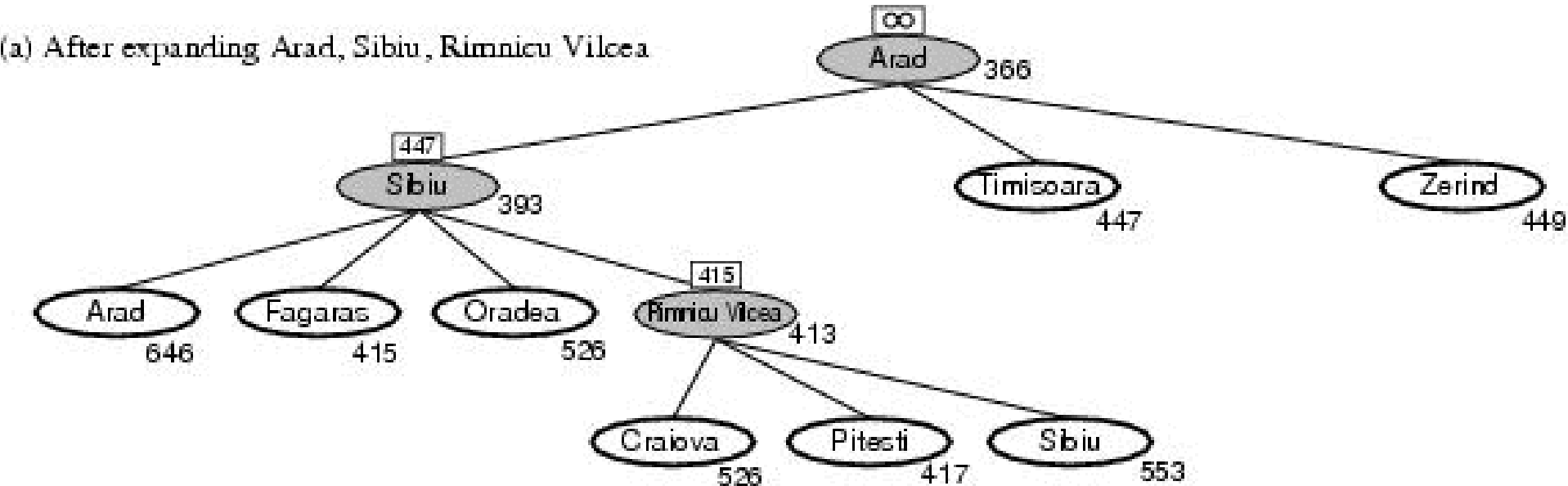
Properties of A*

- Complete?
 - **Yes** (unless there are infinitely many nodes with $f \leq f(G)$)
- Optimal?
 - **Yes**
 - Also optimally efficient:
 - No other optimal algorithm will expand fewer nodes, for a given heuristic
- Time?
 - **Exponential** in worst case
- Space?
 - **Exponential** in worst case
- A* expands all nodes with $f(n) < C^*$
 - This will be exponentially large

Recursive Best-First Search (RBFS)

- In practice A^* runs out of memory before it runs out of time
 - How can we solve the memory problem for A^* search?
 - The solution is **Recursive Best First Search (RBFS)**
- Similar to DFS, but keeps track of the f-value of the best alternative path available from any ancestor of the current node
- If current node exceeds f-limit \rightarrow backtrack to alternative path
- As it backtracks, replace f-value of each node along the path with the best $f(n)$ value of its children
 - This allows it to return to this subtree, if it turns out to look better than alternatives

