COMP3411/9414: Artificial Intelligence

Module 2

Solving problems by searching
Informed Search

Russell & Norvig, Chapter 3.

Outline

- Best-first search
- Greedy best-first search
- A* search
- Heuristics

Informed (Heuristics) Searches

- **Informed search** strategy
  - one that uses problem-specific knowledge beyond the definition of the problem itself
  - can find solutions more efficiently than can an uninformed strategy

- **Uninformed** search algorithms—algorithms that are given no information about the problem other than its definition.
  - some of these algorithms can solve any solvable problem, none of them can do so efficiently
- **Informed** search algorithms, can do quite well given some guidance on where to look for solutions.

Search Strategies

General Search algorithm:

- add initial state to queue
- repeat:
  - take node from front of queue
  - test if it is a goal state; if so, terminate
  - "expand" it, i.e. generate successor nodes and add them to the queue

Search strategies are distinguished by the order in which new nodes are added to the queue of nodes awaiting expansion.

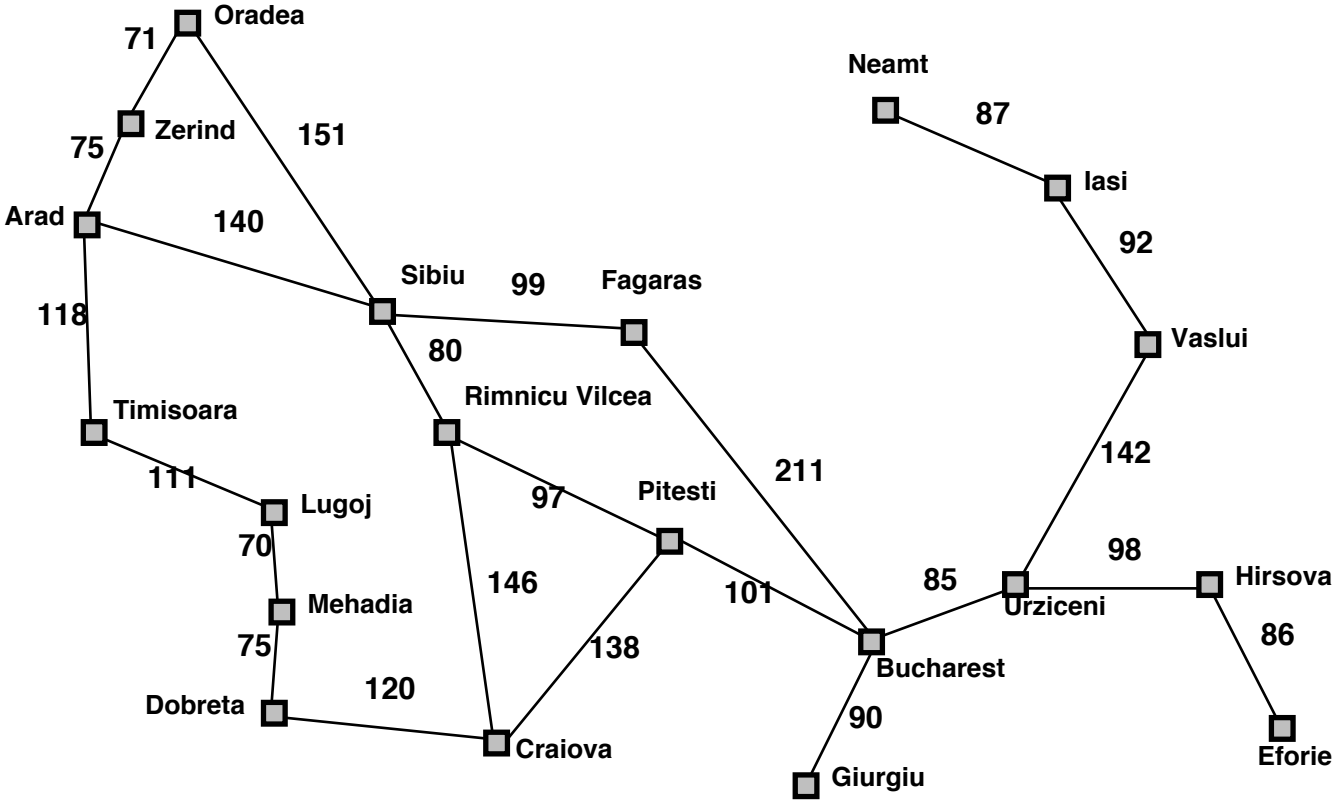
Search Strategies

- BFS and DFS treat all new nodes the same way:
  - BFS add all new nodes to the back of the queue
  - DFS add all new nodes to the front of the queue

- Best First Search uses an evaluation function f () to order the nodes in the queue;
  - Similar to UCS $f(n)$ = cost $g(n)$ of path from root to node n

- Informed or Heuristic search strategies incorporate into f () an estimate of distance to goal
  - Greedy Search $f(n)$ = estimate $h(n)$ of cost from node n to goal
  - A* Search $f(n) = g(n) + h(n)$

Romania with step costs in km



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

However, we are often looking for the path with the shortest total distance rather than the number of steps.

Best-first search

- Use an evaluation function *f(n)* for each node
  - estimate of "desirability"
  - → Expand most desirable unexpanded node

- Implementation:
  Order the nodes in fringe in decreasing order of desirability

- Special cases:
  - greedy best-first search
  - A* search

Heuristic Function

There is a whole family of Best First Search algorithms with different evaluation functions *f* (). A key component of these algorithms is a heuristic function:

- Heuristic function *h*: {Set of nodes} → **R** :
  - *h*(*n*) = estimated cost of the cheapest path from current node *n* to *goal* node.
    - in the area of search, heuristic functions are problem specific functions that provide an estimate of solution cost.
    - nonnegative, with one constraint: if *n* is a goal node, then *h(n) = 0*.

Greedy best-first search

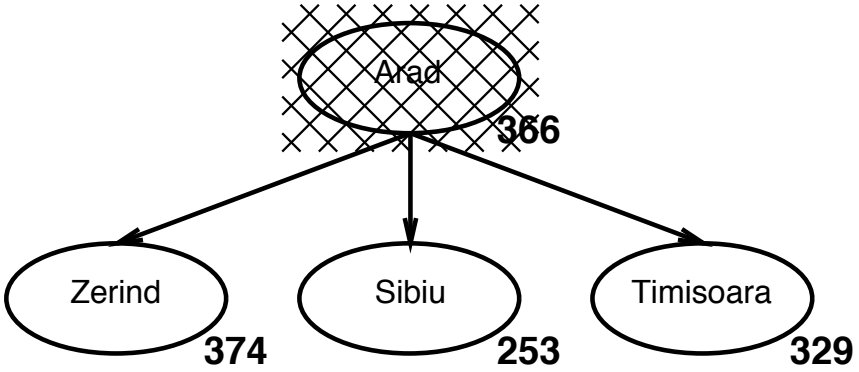
- Greedy Best-First Search: Best-First Search that selects the next node for expansion using the heuristic function for its evaluation function, i.e. $f(n) = h(n)$
- $h(n)=0 \Rightarrow n$ is a goal state
- i.e. greedy search minimizes the estimated cost to the goal; it expands whichever node n is estimated to be closest to the goal.
- Greedy: tries to "bite off" as big a chunk of the solution as possible, without worrying about long-term consequences.

Straight Line Distance as a Heuristic

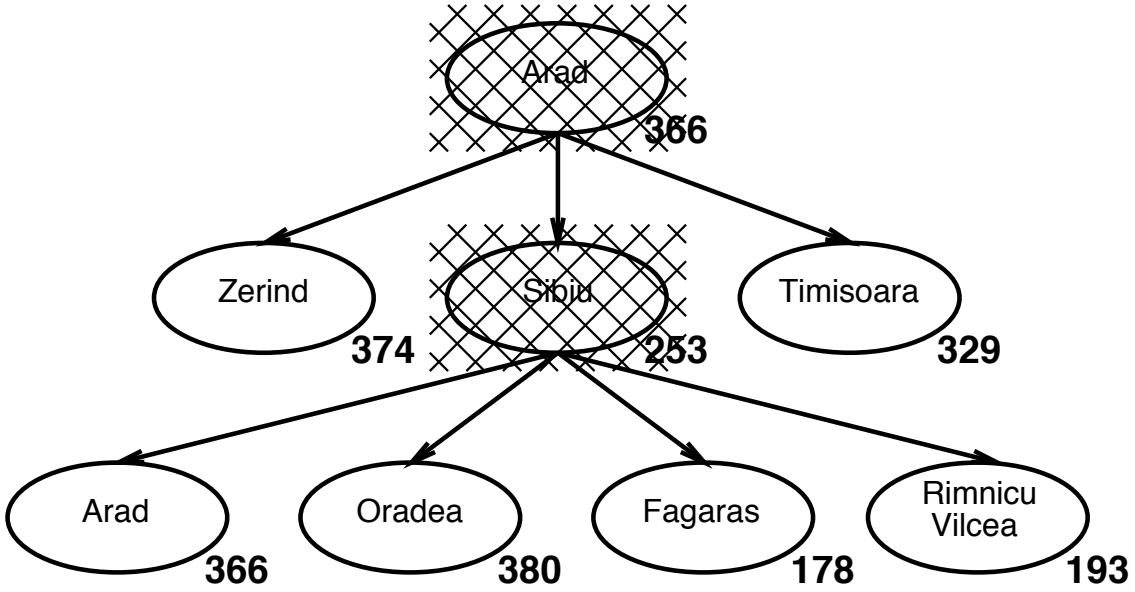
- $h_{SLD}(n)$ = straight-line distance between n and the goal location (Bucharest).
- Assume that roads typically tend to approximate the direct connection between two cities.
- Need to know the map coordinates of the cities:

