Informed search algorithms

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

Outline

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

Best-first search

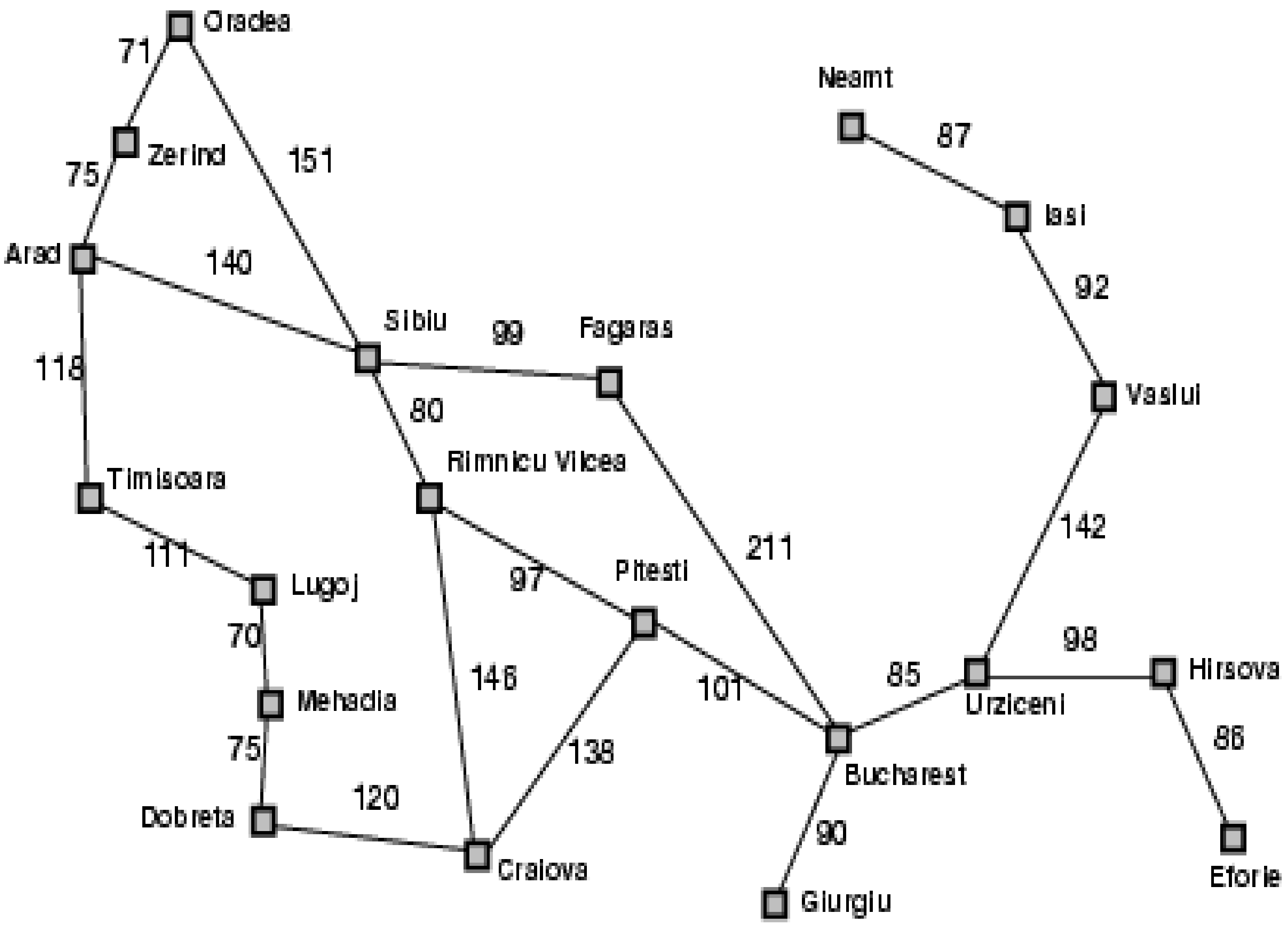
- The general approach we will consider is called **best-first search.**
- **Best-first search is** an instance of the general TREE-SEARCH or GRAPH-SEARCH algorithm in which a node is selected for expansion based on an **evaluation function, f (n) .**

Best-first search

- Idea: use an evaluation function *f(n)* for each node
  - f(n) provides an estimate for the total cost.
  - → Expand the node n with smallest f(n).
  - → $f(n) = h(n)$.
- Implementation:
  Order the nodes in fringe increasing order of cost.