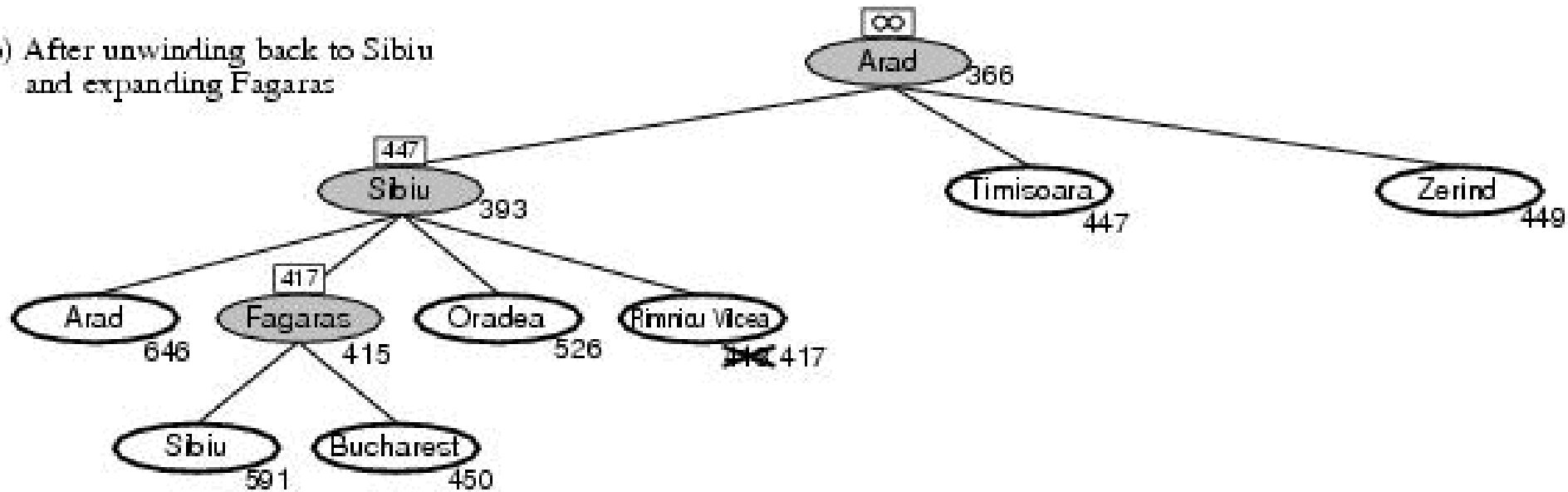
Recursive Best First Search: Example



- Path until **Rumnicu Vilcea** is already expanded
- **Above node**; f -limit for every recursive call is shown on top.
- Below node: $f(n)$
- The path is followed until Pitesti which has a f -value worse than the f -limit.

RBFS example

(b) After unwinding back to Sibiu and expanding Fagaras



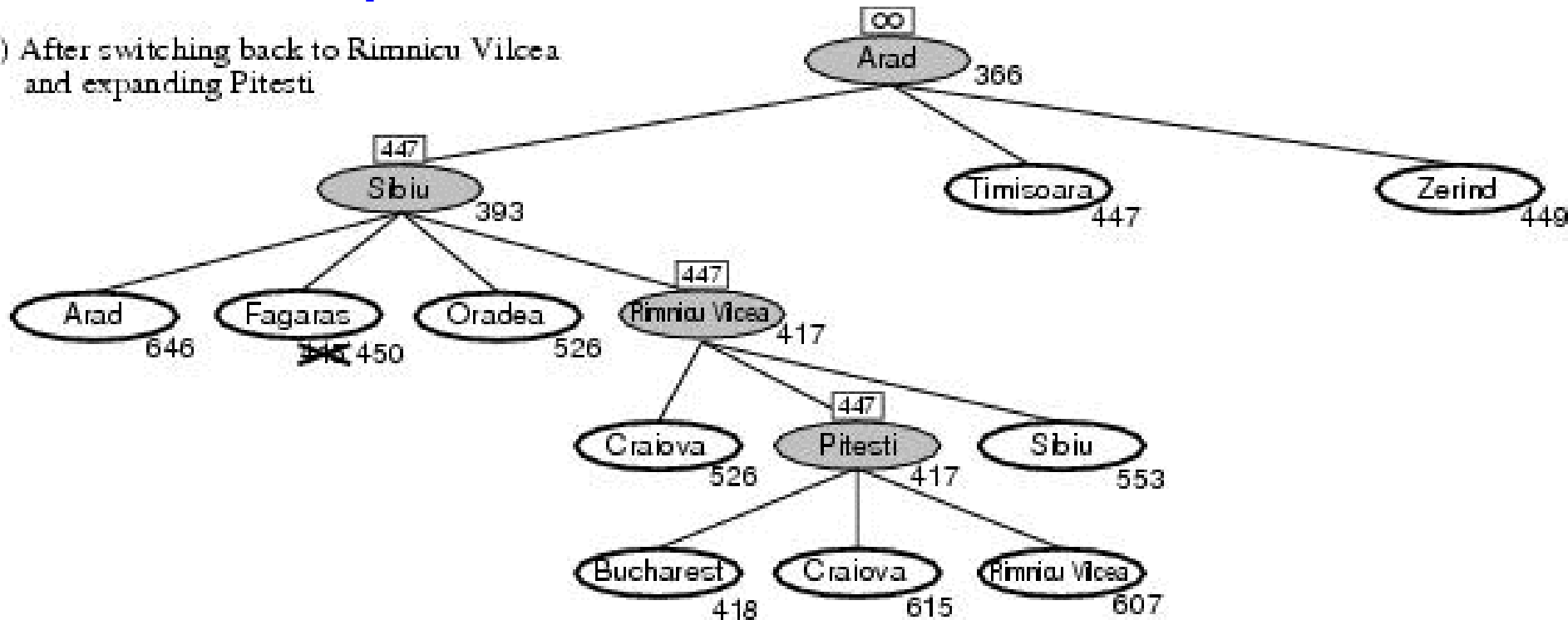
- **Unwind recursion** and store best f -value for current best leaf Pitesti