$$\sqrt{(Sibiu_x - Bucharest_x)^2 + (Sibiu_y - Bucharest_y)^2}$$

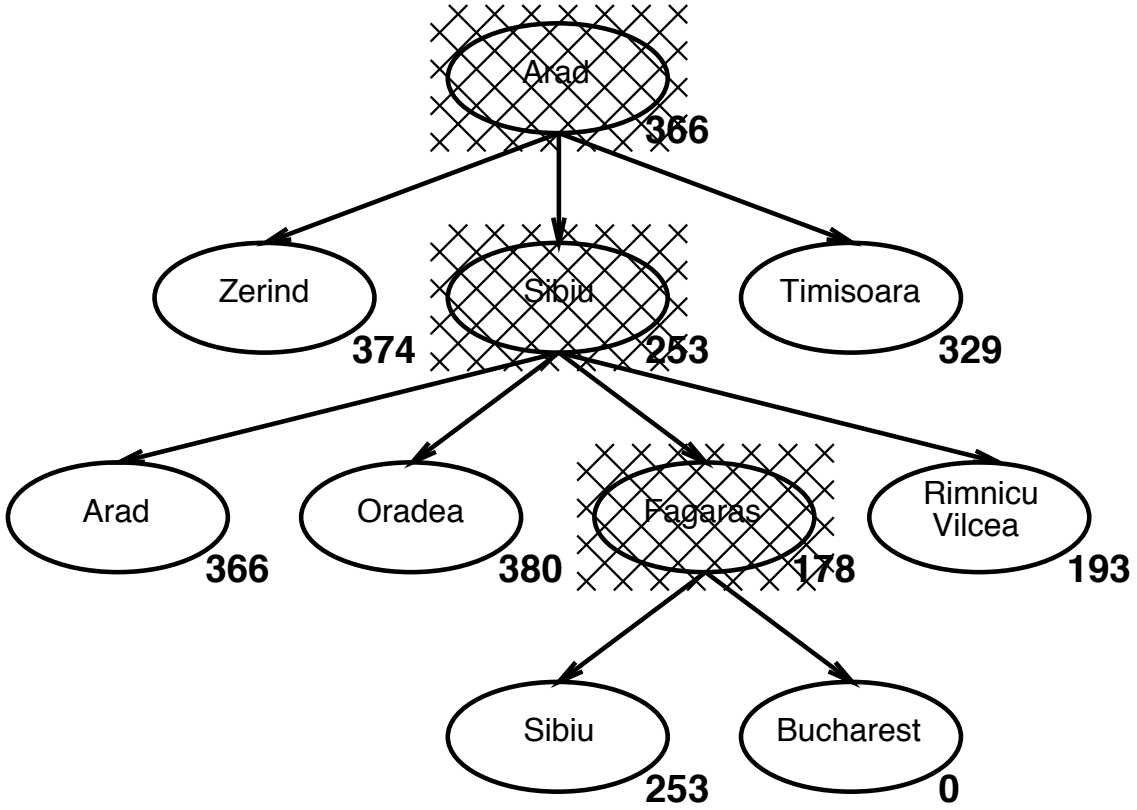
Greedy Best-First Search Example



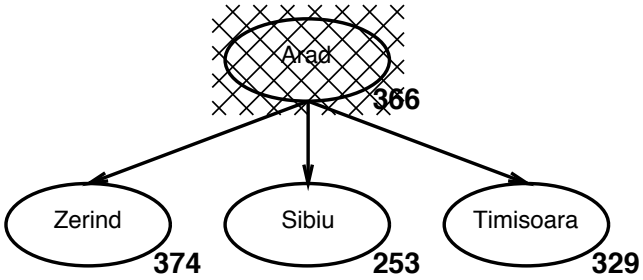
Greedy Best-First Search Example

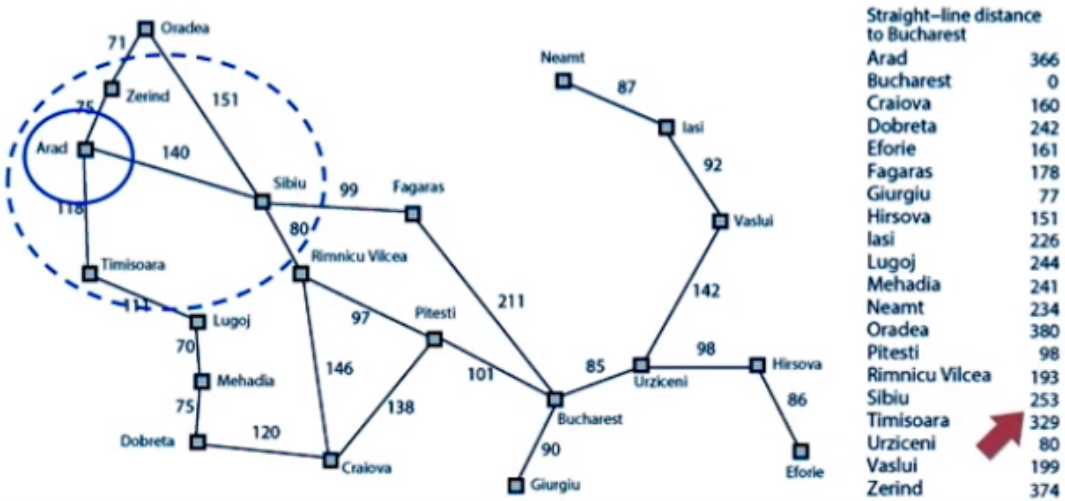


Greedy Best-First Search Example

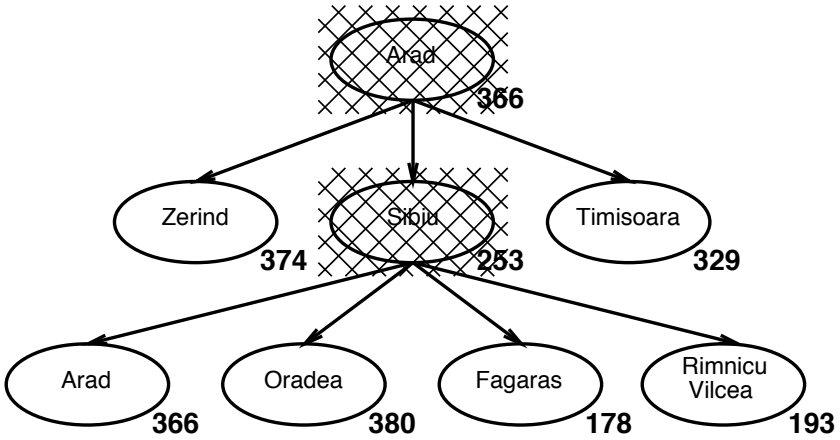


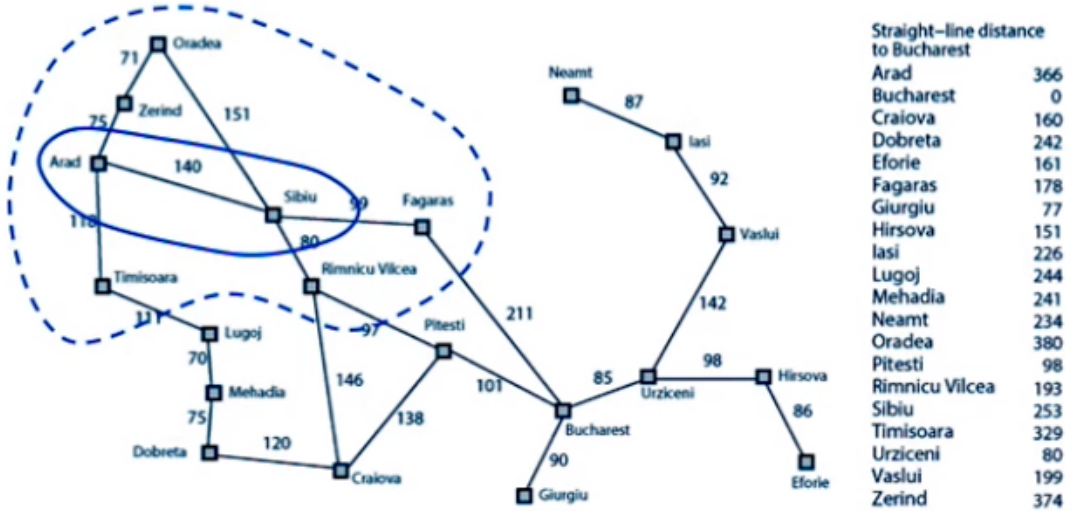
Greedy Best-First Search Example





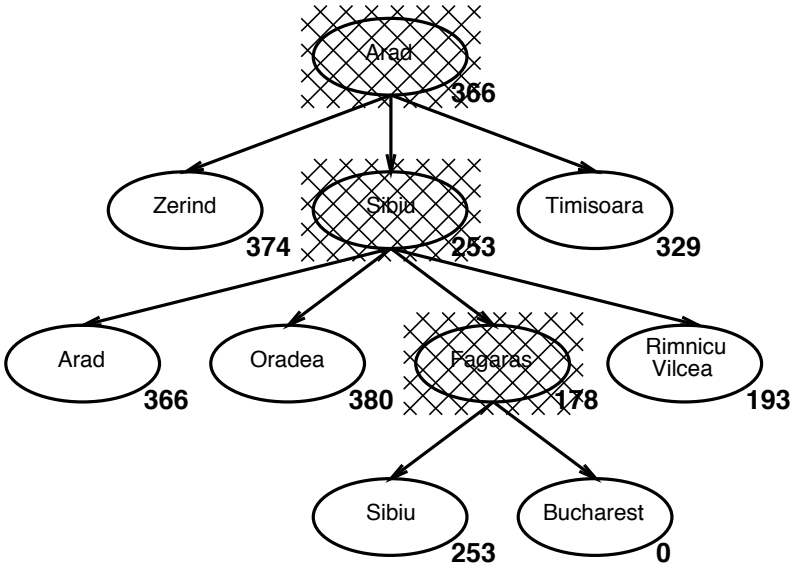
Greedy Best-First Search Example

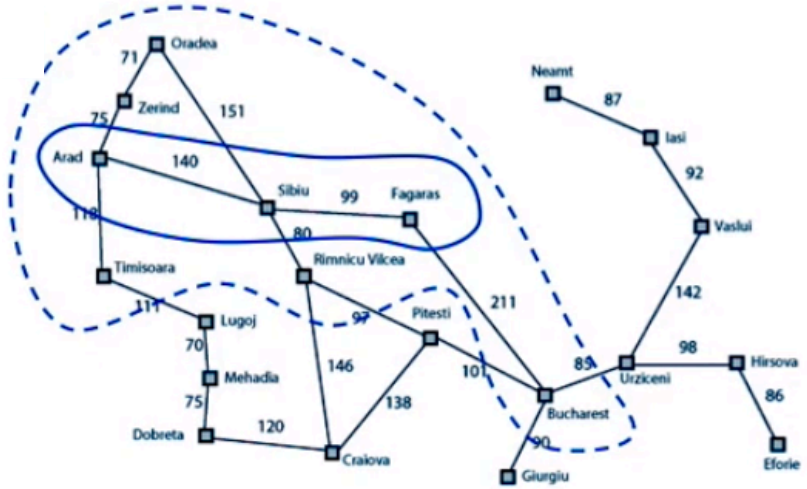




Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

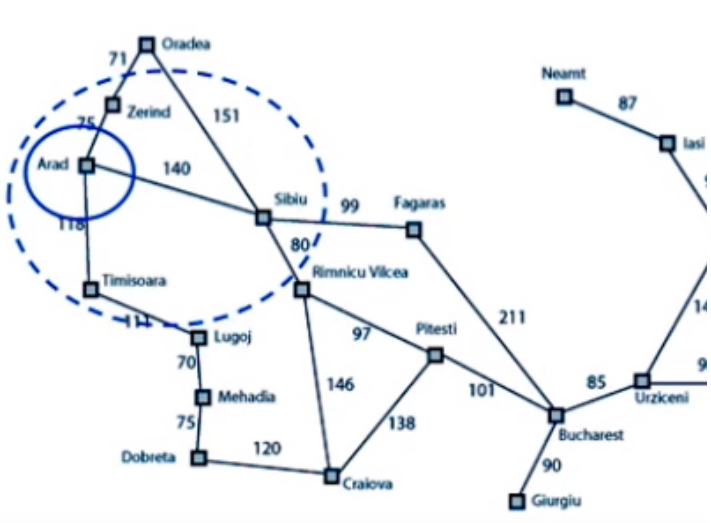
Greedy Best-First Search Example

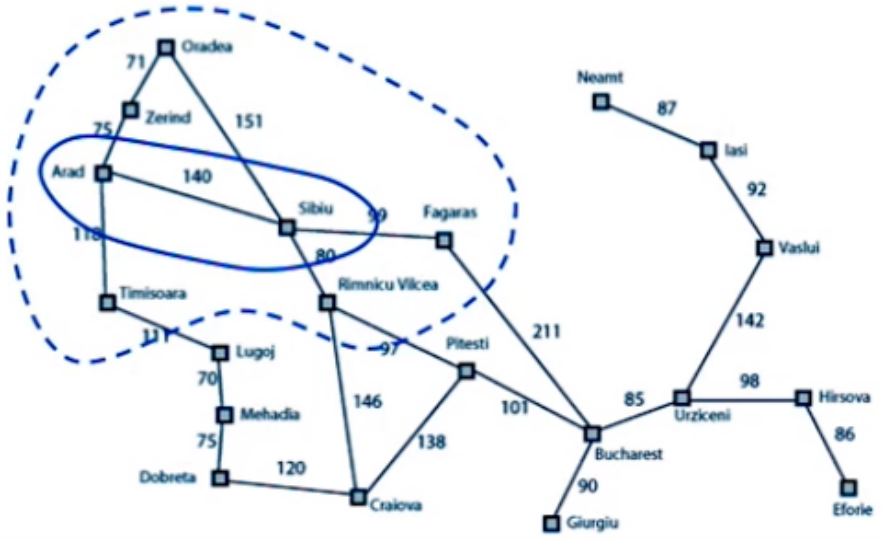




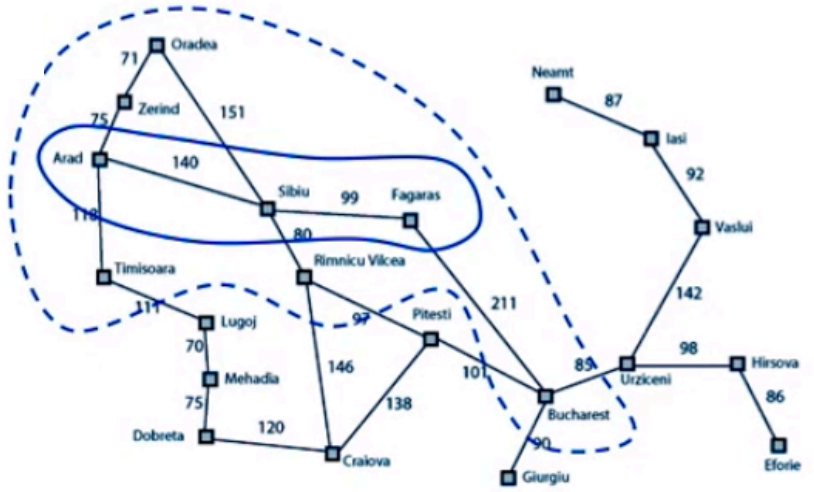
Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy Best-First Search Example





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

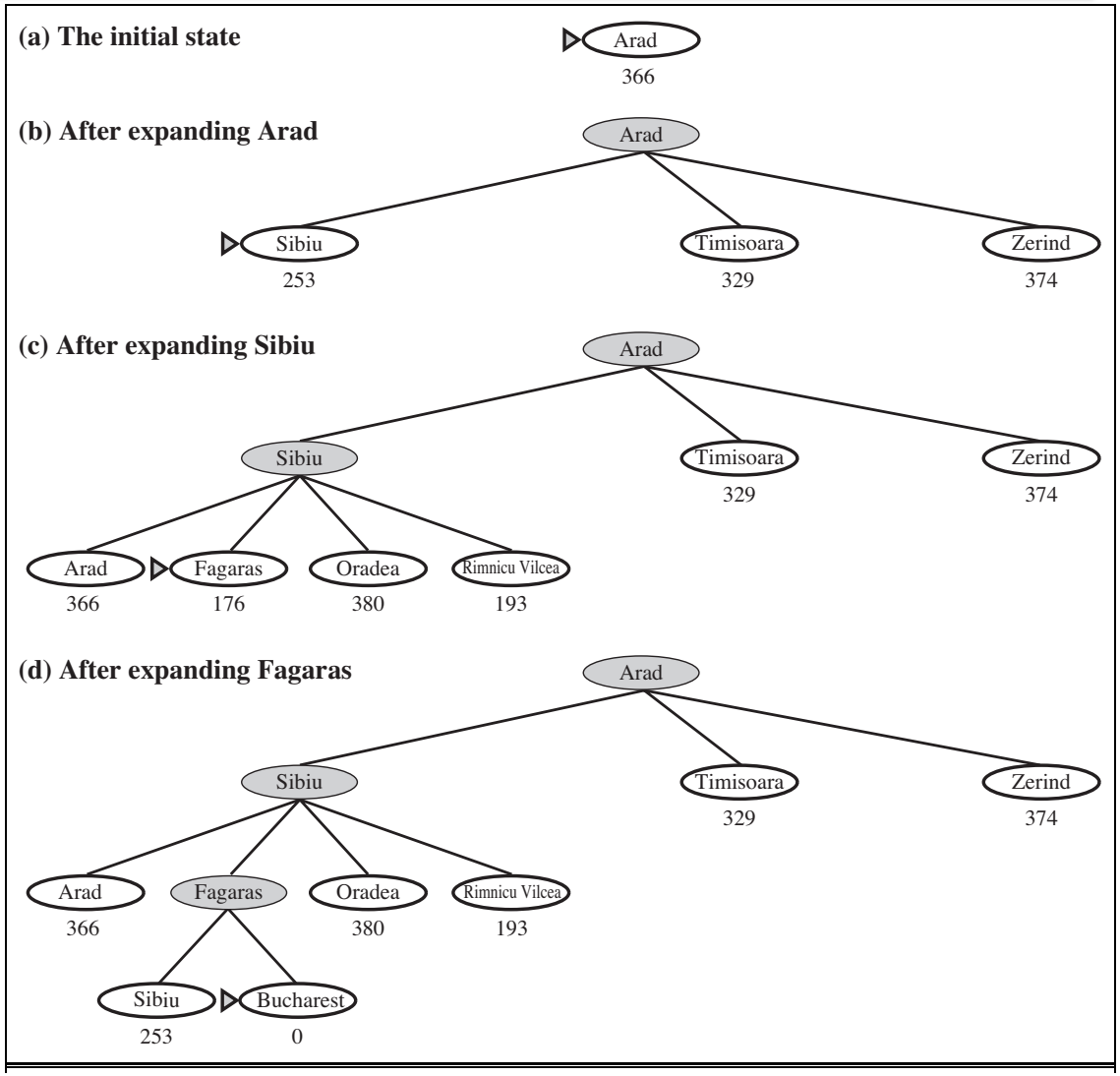


Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Examples of Greedy Best-First Search

Implementation:

Order the nodes in decreasing order of desirability



Stages in a greedy best-first tree search for Bucharest with the straight-line distance heuristic h_{SLD}.

Nodes are labeled with their h-values.

Properties of Greedy Best-First Search

- For this particular problem, greedy best-first search using h_{SLD} finds a solution without ever expanding a node that is not on the solution path;
  - its search cost is minimal.

- It is not optimal,
  - the path via Sibiu and Fagaras to Bucharest is 32 kilometers longer than the path through Rimnicu Vilcea and Pitesti.
  - This shows why the algorithm is called "greedy"—at each step it tries to get as close to the goal as it can.

Examples of Greedy Best-First Search

Try

- Iasi to Fagaras
- Fagaras to Iasi
- Rimnicu Vilcea to Lugoj

Examples of Greedy Best-First Search

Try

- Iasi to Fagaras
- Fagaras to Iasi
- Rimnicu Vilcea to Lugoj

The algorithm will never find this solution, however, because expanding *Neamt* puts Iasi back into the frontier, *Iasi* is closer to *Fagaras* than *Vaslui* is, and so *Iasi* will be expanded again, leading to an infinite loop.

Properties of Greedy Best-First Search

- Complete: No! can get stuck in loops, e.g.,
  Iasi → Neamt → Iasi → Neamt → ...
  Complete in finite space with repeated-state checking

- Time: $O(b^m)$, where m is the maximum depth in search space.

- Space: $O(b^m)$ (retains all nodes in memory)

- Optimal: No!
  - e.g., the path Sibiu → Fagaras → Bucharest is 32 km longer than Sibiu → Rimnicu Vilcea → Pitesti → Bucharest.

Properties of Greedy Best-First Search

- Complete: No! can get stuck in loops, e.g.,
  Iasi → Neamt → Iasi → Neamt → ...
  Complete in finite space with repeated-state checking

- Time: $O(b^m)$, where m is the maximum depth in search space.

- Space: $O(b^m)$ (retains all nodes in memory)

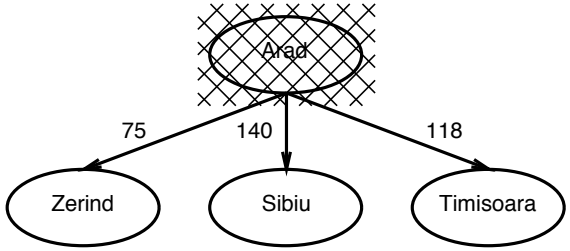
- Optimal: No!
  - e.g., the path Sibiu → Fagaras → Bucharest is 32 km longer than Sibiu → Rimnicu Vilcea → Pitesti → Bucharest.

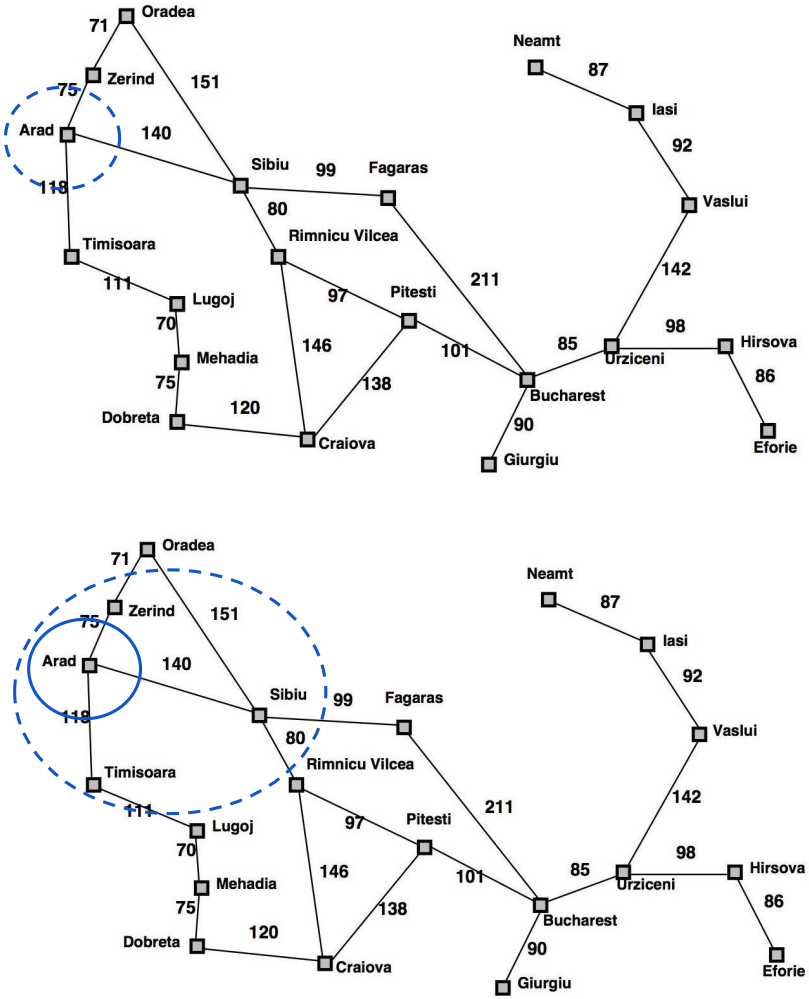
Therefore Greedy Search has the same deficits as Depth-First Search.
However, a good heuristic can reduce time and memory costs substantially.