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

Romania with straight-line dist.

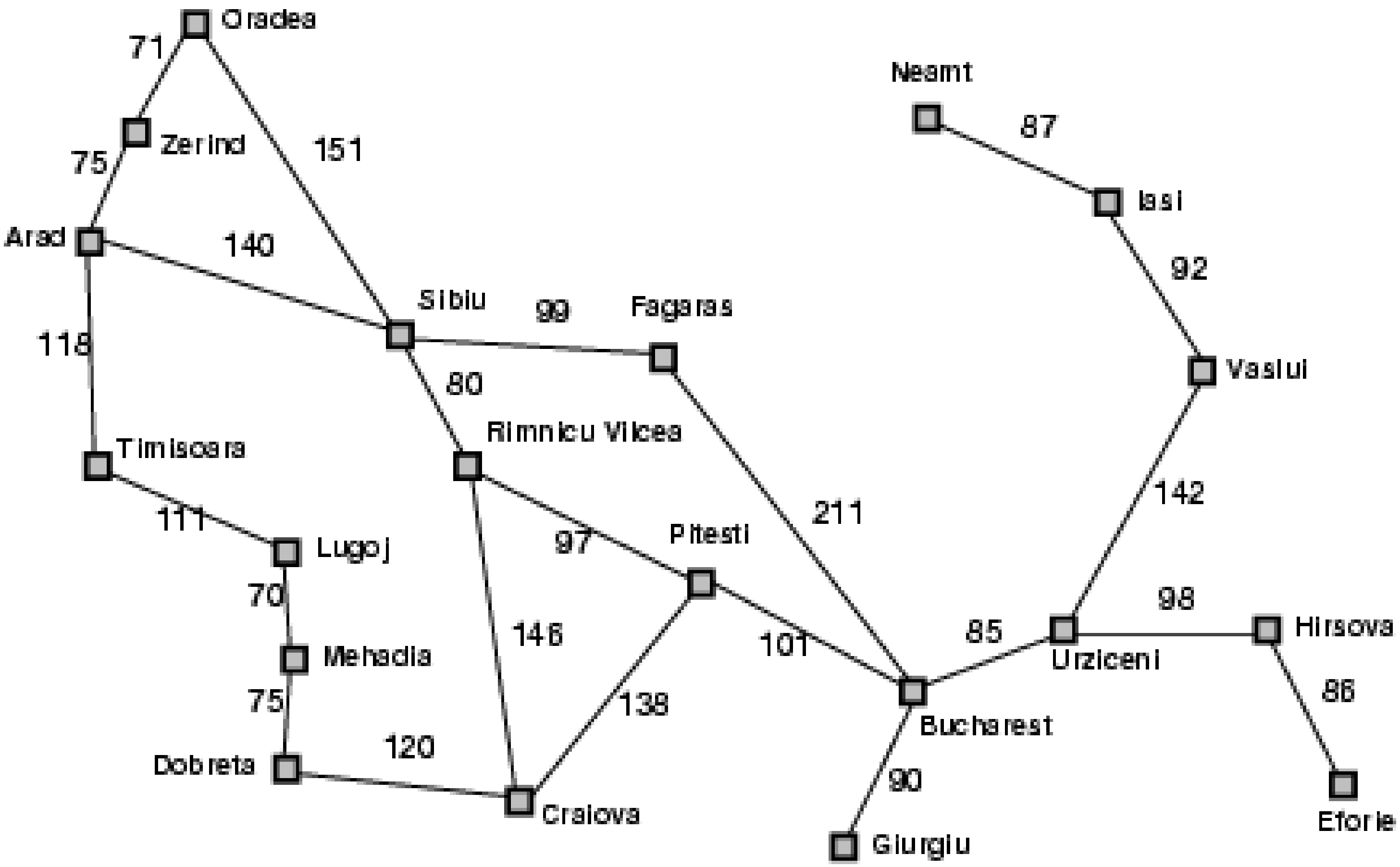


Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy best-first search

- f(n) = estimate of cost from *n* to *goal*
- e.g., *f(n)* = straight-line distance from *n* to Bucharest
- Greedy best-first search expands the node that appears to be closest to goal.