$result, f[best] \leftarrow \text{RBFS}(\text{problem}, best, \min(f_limit, alternative))$

- $best$ is now Fagaras. Call RBFS for new $best$
 - $best$ value is now 450 which is larger than best f -value

RBFS example

(c) After switching back to Rimnicu Vilcea and expanding Pitesti



- **Unwind recursion again** and store best f -value for current best leaf Fagaras

$$result, f[best] \leftarrow \text{RBFS}(\text{problem}, \text{best}, \min(f_limit, \text{alternative}))$$
- $best$ is now Rimnicu Viclea (again). Call RBFS for new $best$
 - Subtree is again expanded.
 - Best *alternative* subtree is now through Timisoara.
- Solution is found since because $447 > 418$.

RBFS properties

- Like A^* , optimal if $h(n)$ is admissible
- Time complexity difficult to characterize
 - Depends on accuracy of $h(n)$ and how often best path changes.
 - Can end up “switching” back and forth
- Space complexity is $O(bd)$
 - Other extreme to A^* - uses ***too little*** memory.

(Simplified) Memory-bounded A* (SMA*)

- This is like A*, but when memory is full we delete the worst node (largest f-value).
- Like RBFS, we remember the best descendant in the branch we delete.
- If there is a tie (equal f-values) we delete the oldest nodes first.
- simplified-MA* finds the optimal *reachable* solution given the memory constraint.
- **Time can still be exponential.**

Designing heuristic functions

- Heuristics for the 8-puzzle

$h_1(n)$ = number of misplaced tiles

$h_2(n)$ = total Manhattan distance (number of squares from desired location of each tile)

| | | |
|---|---|---|
| 7 | 2 | 4 |
| 5 | | 6 |
| 8 | 3 | 1 |

Start State

| | | |
|---|---|---|
| | 1 | 2 |
| 3 | 4 | 5 |
| 6 | 7 | 8 |

Goal State

$$h_1(\text{start}) = 8$$

$$h_2(\text{start}) = 3+1+2+2+2+3+3+2 = 18$$

- Are h_1 and h_2 admissible?

Inventing heuristics via “relaxed problems”

- A problem with **fewer restrictions** on the actions is called a **relaxed problem**
- **The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem**
- If the rules of the 8-puzzle are relaxed so that a tile can move **anywhere**, then $h_1(n)$ gives the shortest solution
- If the rules are relaxed so that a tile can move to **any adjacent square**, then $h_2(n)$ gives the shortest solution

Notion of dominance

- If $h_2(n) \geq h_1(n)$ for all n (both admissible) then h_2 **dominates** h_1
In other words h_2 is better for search
- Typical search costs (average number of nodes expanded) for 8-puzzle problem

$d=12$ IDS = 3,644,035 nodes
 $A^*(h_1)$ = 227 nodes
 $A^*(h_2)$ = 73 nodes

$d=24$ IDS = too many nodes
 $A^*(h_1) = 39,135$ nodes
 $A^*(h_2) = 1,641$ nodes

Effective branching factor

- **Effective branching factor b^***
 - Is the branching factor that a uniform tree of depth d would have in order to contain $N+1$ nodes.

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

- Measure is fairly constant for sufficiently hard problems.
 - Can thus provide a good guide to the heuristic's overall usefulness.

Effectiveness of different heuristics

| d | Search Cost | | | Effective Branching Factor | | |
|-----|-------------|------------|------------|----------------------------|------------|------------|
| | 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 |

- Results averaged over random instances of the 8-puzzle

Summary

- Uninformed search methods have their limits
- Informed (or heuristic) search uses problem-specific heuristics to improve efficiency
 - Best-first
 - A*
 - RBFS
 - SMA*
 - Techniques for generating heuristics
- Can provide significant speed-ups in practice
 - e.g., on 8-puzzle
 - But can still have worst-case exponential time complexity