Recall: Uniform-Cost Search

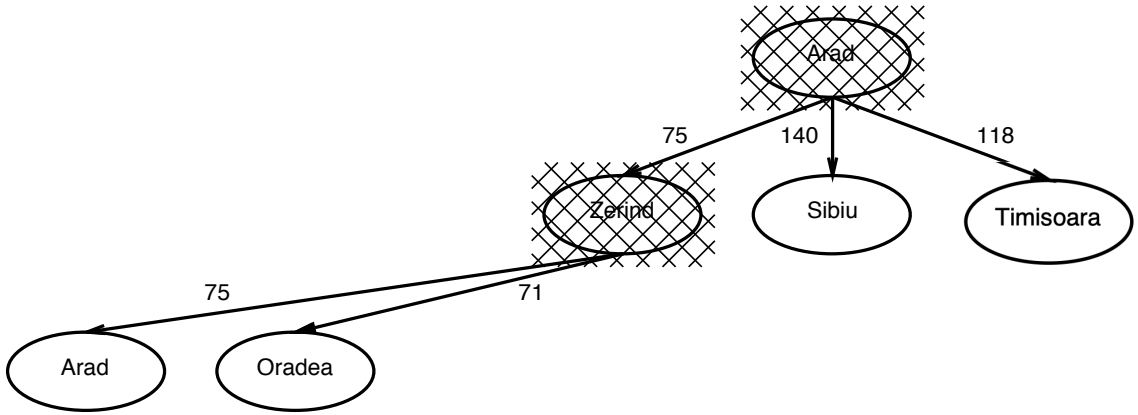
- Expand root first, then expand least-cost unexpanded node
- Implementation: QUEUEINGFN = insert nodes in order of increasing path cost.
- Reduces to breadth-first search when all actions have same cost
- Finds the cheapest goal provided path cost is monotonically increasing along each path (i.e. no negative-cost steps)

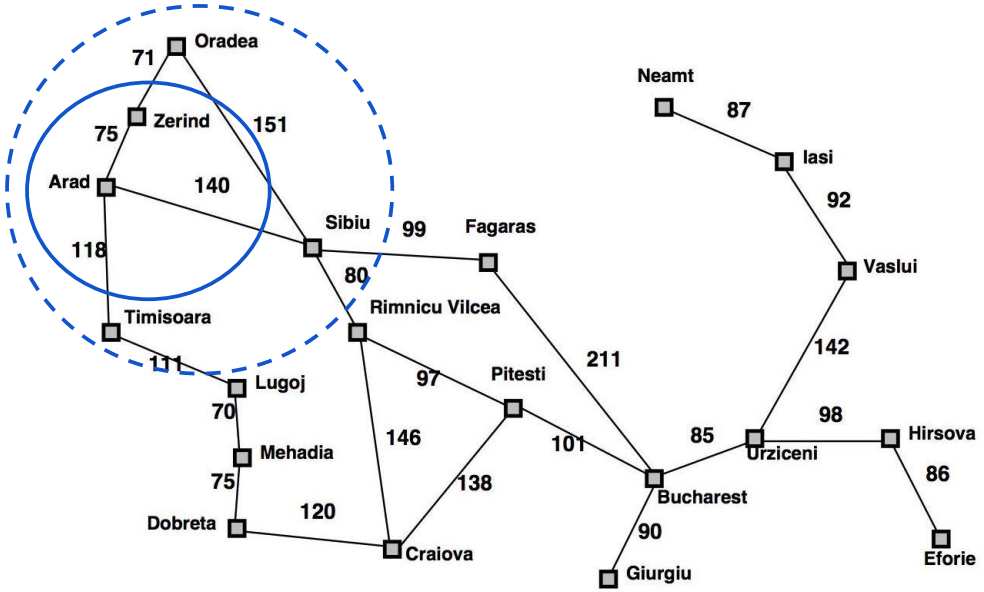
Uniform Cost Search



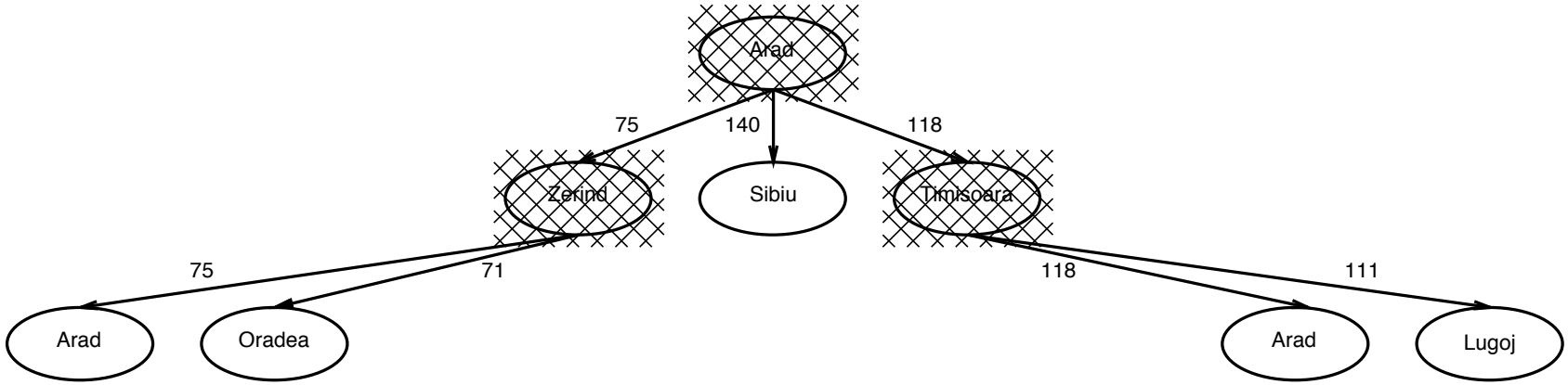


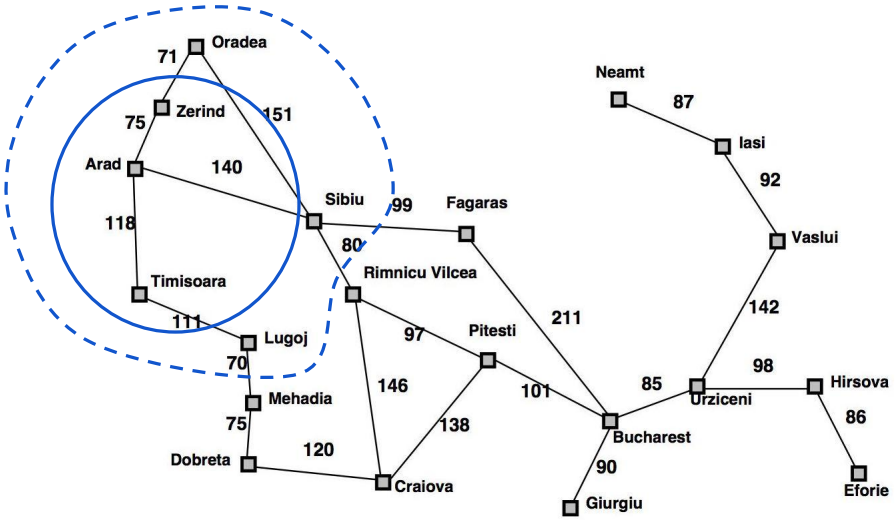
Uniform-Cost Search





Uniform-Cost Search





Properties of Uniform Cost Search

- Complete? Yes, if *b* is finite and step costs ≥ ε with ε > 0.
- Optimal? Yes.
- Guaranteed to find optimal solution, but does so by exhaustively expanding all nodes closer to the initial state than the goal.

Q: can we still guarantee optimality but search more efficiently, by giving priority to more "promising" nodes?

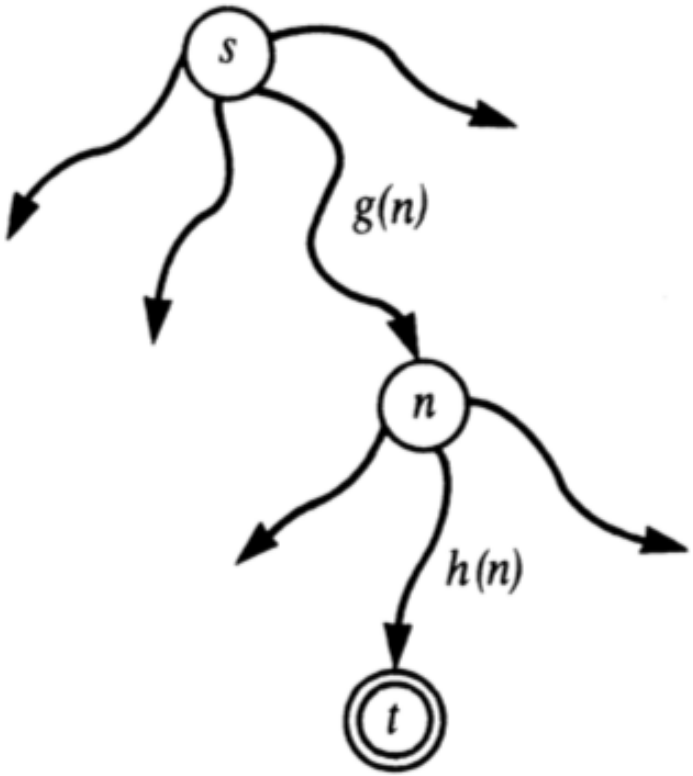
A* Search

- A* Search uses evaluation function $f(n) = g(n) + h(n)$
  - $g(n)$ = cost from initial node to node n
  - $h(n)$ = estimated cost of cheapest path from n to goal
  - $f(n)$=estimated total cost of cheapest solution through node n

- Greedy Search minimizes $h(n)$
  - efficient but not optimal or complete

- Uniform Cost Search minimizes $g(n)$
  - optimal and complete but not efficient

Heuristic function

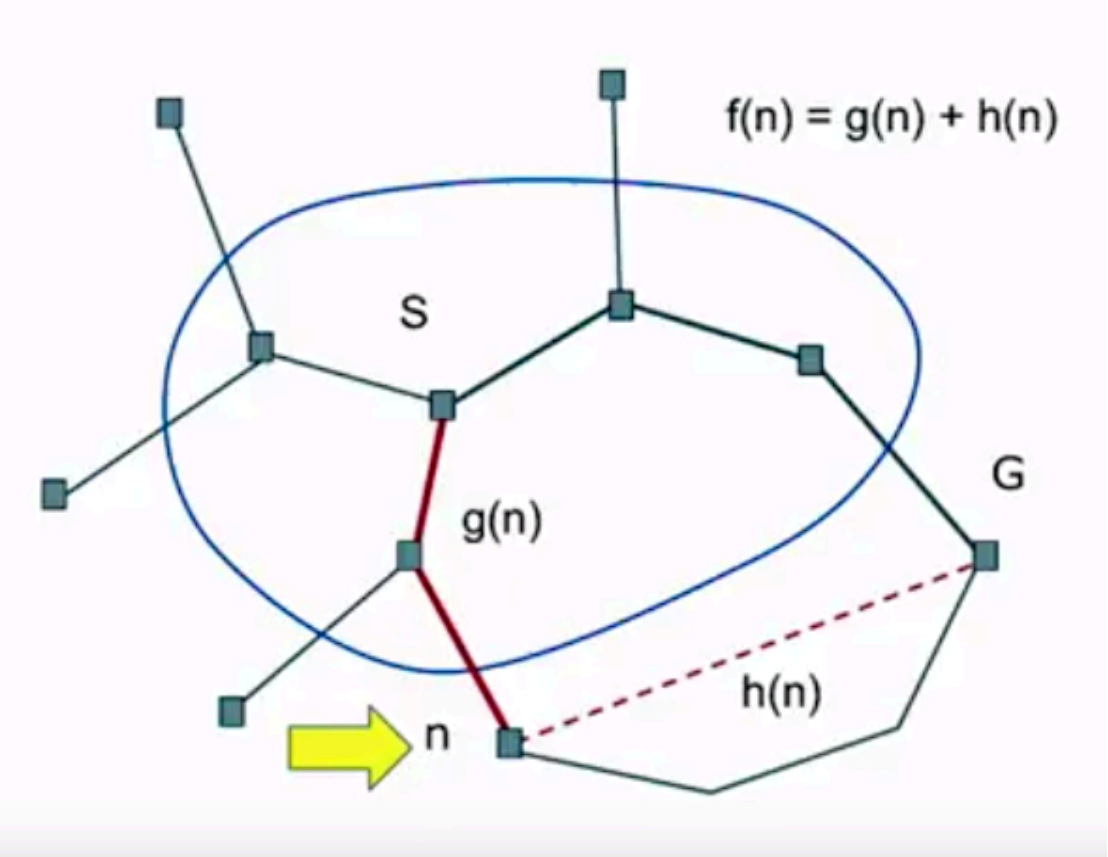


Heuristic estimate *f(n) of*
the cost of the cheapest paths from ***s*** to ***t***
via ***n***: *f(n) = g(n)+ h(n)*

g(n) is an estimate of the cost of an optimal path from s to n

h(n) is an estimate of the cost of an optimal path from n to t.

A* Search



A* Search

- A* Search minimizes $f(n) = g(n) + h(n)$
  - idea: preserve efficiency of Greedy Search but avoid expanding paths that are already expensive

- Q: is A* Search optimal and complete ?

A* Search

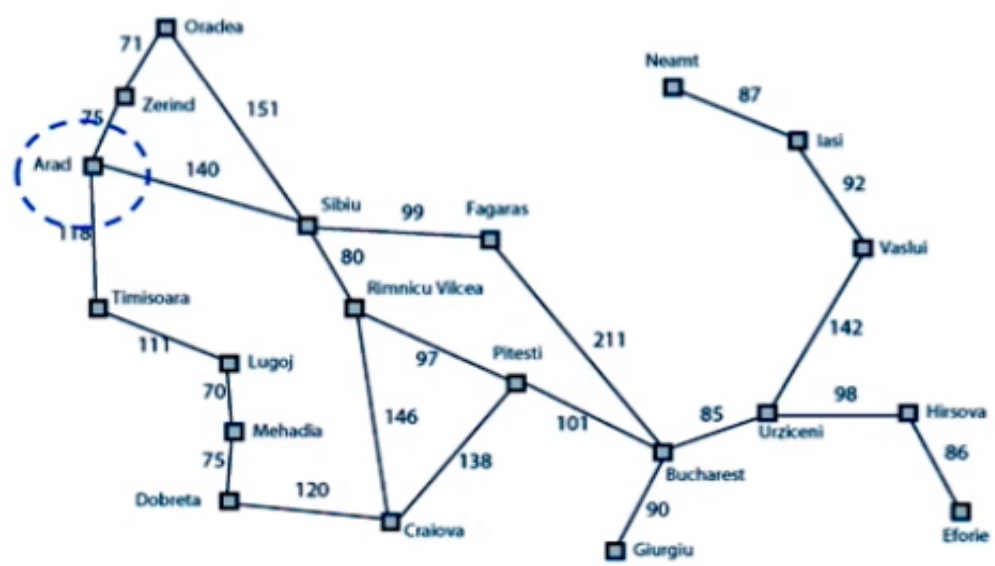
- A* Search minimizes $f(n) = g(n) + h(n)$
  - idea: preserve efficiency of Greedy Search but avoid expanding paths that are already expensive

- Q: is A* Search optimal and complete ?

- A: Yes! provided $h()$ is admissible in the sense that it never overestimates the cost to reach the goal.