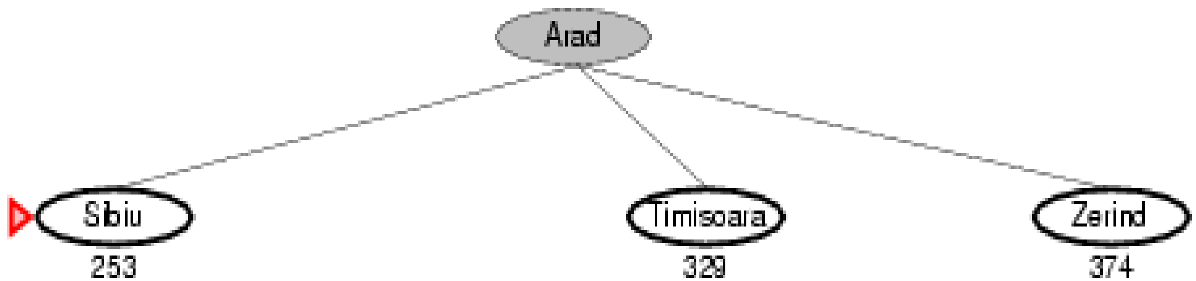
Greedy best-first search example

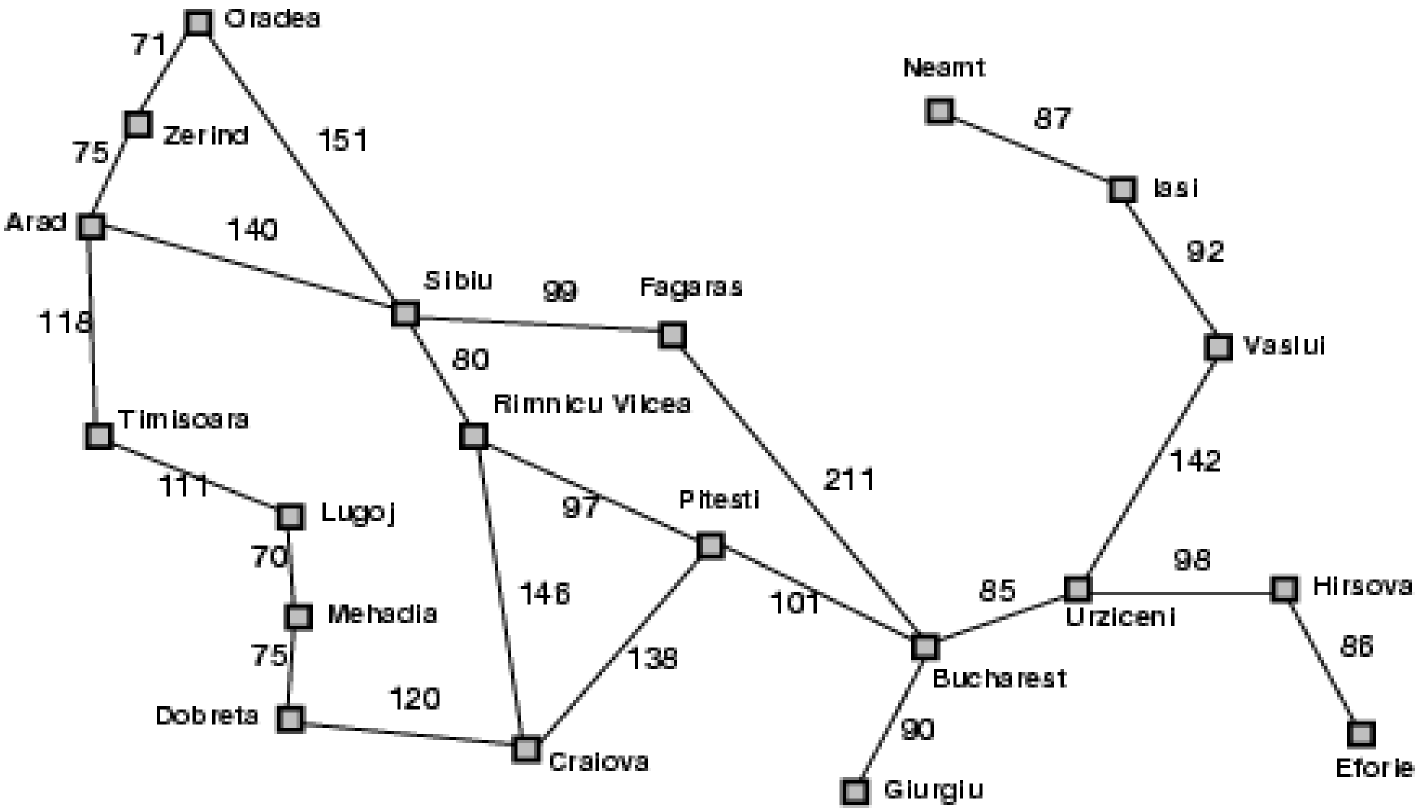


Straight–line distance to Bucharest

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

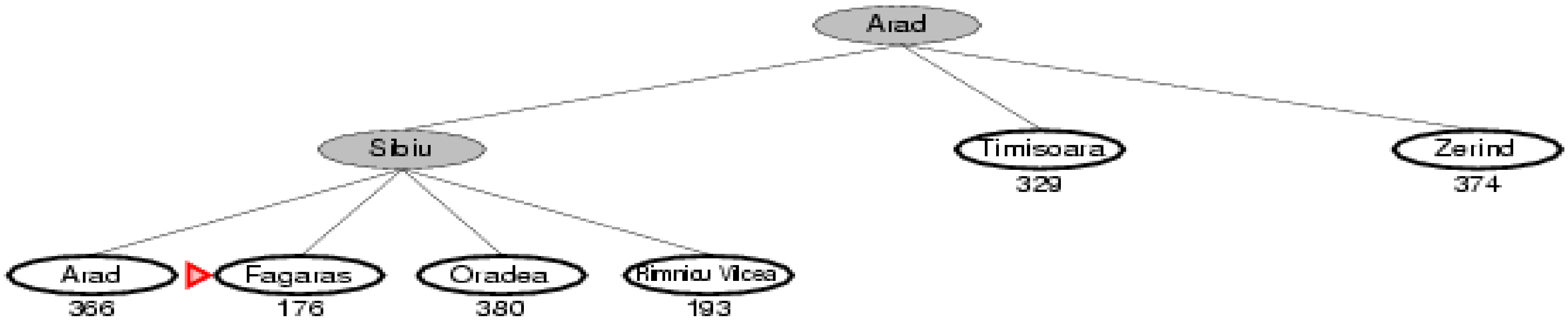


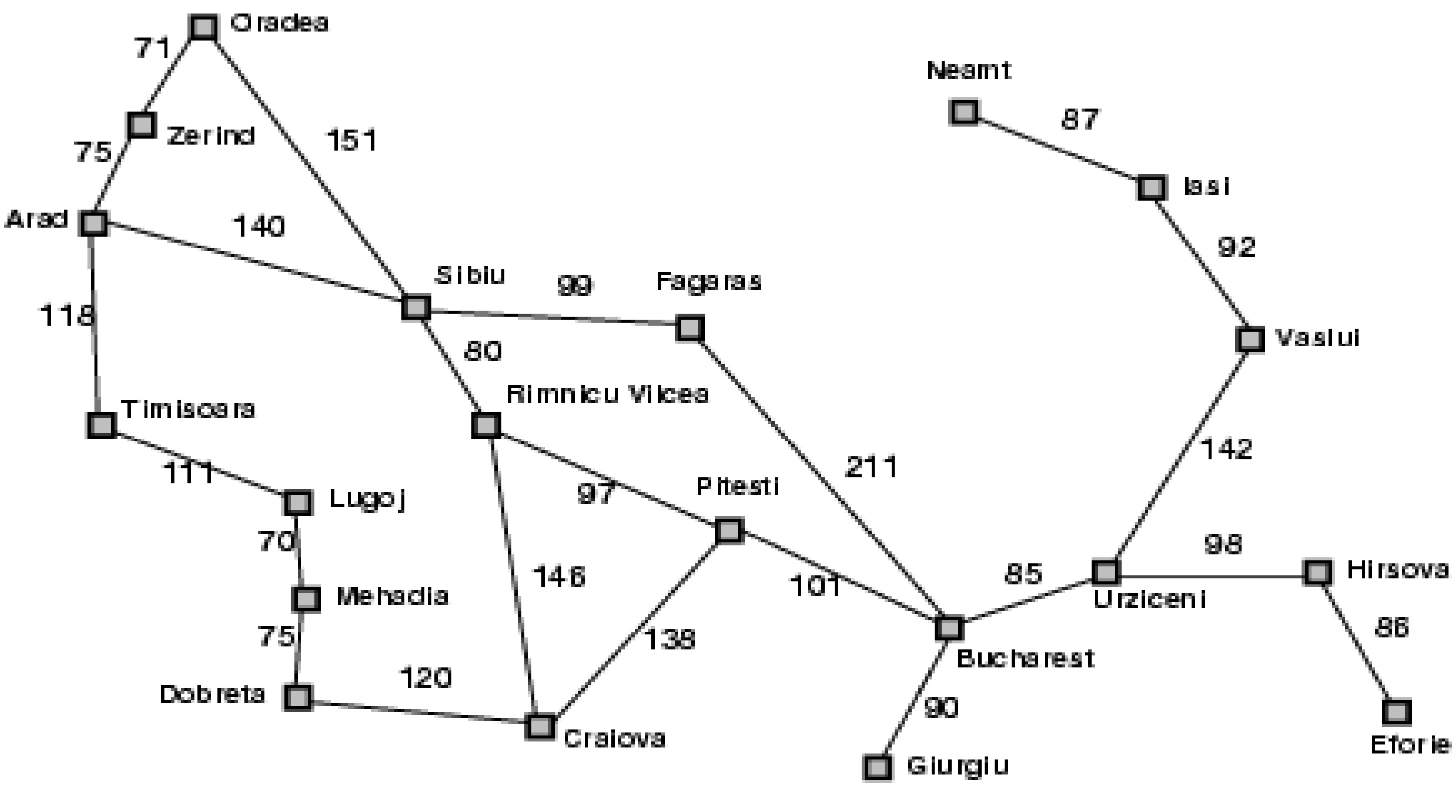


Straight–line distance to Bucharest

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

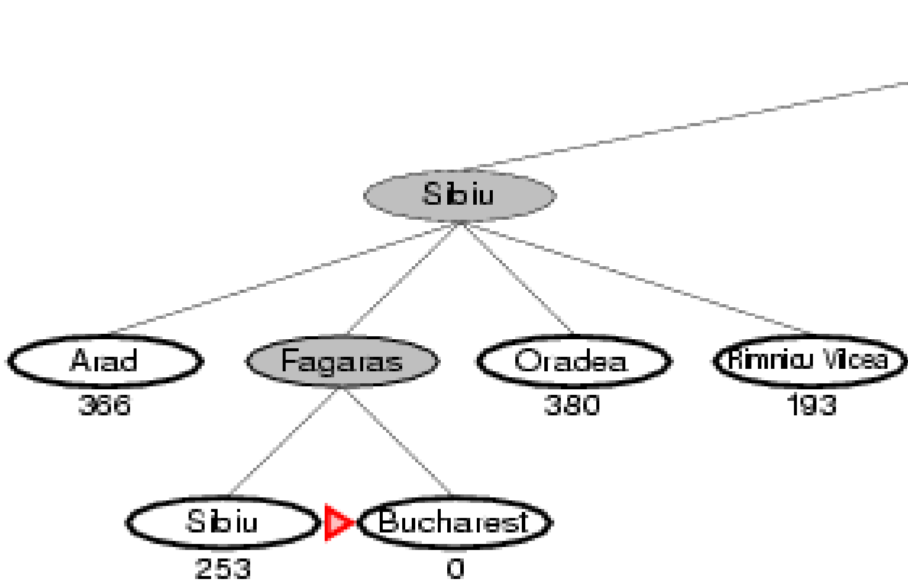


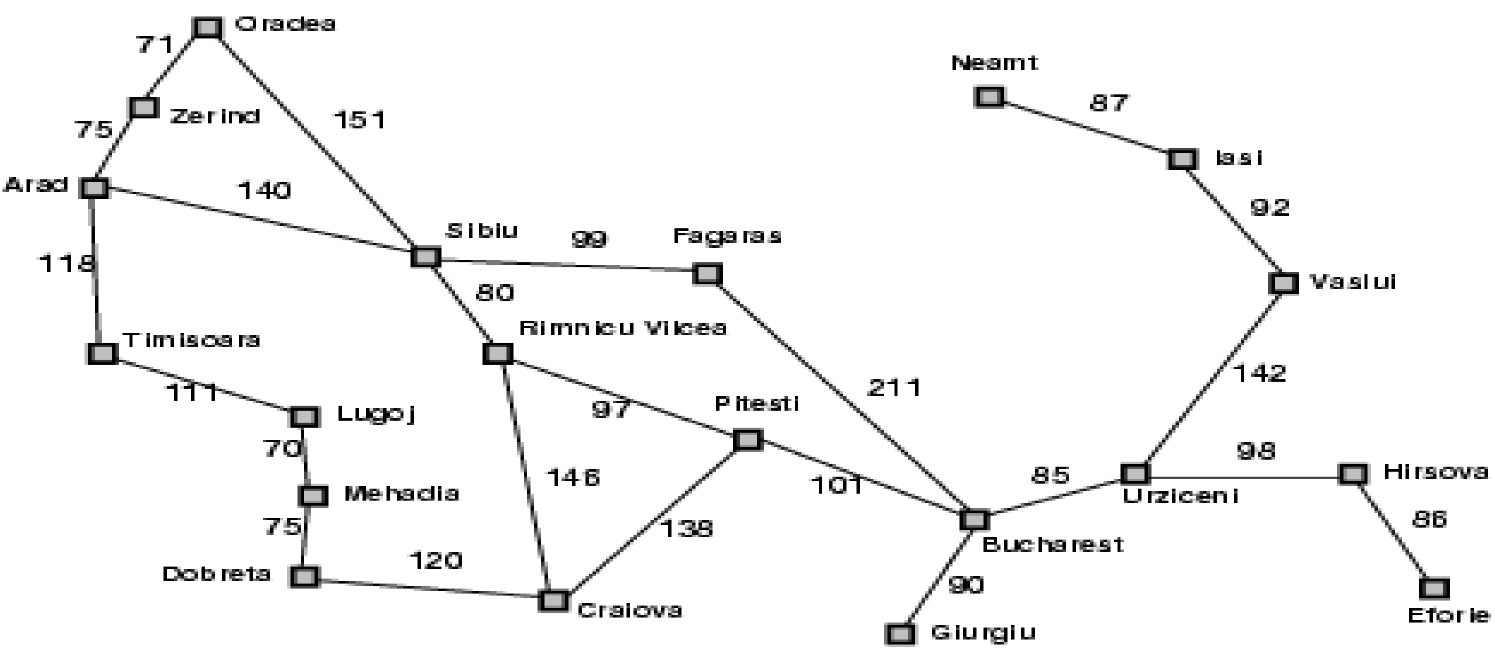


Straight–line distance to Bucharest

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





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

Properties of greedy best-first search

Think of an example

- Complete? No – can get stuck in loops.

Iasi to Fagaras

- Neamt- dead end
- Soln: Iasi -Vaslui-Urziceni- Bucharest-Fagaras.
- Time? $O(b^m)$, but a good heuristic can give dramatic improvement
- Space? $O(b^m)$ - keeps all nodes in memory
- Optimal? No

e.g. Arad→Sibiu→Rimnicu Virea→Pitesti→Bucharest is shorter!