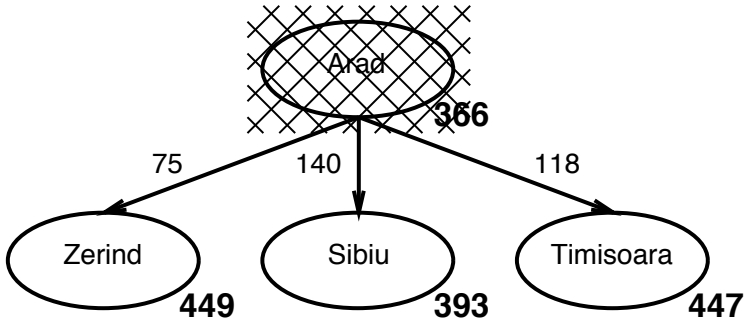
A* Search

- The SLD distances are underestimates
- Can not be a shorter path then the SLD in real life

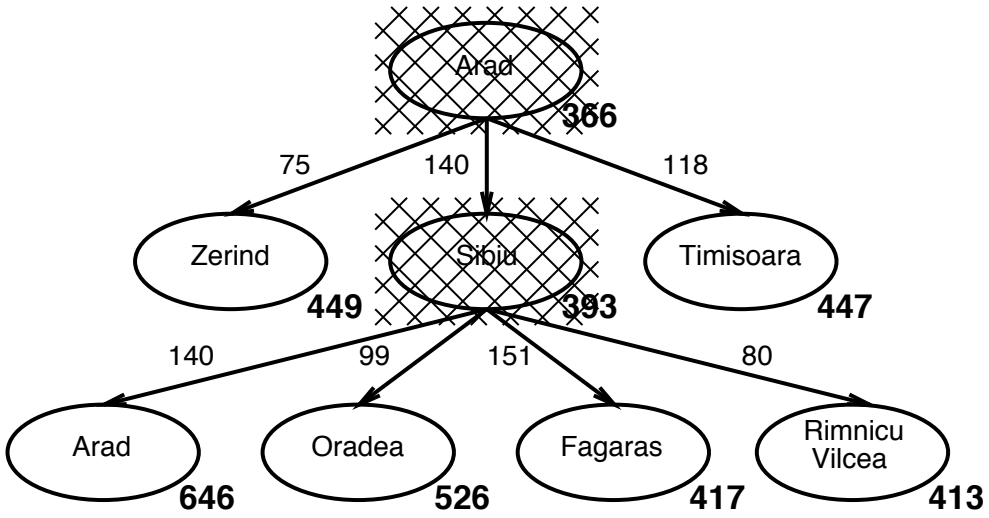


Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

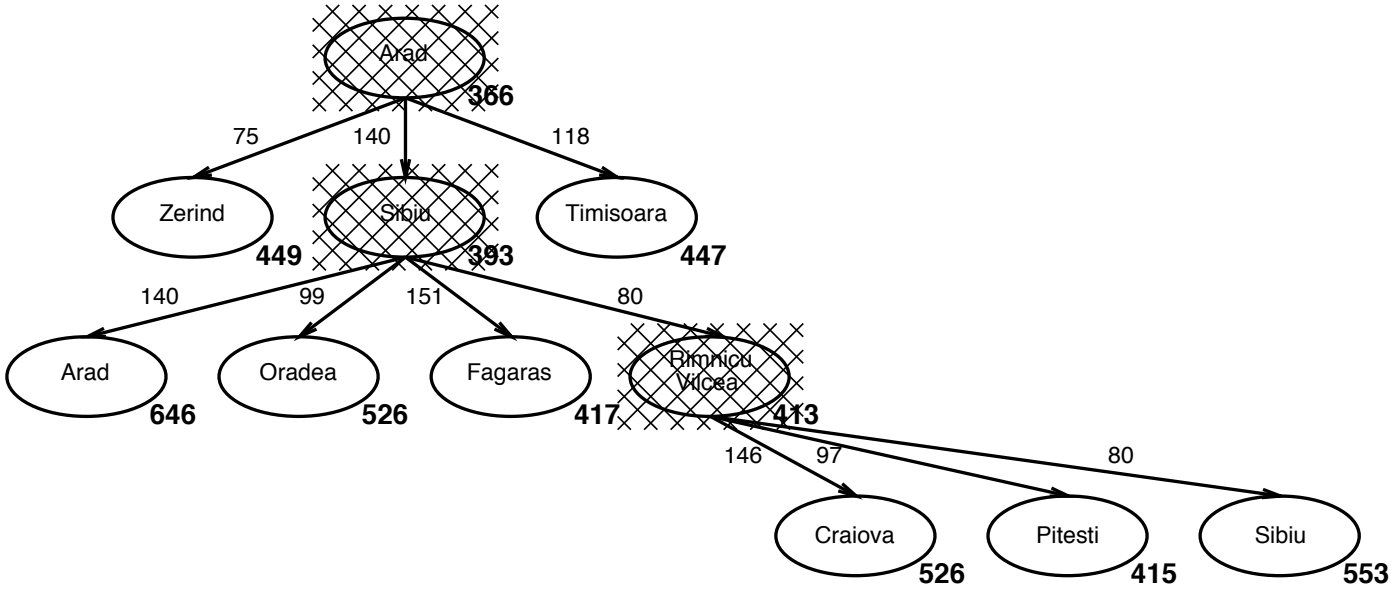
A* Search Example



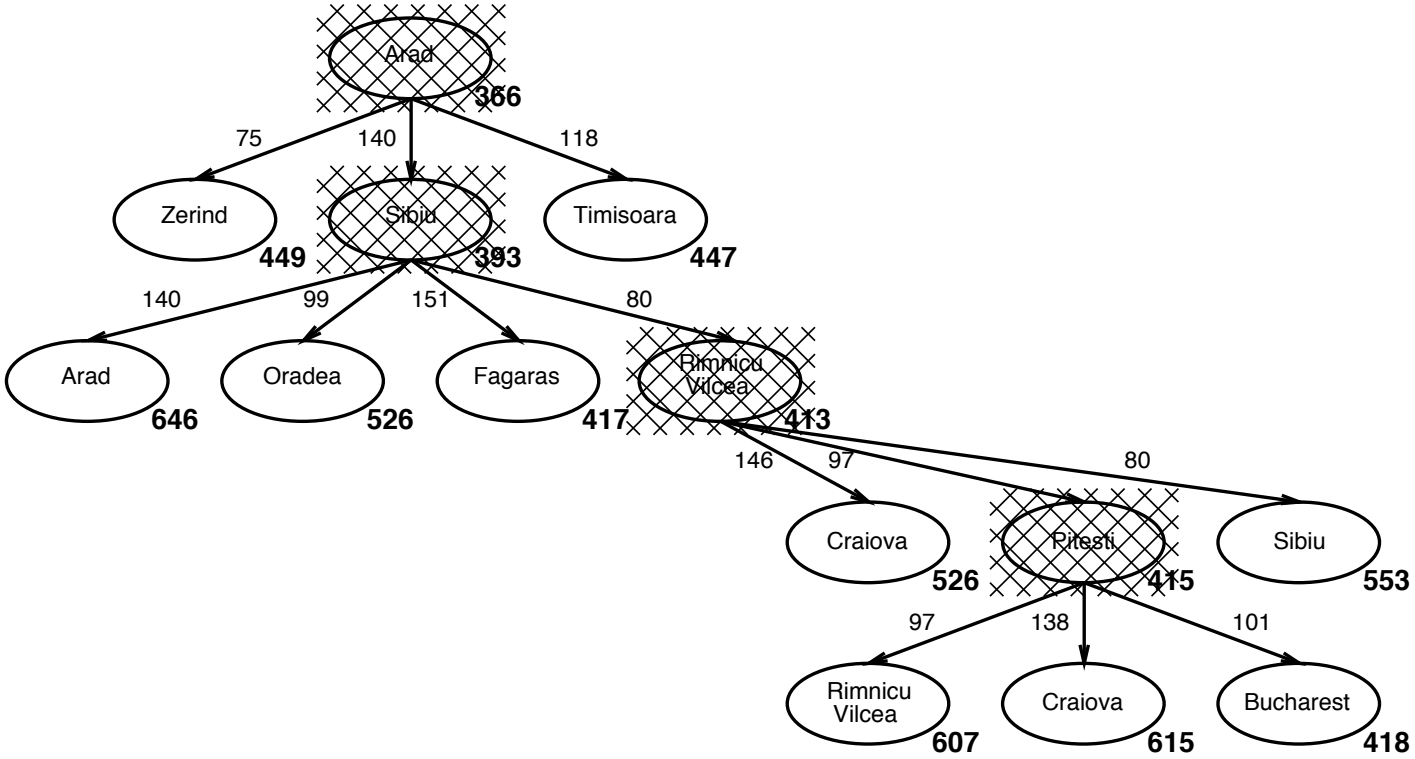
A* Search Example



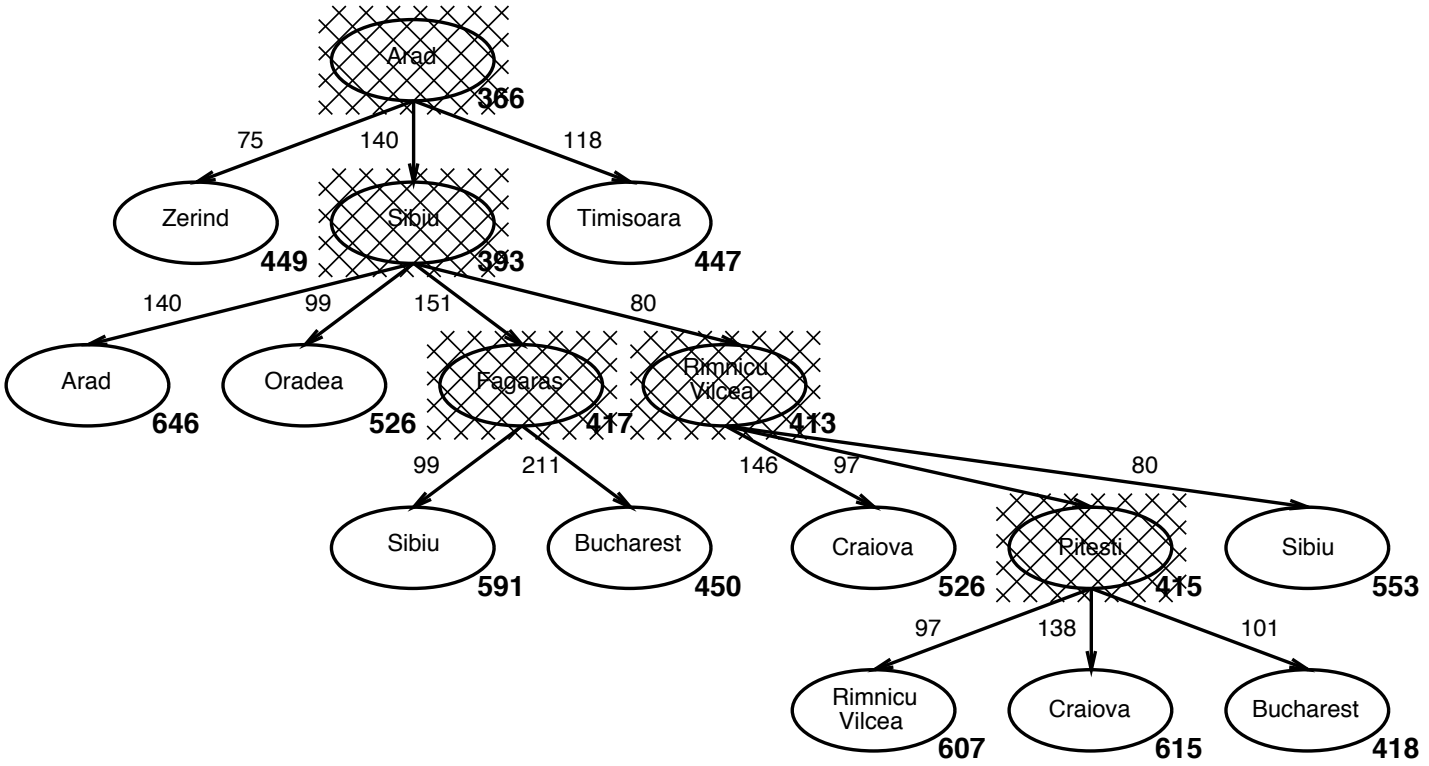
A* Search Example



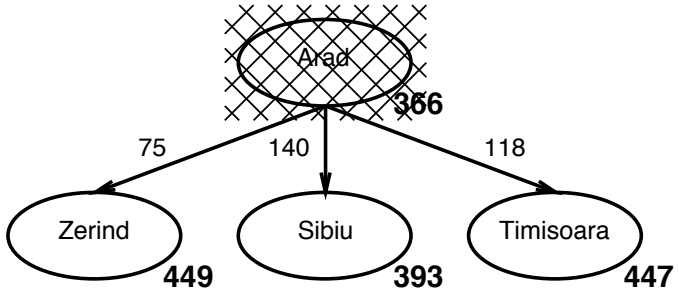
A* Search Example

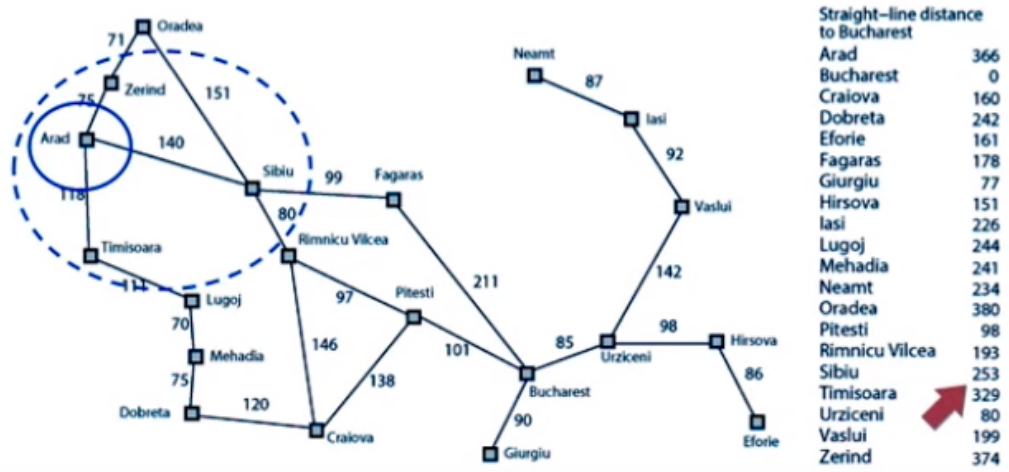


A* Search Example

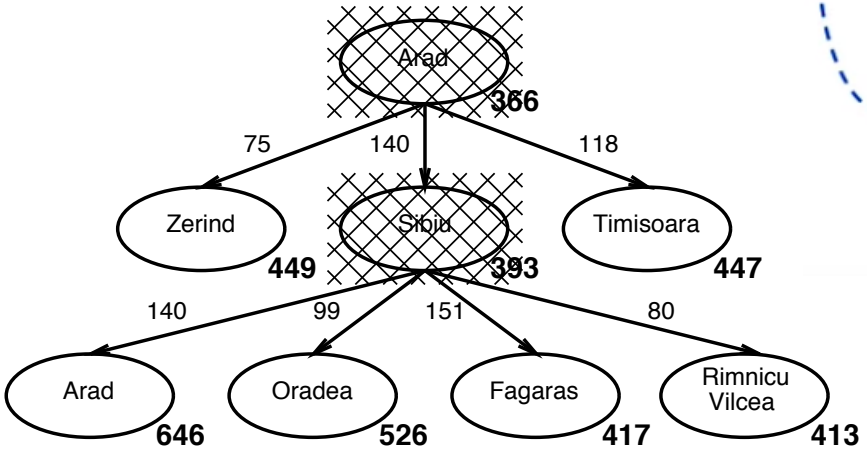


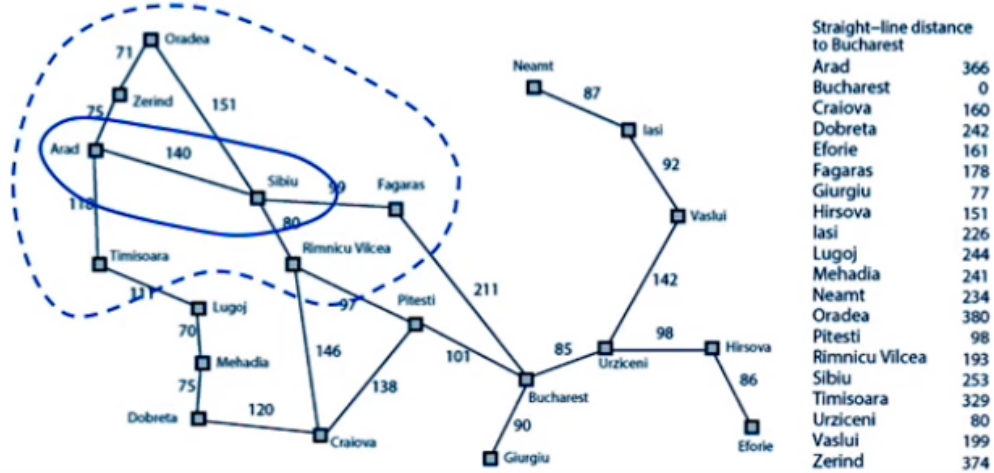
A* Search Example





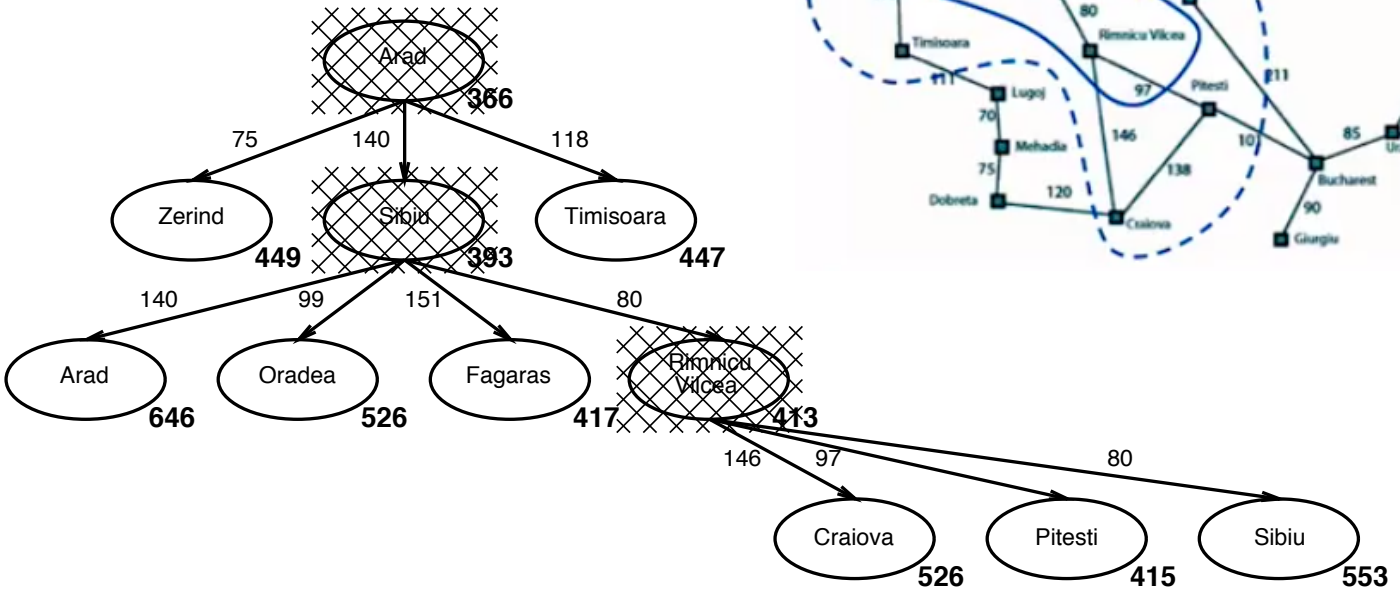
A* Search Example

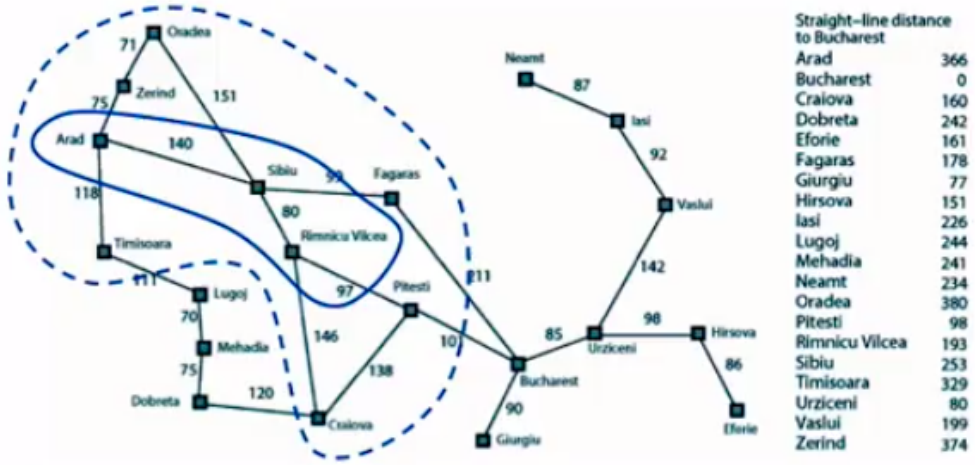




Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

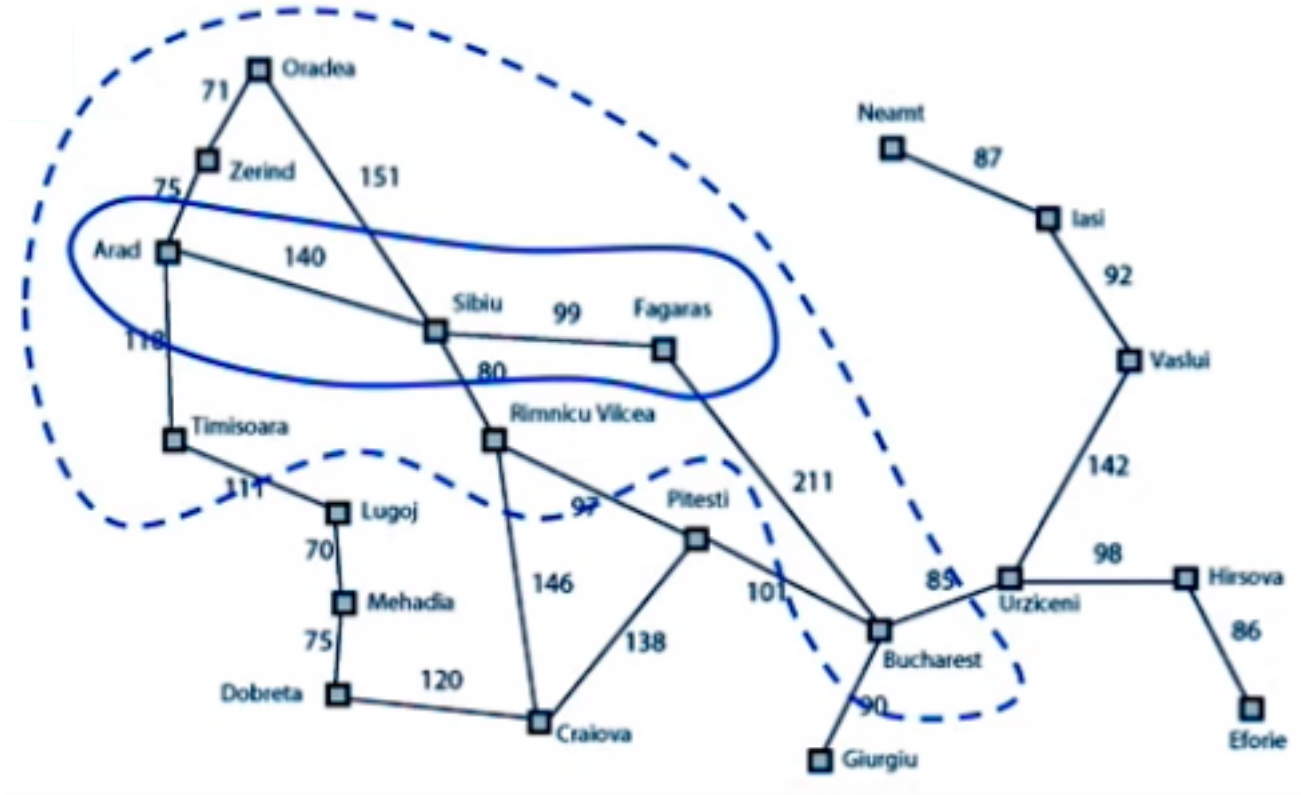
A* Search Example





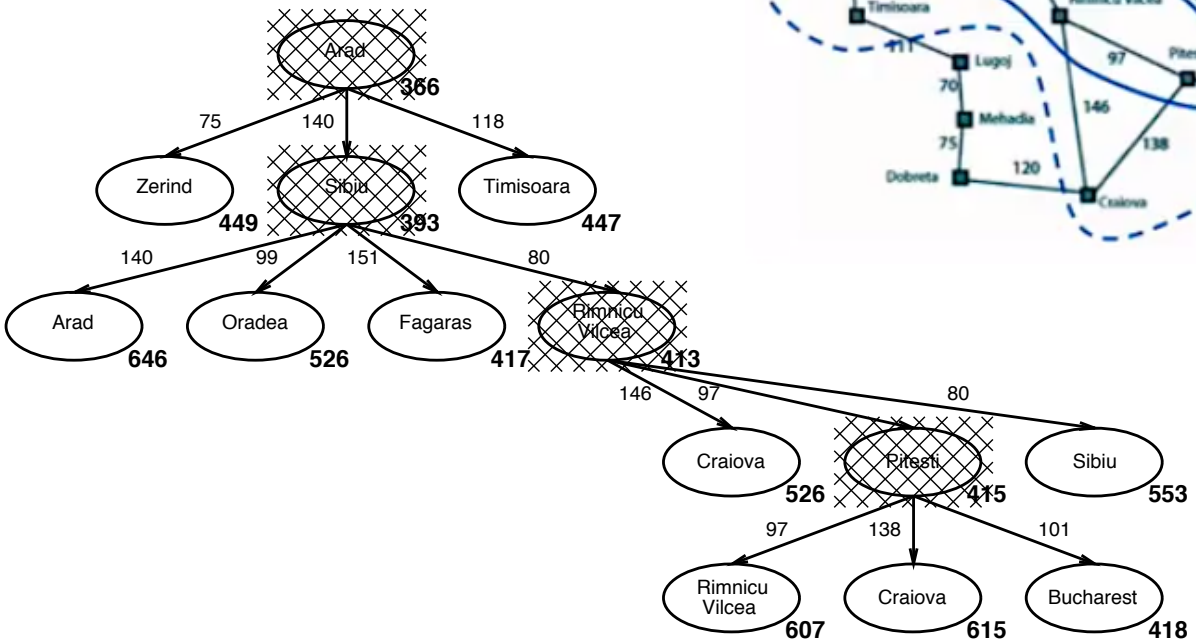
Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

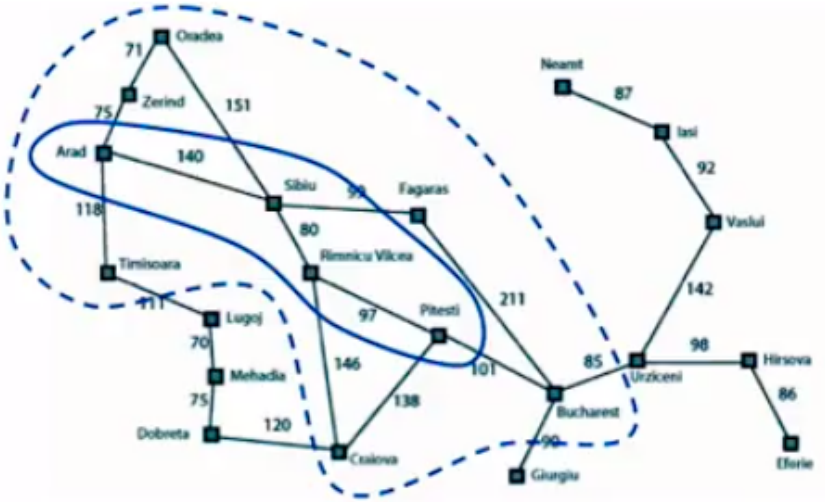
A* Search Example A* Search Example



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

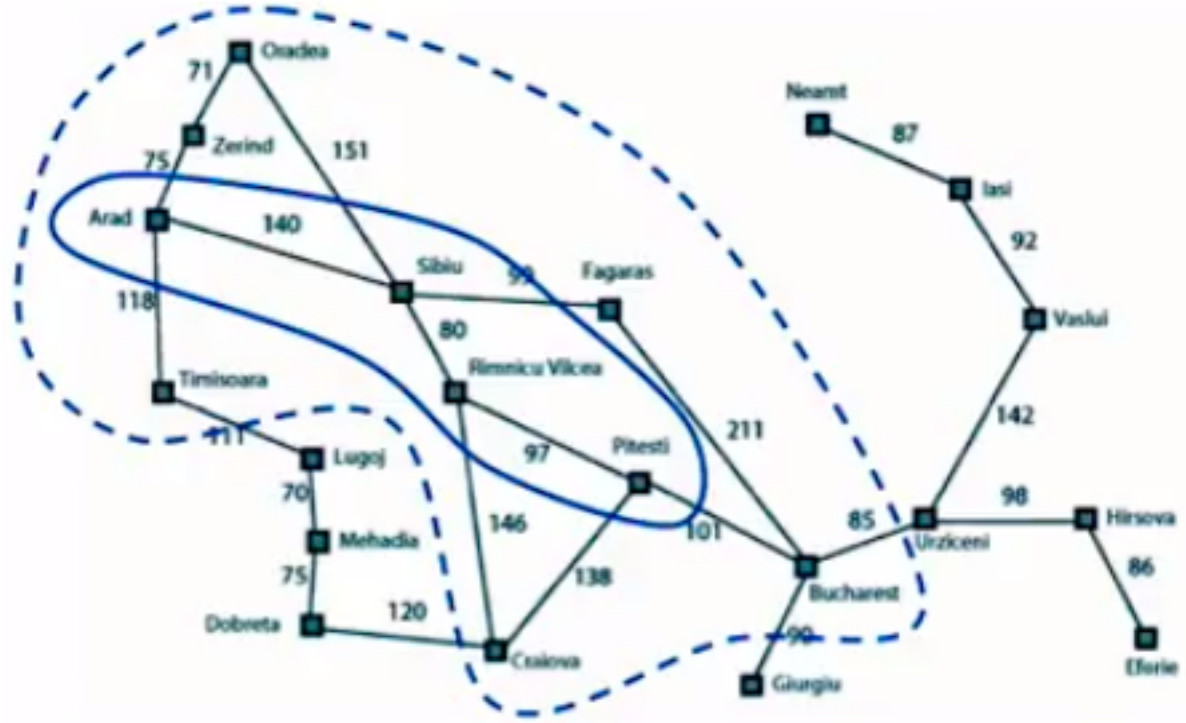
A* Search Example





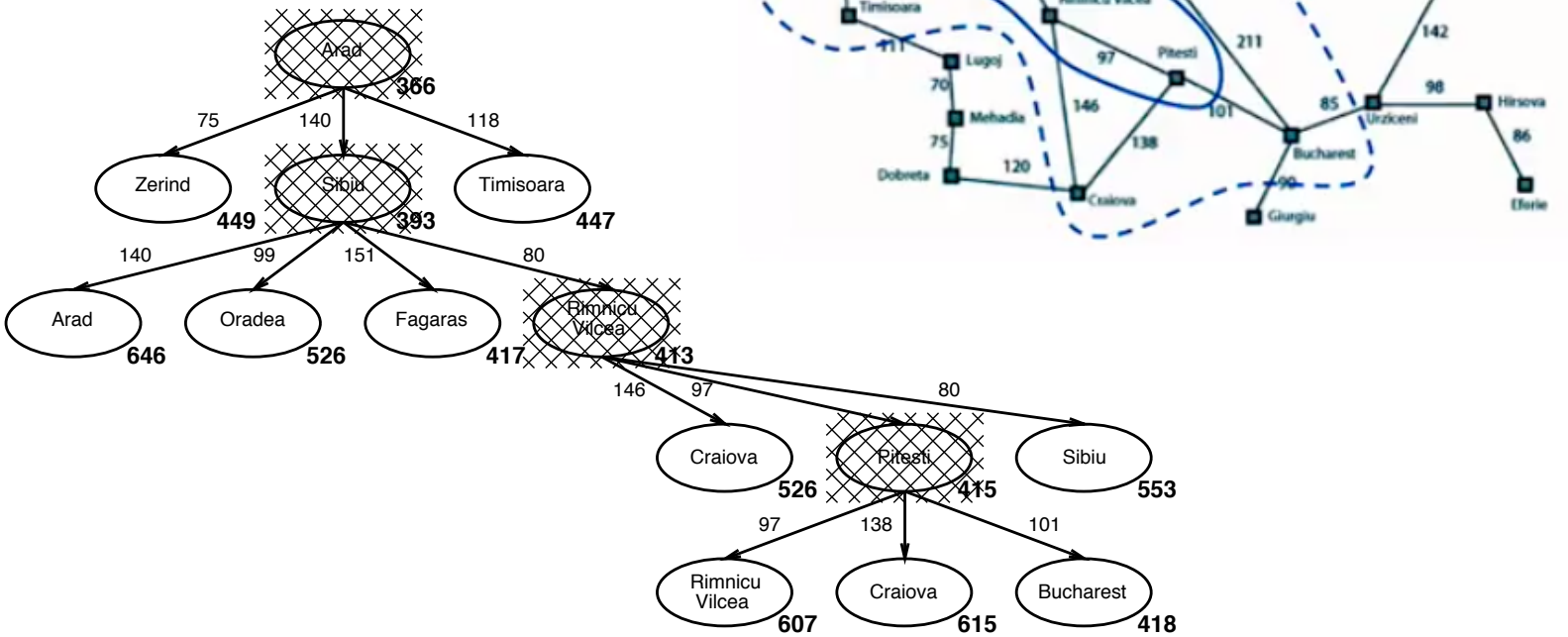
Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

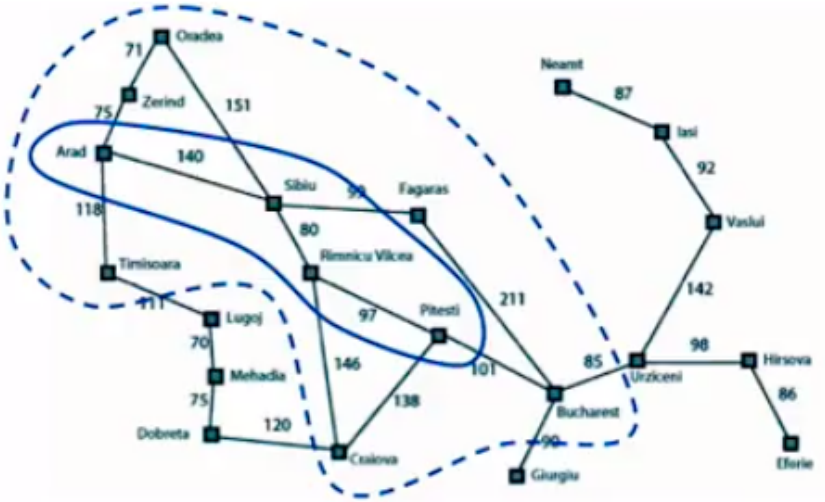
A* Search Example



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 Example

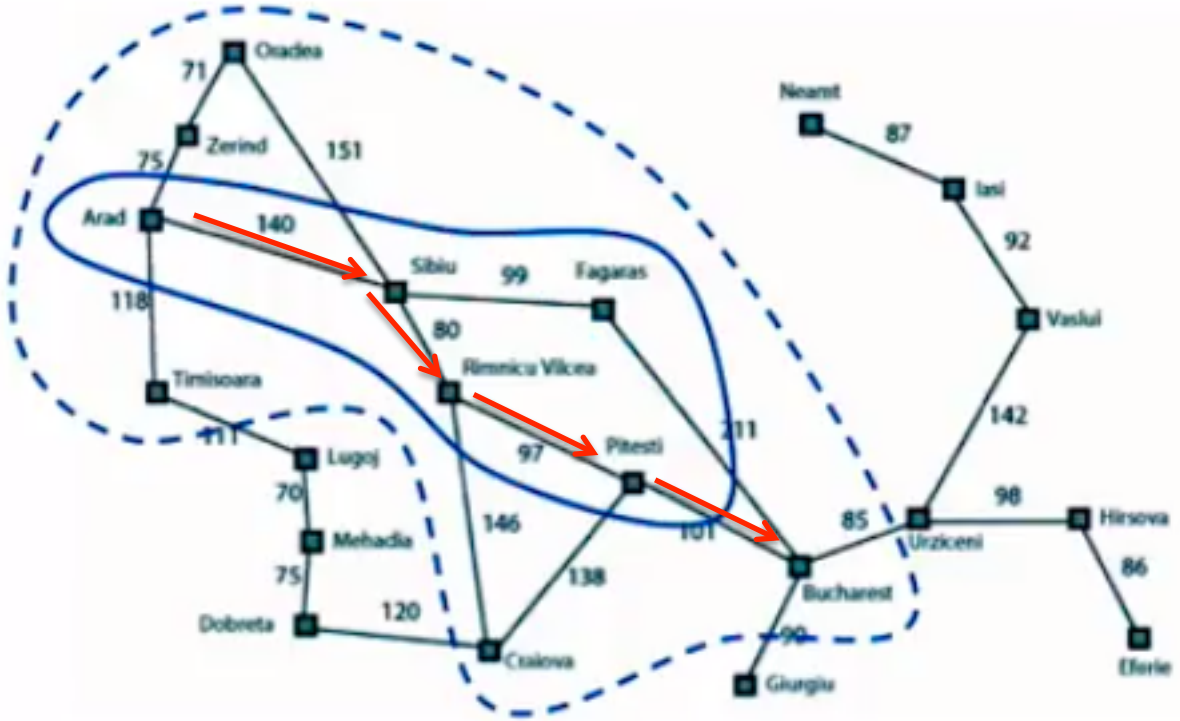




Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Will stop finding the shorter path to Buchurest

A* Search Example



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

Will stop finding the shorter path to Buchurest

A* Search

- Heuristic $h()$ is called admissible if

  $\forall n\ h(n) \leq h^*(n)$ where $h^*(n)$ is true cost from n to goal

- If h is admissible then $f(n)$ never overestimates the actual cost of the best solution through n.

- Example: $h_{SLD}()$ is admissible because the shortest path between any two points is a line.

- Theorem: A* Search is optimal if $h()$ is admissible.

Conditions for optimality: Admissibility and consistency

- The first condition we require for optimality is that h(n) be an **admissible heuristic**.
- An admissible heuristic is one that *never overestimates* the cost to reach the goal.
  - Because g(n) is the actual cost to reach n along the current path, and

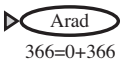
  f (n) = g(n) + h(n), we have as an immediate consequence that f(n) never overestimates the true cost of a solution along the current path through n.
- Admissible heuristics are by nature optimistic because they think the cost of solving the problem is less than it actually is.
  - example of an admissible heuristic is the straight-line distance h_{SLD} that we used in getting to Bucharest.

Admissibility

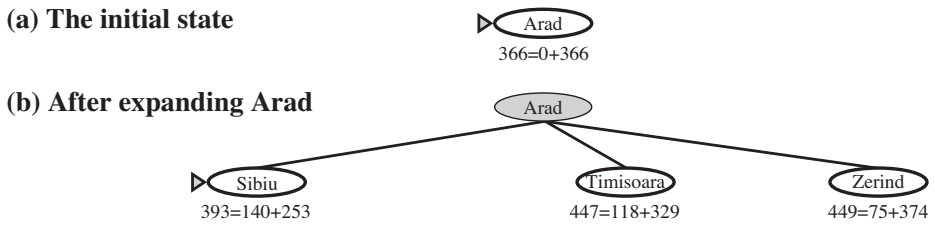
- Straight-line distance is admissible because the shortest path between any two points is a straight line, so the straight line cannot be an overestimate.

A* Search Example

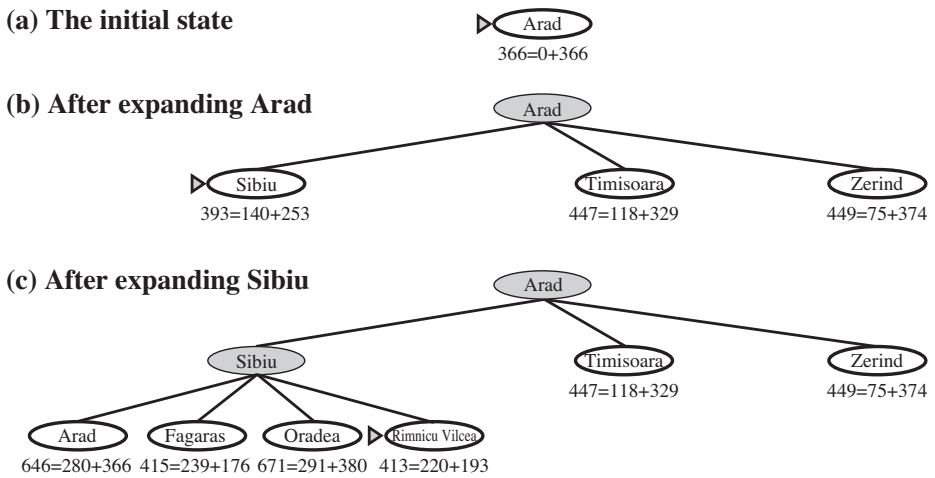
(a) The initial state



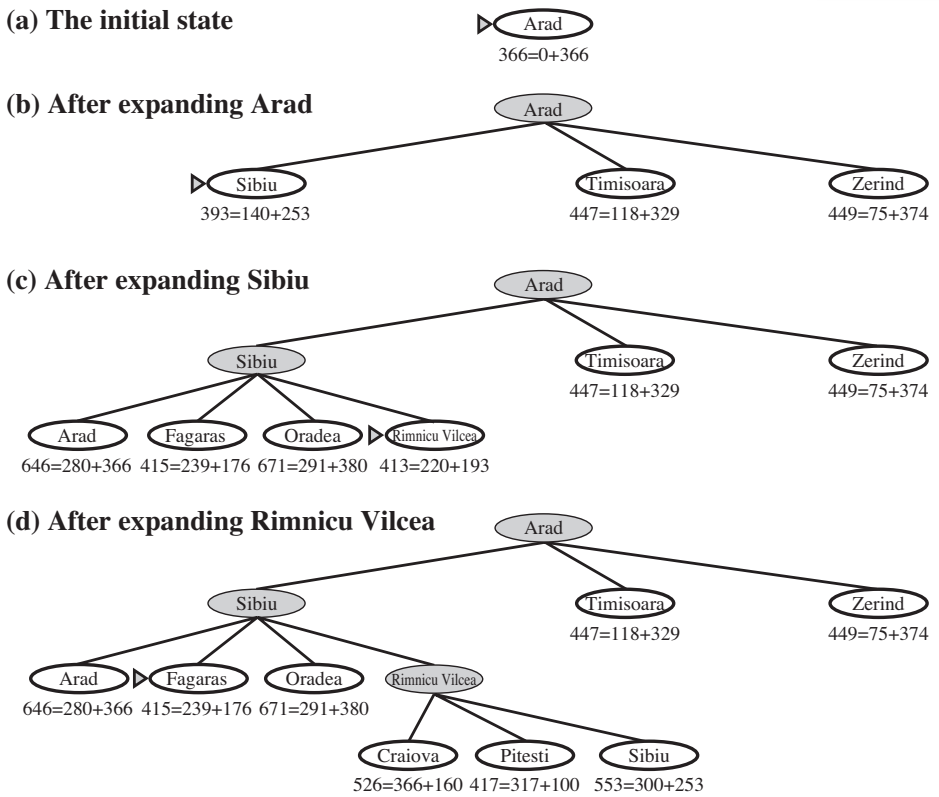
A* Search Example



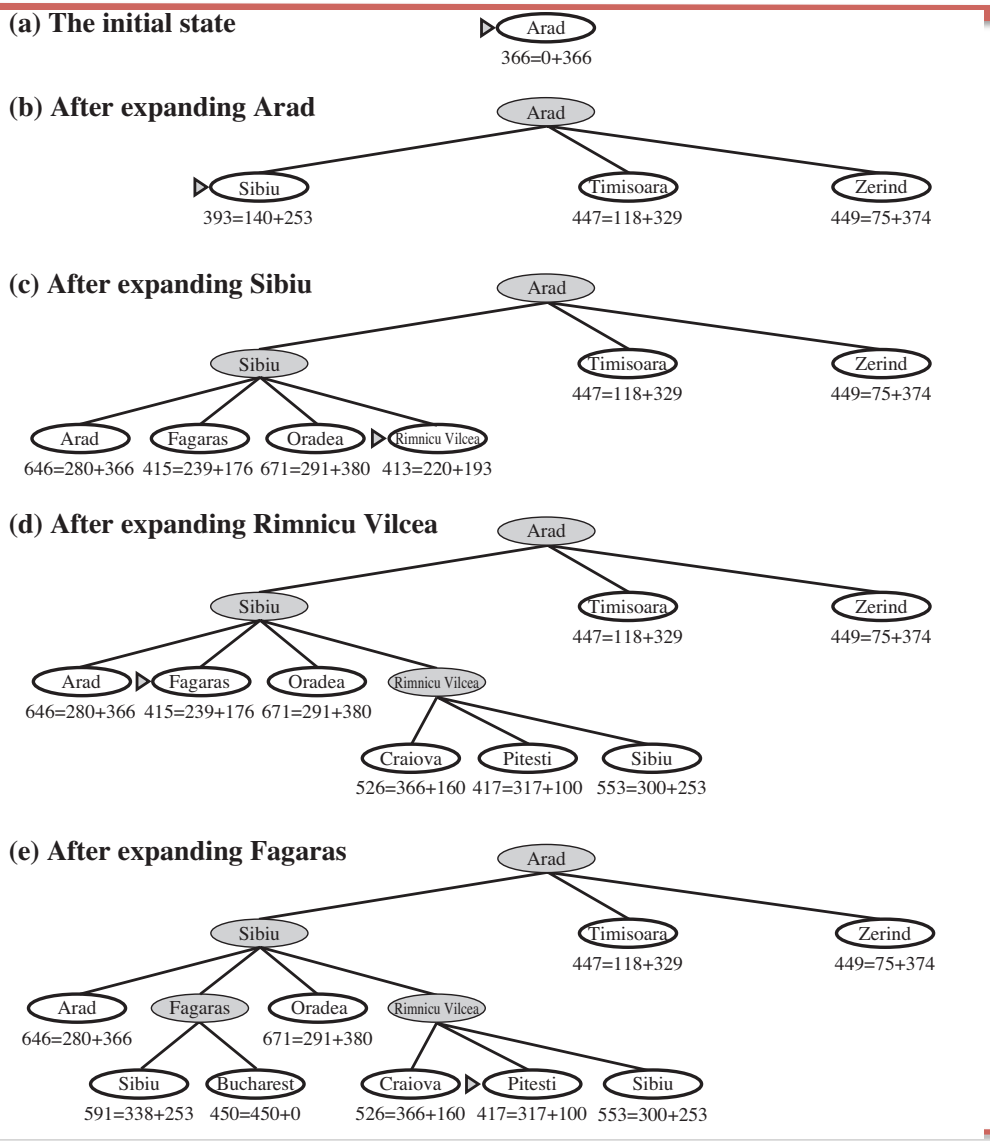
A* Search Example



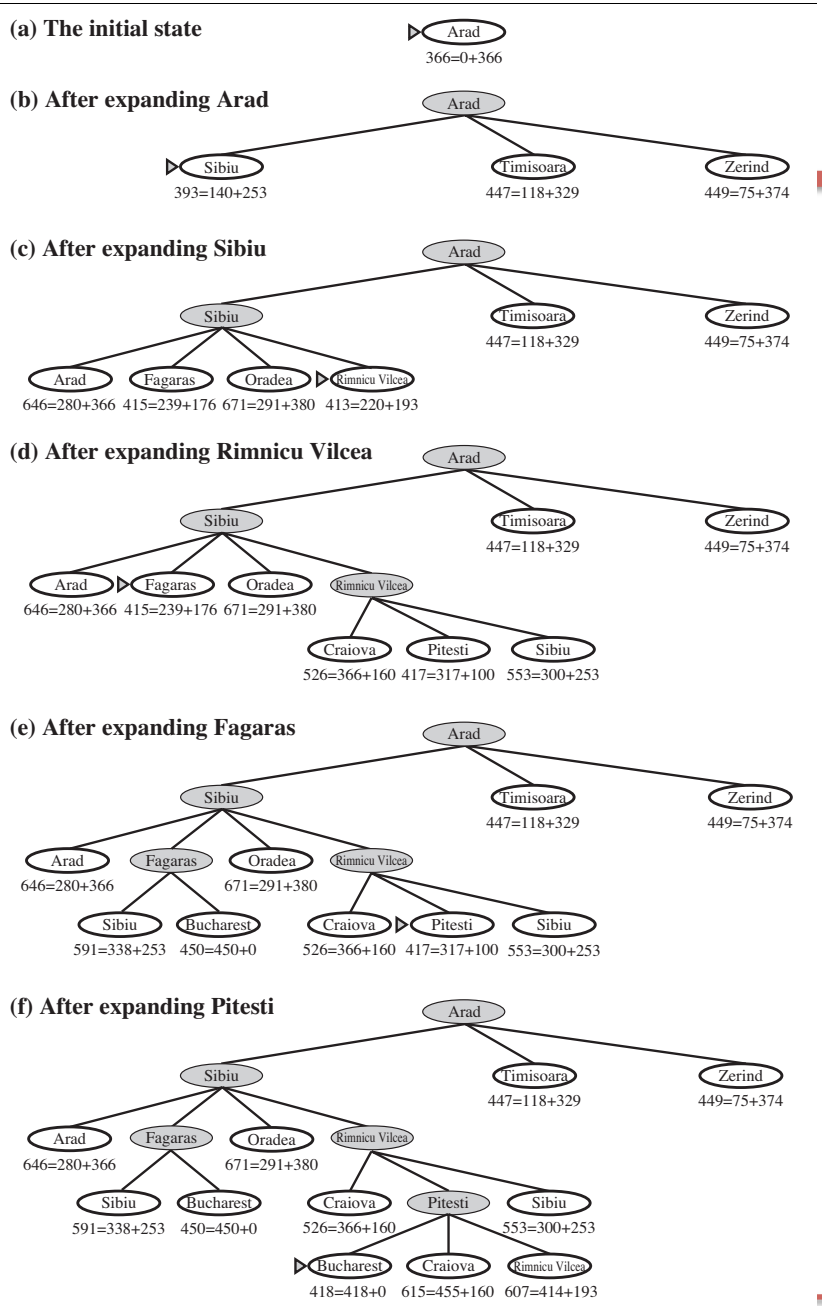
A* Search Example



A* Search Example

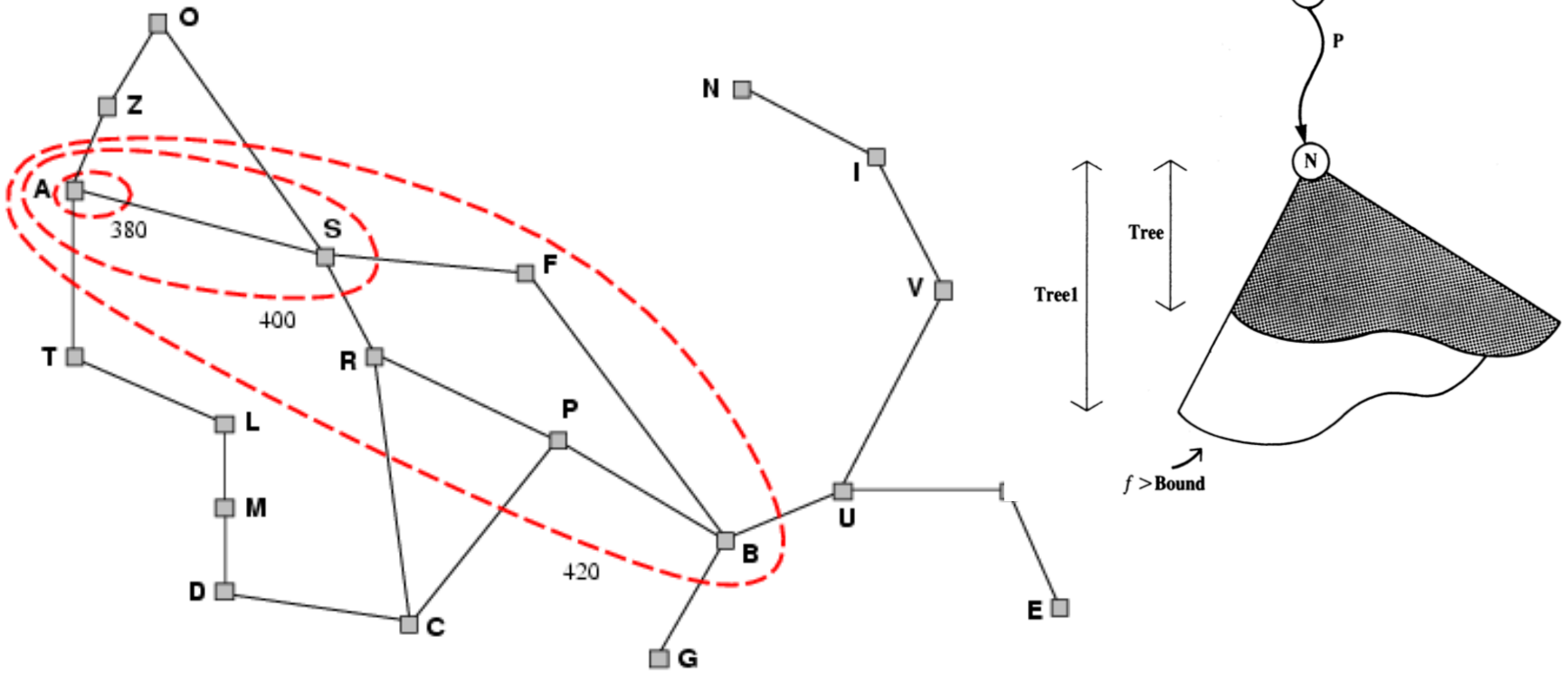


A* Search Example



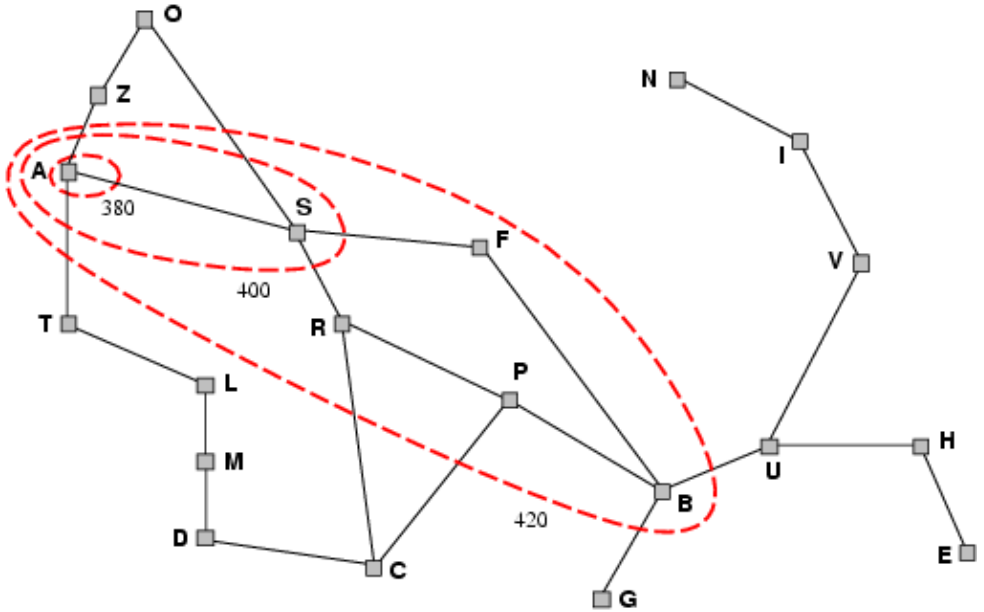
Optimality of A*

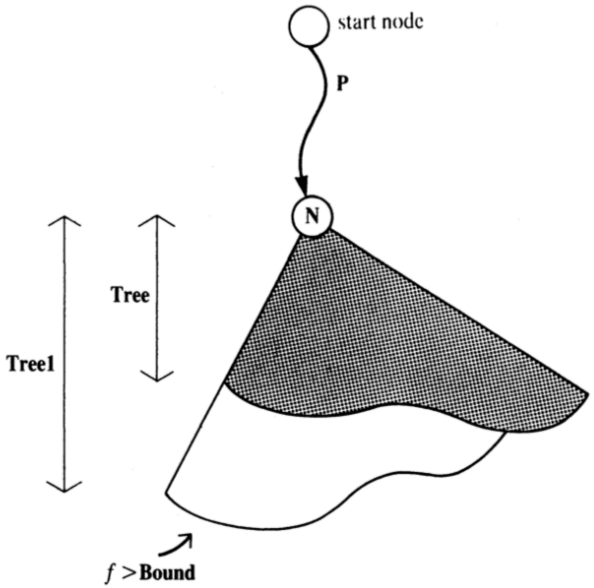
- A* expands nodes in order of increasing *f* value
- Gradually adds "*f*-contours" of nodes
- Contour *i* has all nodes with $f = f_i$, where $f_i < f_{i+1}$



Optimality of A*

- 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 $f_i < f_{i+1}$

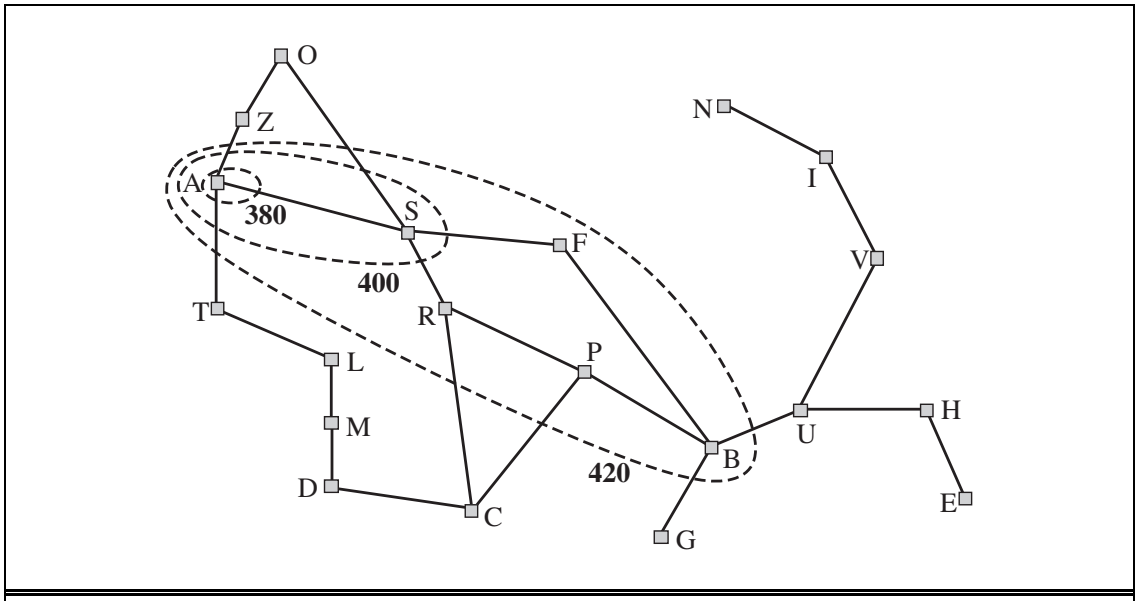




The fact that f-costs are nondecreasing along any path also means that we can draw **contours** in the state space, just like the contours in a topographic map

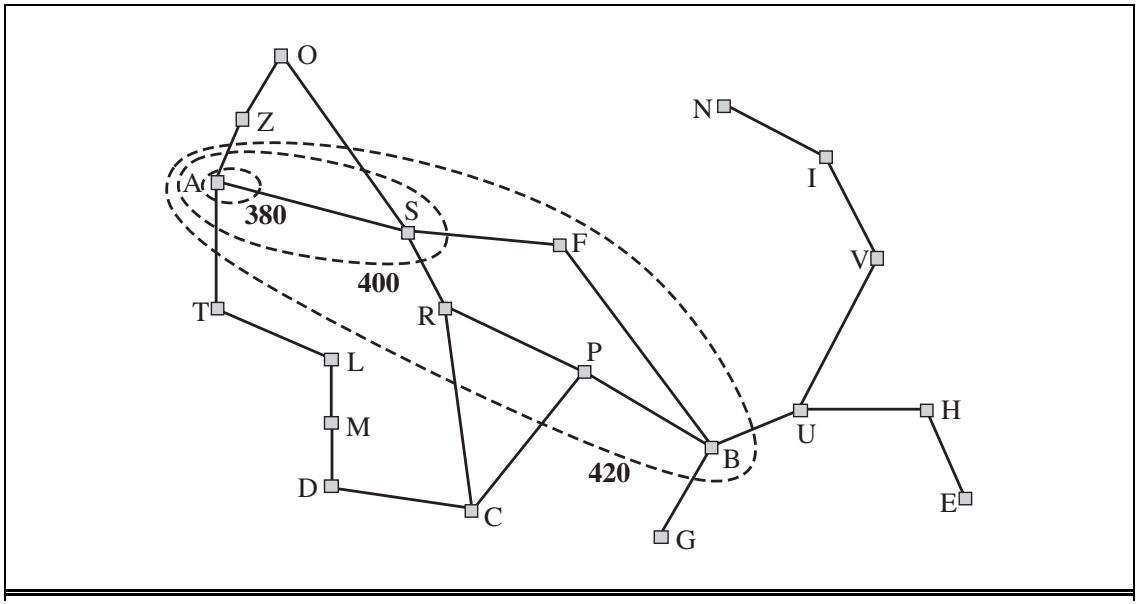
A* Search

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



A* Search

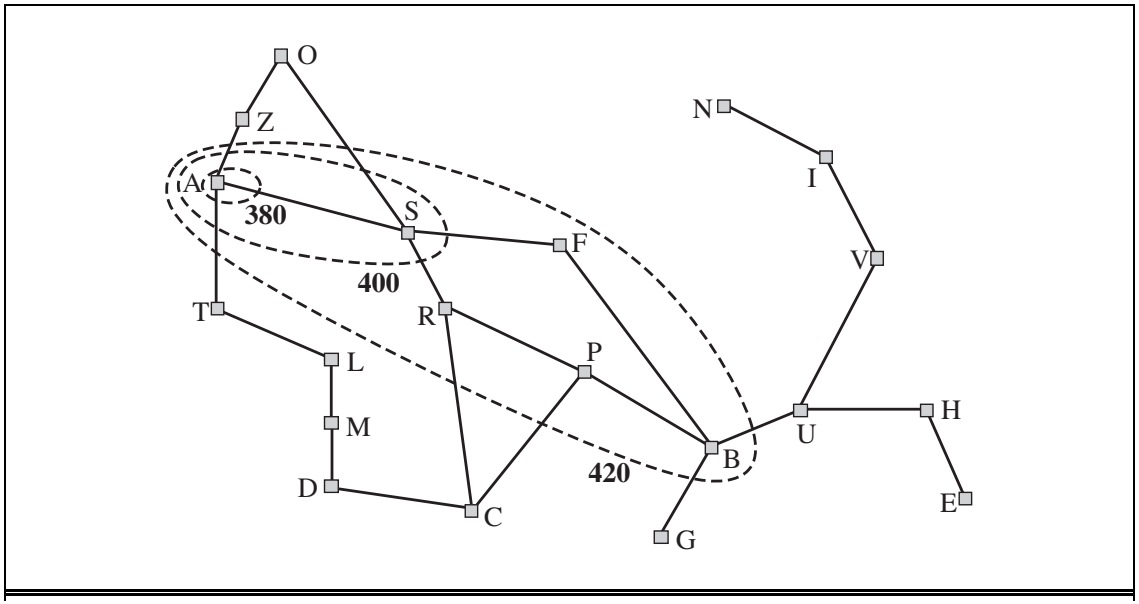
Map of Romania showing contours at f = 380, f = 400, and f = 420, with Arad as the start state. Nodes inside a given contour have f-costs less than or equal to the contour value.



Inside the contour labeled 400, all nodes have f(n) less than or equal to 400, and so on. Because A* expands the frontier node of lowest f-cost, we can see that an A* search fans out from the start node, adding nodes in concentric bands of increasing f-cost.

A* Search

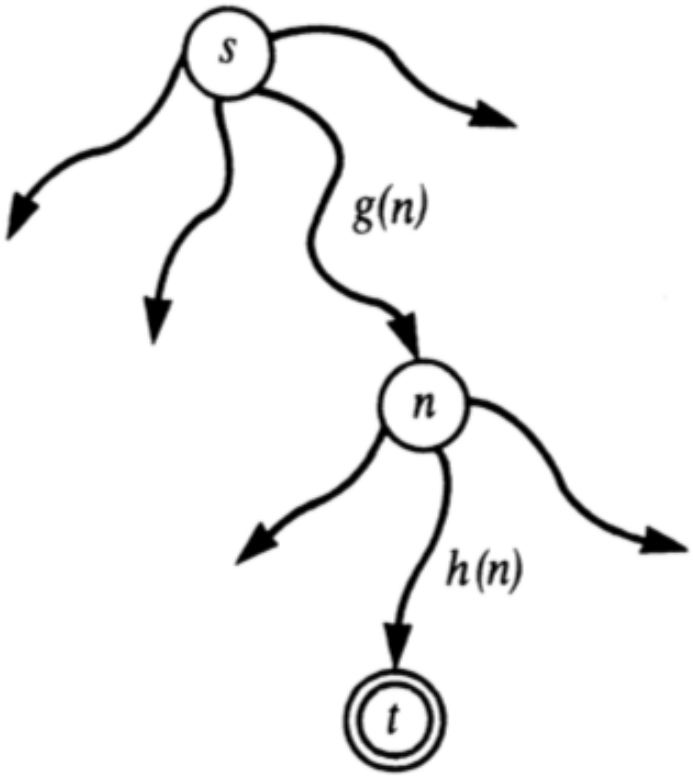
Map of Romania showing contours at f = 380, f = 400, and f = 420, with Arad as the start state. Nodes inside a given contour have f-costs less than or equal to the contour value.



Inside the contour labeled 400, all nodes have f(n) less than or equal to 400, and so on. Because A* expands the frontier node of lowest f-cost, we can see that an A* search fans out from the start node, adding nodes in concentric bands of increasing f-cost.

With uniform-cost search (A* search using h(n) = 0), the bands will be “circular” around the start state. With more accurate heuristics, the bands will stretch toward the goal state and become more narrowly focused around the optimal path.

Heuristic function



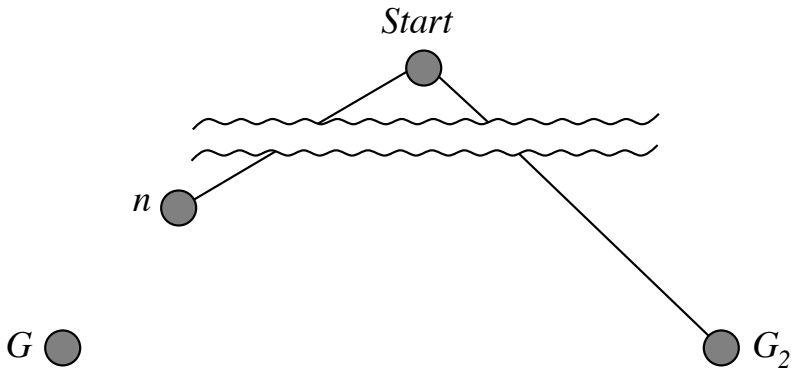
Heuristic estimate *f(n) of*
the cost of the cheapest paths from ***s*** to ***t***
via ***n***: *f(n) = g(n)+ h(n)*

g(n) is an estimate of the cost of an optimal path from s to n

h(n) is an estimate of the cost of an optimal path from n to t.

Optimality of A* Search

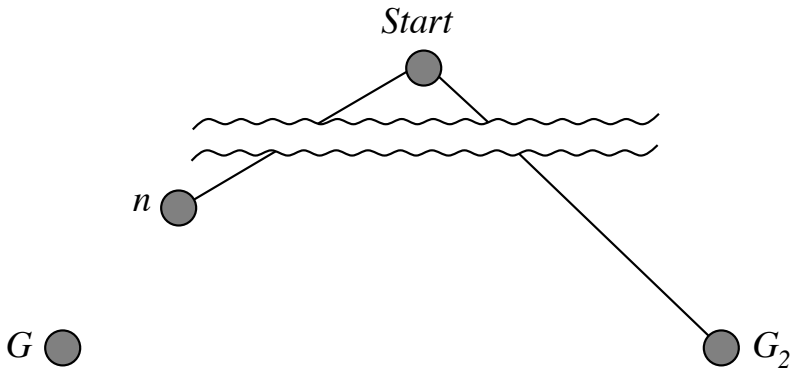
Suppose a suboptimal goal node *G*2 has been generated and is in the queue. Let *n* be the last unexpanded node on a shortest path to an optimal goal node *G*.



$$
\begin{aligned}
f(G_2) &= g(G_2) && \text{since } h(G_2) = 0 \\
&> g(G) && \text{since } G_2 \text{ is suboptimal} \\
&\geq f(n) && \text{since } h \text{ is admissible.}
\end{aligned}
$$

Optimality of A* Search

Suppose a suboptimal goal node *G*2 has been generated and is in the queue. Let *n* be the last unexpanded node on a shortest path to an optimal goal node *G*.



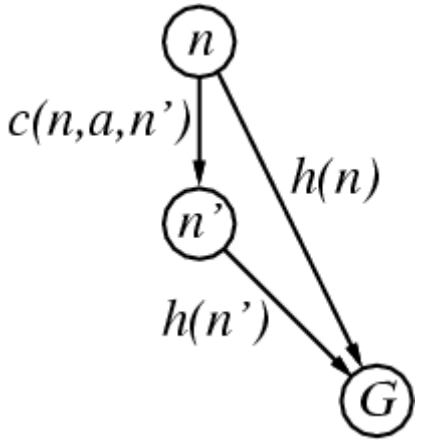
$$
\begin{aligned}
f(G_2) &= g(G_2) && \text{since } h(G_2) = 0 \\
&> g(G) && \text{since } G_2 \text{ is suboptimal} \\
&\geq f(n) && \text{since } h \text{ is admissible.}
\end{aligned}
$$

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

Consistent heuristics

- A heuristic is consistent if for every node *n*, every successor *n'* of *n* generated by any action *a*,

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



- If *h* is consistent, we have

  f(n') = g(n') + h(n')
  = g(n) + c(n,a,n') + h(n')
  ≥ g(n) + h(n)
  = f(n)

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

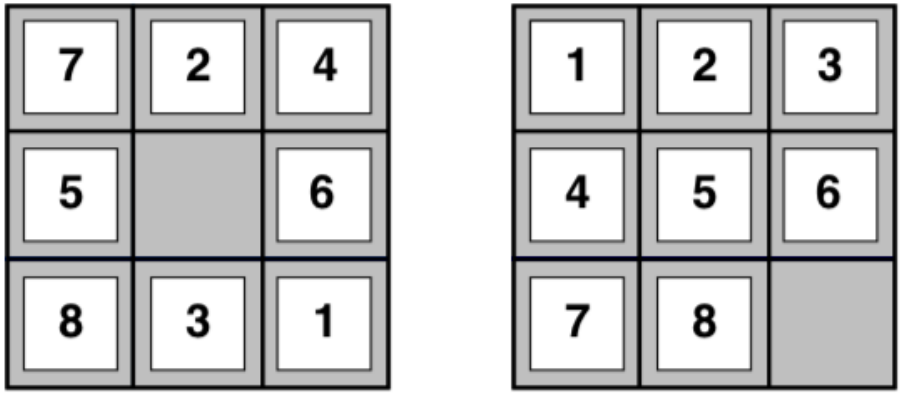
- Theorem: If *h(n)* is consistent, A* using GRAPH–SEARCH is optimal

Examples of Admissible Heuristics

e.g. for the 8-puzzle:

$h1(n)$ = total number of misplaced tiles

$h2(n)$ = total Manhattan distance = ∑distance from goal position



$h1(S)$ = ?

$h2(S)$ = ?

- Why are $h1$, $h2$ admissible?

Optimality of A* Search

- Since $f(G2) > f(n)$, A* will never select $G2$ for expansion.
- Note: suboptimal goal node $G2$ may be generated, but it will never be expanded.
- In other words, even after a goal node has been generated, A* will keep searching so long as there is a possibility of finding a shorter solution.
- Once a goal node is selected for expansion, we know it must be optimal, so we can terminate the search.

Properties of A* Search

- Complete: Yes, unless there are infinitely many nodes with $f \leq$ cost of solution
- Time: Exponential in [relative error in h× length of solution]
- Space: Keeps all nodes is memory
- Optimal: Yes (assuming $h()$ is admissible).

Iterative Deepening A* Search

- Iterative Deepening A* is a low-memory variant of A* which performs a series of depth-first searches, but cuts off each search when the sum f () = g() + h() exceeds some pre-defined threshold.

- The threshold is steadily increased with each successive search.

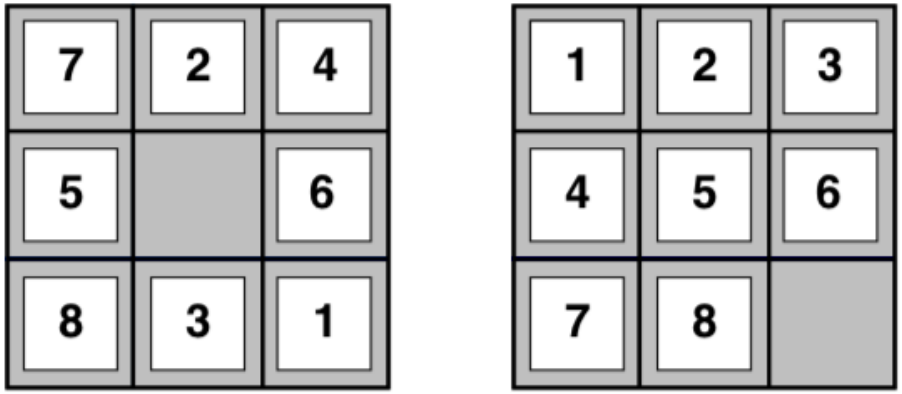
- IDA* is asymptotically as efficient as A* for domains where the number of states grows exponentially.

Examples of Admissible Heuristics

e.g. for the 8-puzzle:

$h1(n)$ = total number of misplaced tiles

$h2(n)$ = total Manhattan distance = ∑distance from goal position



$h1(S)$ = ?

$h2(S)$ = ?

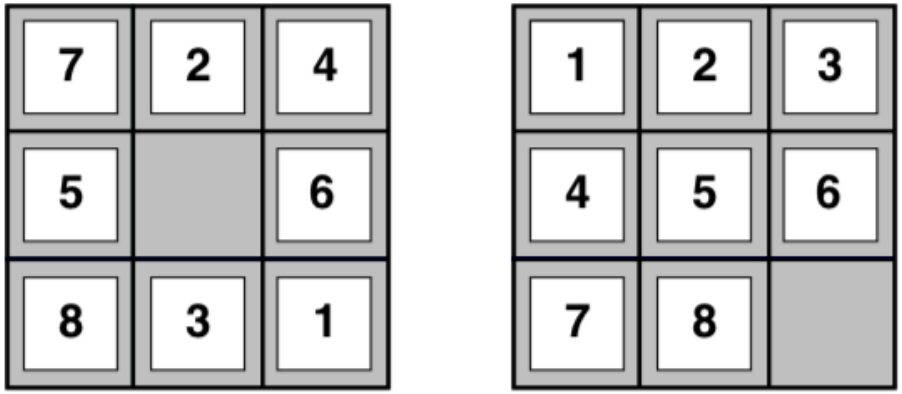
- Why are $h1$, $h2$ admissible?

Examples of Admissible Heuristics

e.g. for the 8-puzzle:

$h1(n)$ = total number of misplaced tiles

$h2(n)$ = total Manhattan distance = ∑distance from goal position



Start State Goal State

$h1(S) = 6$

$h2(S) = 4+0+3+3+1+0+2+1 = 14$

- $h1$: every tile must be moved at least once.
- $h2$: each action can only move one tile one step closer to the goal.

Dominance

- if $h2(n) \geq h1(n)$ for all n (both admissible) then $h2$ dominates $h1$ and is better for search. So the aim is to make the heuristic $h()$ as large as possible, but without exceeding $h^*()$.
- typical search costs:

14-puzzle	IDS	$= 3{,}473{,}941$ nodes
	$A^*(h_1)$	$= 539$ nodes
	$A^*(h_2)$	$= 113$ nodes
24-puzzle	IDS	$\approx 54 \times 10^9$ nodes
	$A^*(h_1)$	$= 39{,}135$ nodes
	$A^*(h_2)$	$= 1{,}641$ nodes

How to Find Heuristic Functions ?

- Admissible heuristics can often be derived from the exact solution cost of a simplified or "relaxed" version of the problem. (i.e. with some of the constraints weakened or removed)
  - If the rules of the 8-puzzle are relaxed so that a tile can move anywhere, then $h1(n)$ gives the shortest solution.
  - If the rules are relaxed so that a tile can move to any adjacent square, then $h2(n)$ gives the shortest solution.

Composite Heuristic Functions

- Let $h_1, h_2, \ldots, h_m$ be admissible heuristics for a given task.
- Define the composite heuristic

$$h(n) = \max(h_1(n), h_2(n), \ldots, h_m(n))$$

- h is admissible
- h dominates $h_1, h_2, \ldots, h_m$

Memory-bounded heuristic search

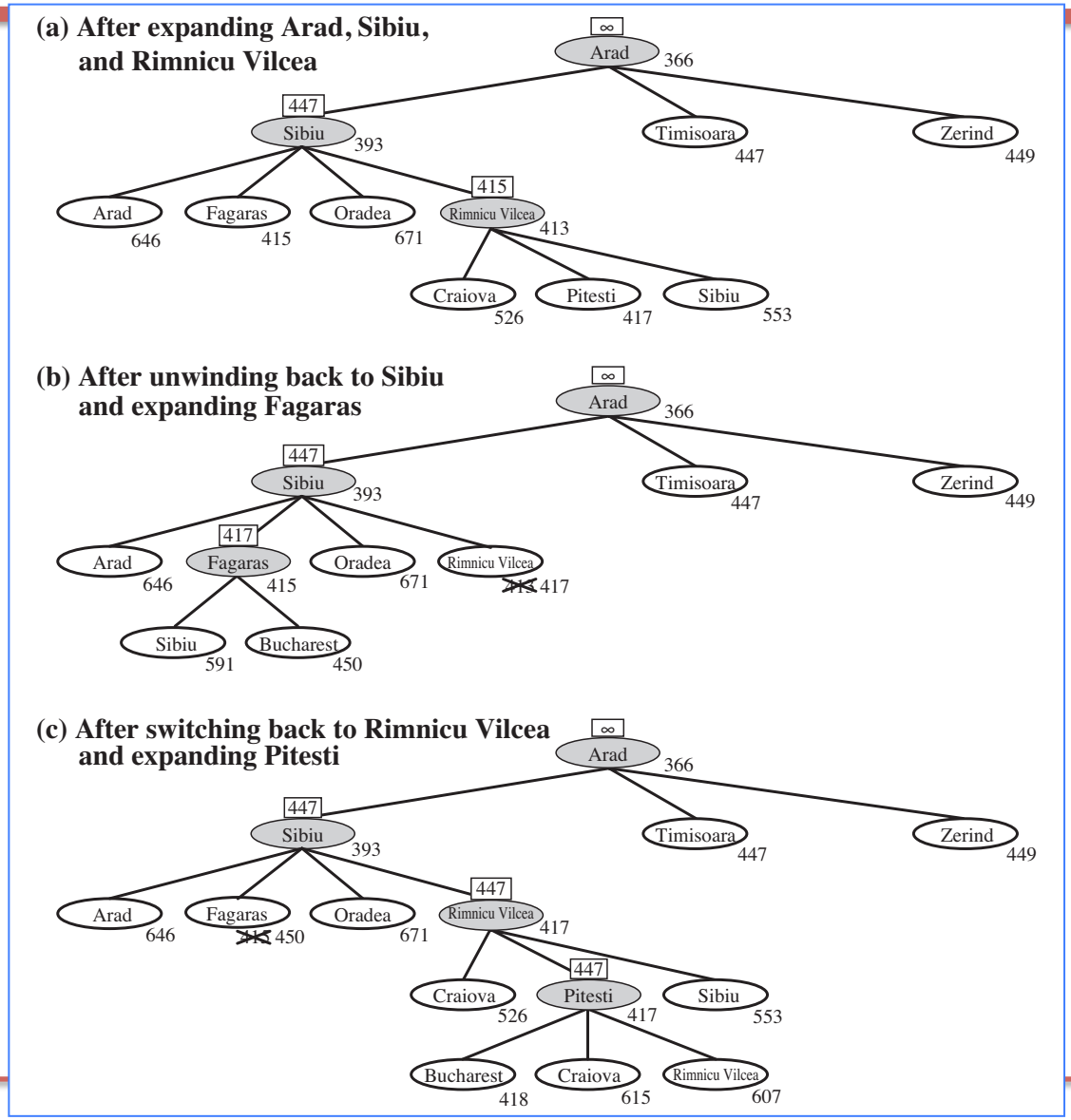
Recursive best-first search

- **Recursive best-first search** (RBFS) is a simple recursive algorithm that attempts to mimic the operation of standard best-first search, but using only linear space.
- Its structure is similar to that of a recursive depth-first search, but 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 of the current node.
- 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.
- In this way, RBFS remembers the f-value of the best leaf in the forgotten subtree and can therefore decide whether it's worth reexpanding the subtree at some later time.

Memory-bounded heuristic search

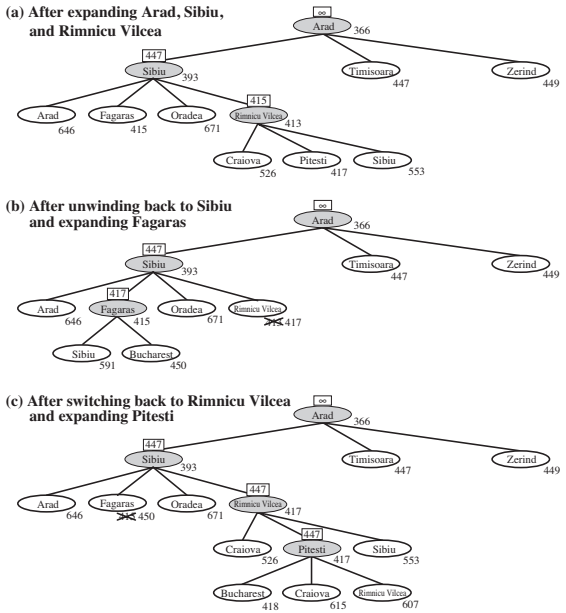
- The simplest way to reduce memory requirements for A* is to adapt the idea of iterative deepening to the heuristic search context, resulting in the **iterative-deepening A*** (IDA*) al- gorithm.
- The main difference between IDA* and standard iterative deepening is that the cutoff used is the f -cost (g + h) rather than the depth;
- at each iteration, the cutoff value is the smallest f-cost of any node that exceeded the cutoff on the previous iteration. IDA* is practical for many problems with unit step costs and avoids the substantial overhead associated with keeping a sorted queue of nodes. Unfortunately, it suffers from the same difficulties with real- valued costs as does the iterative version of uniform-cost search described in Exercise 3.17. This section briefly examines two other memory-bounded algorithms, called RBFS and MA*.

Recursive best-first search



Recursive best-first search

Stages in an RBFS search for the shortest route to Bucharest. The f-limit value for each recursive call is shown on top of each current node, and every node is labeled with its f -cost. (a) The path via Rimnicu Vilcea is followed until the current best leaf (Pitesti) has a value that is worse than the best alternative path (Fagaras). (b) The recursion unwinds and the best leaf value of the forgotten subtree (417) is backed up to Rimnicu Vilcea; then Fagaras is expanded, revealing a best leaf value of 450. (c) The recursion unwinds and the best leaf value of the forgotten subtree (450) is backed up to Fagaras; then Rimnicu Vilcea is expanded. This time, because the best alternative path (through Timisoara) costs at least 447, the expansion continues to Bucharest.



Properties of Recursive best-first search

- Like A* tree search, RBFS is an optimal algorithm if the heuristic function h(n) is admissible. Its space complexity is linear in the depth of the deepest optimal solution, but its time complexity is rather difficult to characterize: it depends both on the accuracy of the heuristic function and on how often the best path changes as nodes are expanded.

- they suffer the potentially exponential increase in complexity associated with redundant paths in graphs

Summary of Informed Search

- Heuristics can be applied to reduce search cost.
- Greedy Search tries to minimize cost from current node *n* to the goal.
- A* combines the advantages of Uniform-Cost Search and Greedy Search.
- A* is complete, optimal and optimally efficient among all optimal search algorithms.
- Memory usage is still a concern for A*. A* is a low-memory variant.

Summary

- Before an agent can start searching for solutions, a **goal** must be identified and a well- defined **problem** must be formulated.
- A problem consists of five parts: the **initial state**, a set of **actions**, a **transition model** describing the results of those actions, a **goal test** function, and a **path cost** function.
- The environment of the problem is represented by a **state space**.
- A **path** through the state space from the initial state to a goal state is a **solution**.

Summary

- Search algorithms treat states and actions as **atomic**: they do not consider any internal structure they might possess.
- A general TREE-SEARCH algorithm considers all possible paths to find a solution, whereas a GRAPH-SEARCH algorithm avoids consideration of redundant paths.
- Search algorithms are judged on the basis of **completeness**, **optimality**, **time complex- ity**, and **space complexity**. Complexity depends on b, the branching factor in the state space, and d, the depth of the shallowest solution.

Summary

- **Uninformed search** methods have access only to the problem definition. The basic algorithms are as follows:
  - **Breadth-first search** expands the shallowest nodes first; it is complete, optimal for unit step costs, but has exponential space complexity.
  - **Uniform-cost search** expands the node with lowest path cost, g(n), and is optimal for general step costs.
  - **Depth-first search** expands the deepest unexpanded node first. It is neither com- plete nor optimal, but has linear space complexity. **Depth-limited search** adds a depth bound.
  - **Iterative deepening search** calls depth-first search with increasing depth limits until a goal is found. It is complete, optimal for unit step costs, has time complexity comparable to breadth-first search, and has linear space complexity.
  - **Bidirectional search** can enormously reduce time complexity, but it is not always applicable and may require too much space.

Summary

- **Informed search** methods may have access to a **heuristic** function h(n) that estimates the cost of a solution from n.
  - The generic **best-first search** algorithm selects a node for expansion according to an **evaluation function**.
  - **Greedy best-first search** expands nodes with minimal h(n). It is not optimal but is often efficient.
  - **A∗ search** expands nodes with minimal f (n) = g(n) + h(n). A∗ is complete and optimal, provided that h(n) is admissible (for TREE-SEARCH) or consistent (for GRAPH-SEARCH). The space complexity of A∗ is still prohibitive. (**RBFS** (recursive best-first search))
- The performance of heuristic search algorithms depends on the quality of the heuristic function. One can sometimes construct good heuristics by relaxing the problem definition, by storing precomputed solution costs for subproblems in a pattern database, or by learning from experience with the problem class.