A* search: Minimizing the total estimated solution cost

- Idea: avoid expanding paths that are already expensive
- Evaluation function $f(n) = g(n) + h(n)$
- $g(n)$ = cost to reach n
- *$h(n)$ = estimated cost of the cheapest path from node n to a goal node.*
- $f(n)$ = estimated total cost of path through n to goal
- Best First search has $f(n)=h(n)$
- Uniform Cost search has $f(n)=g(n)$
- if *n is a goal node, then $h(n) = 0$.*

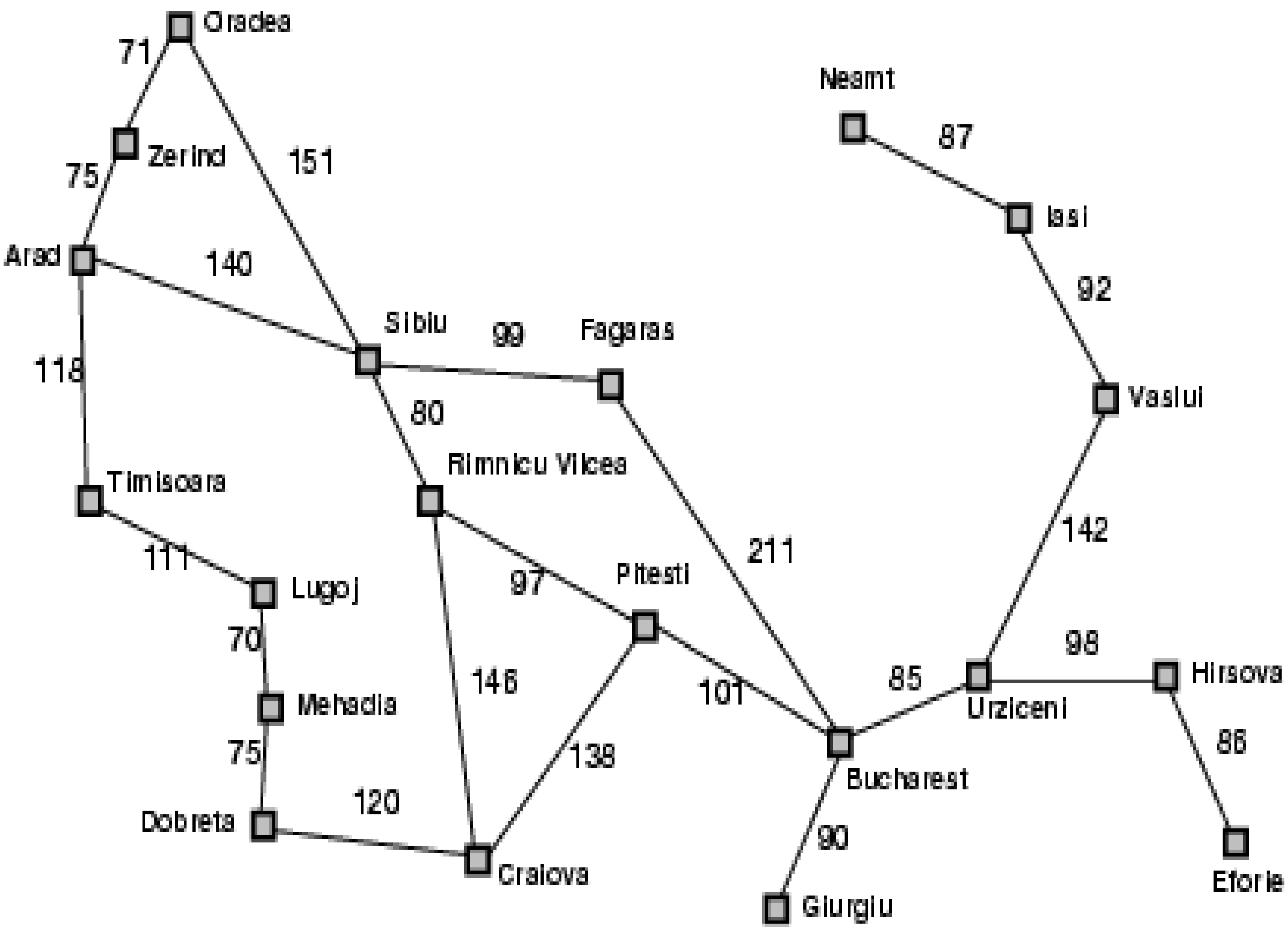
Conditions for optimality: Admissibility and consistency

- The first condition we require for optimality is that h(n) be an **admissible heuristic.**
- A second, slightly stronger condition called **consistency (or sometimes monotonicity)** is required only for applications of A∗ to graph search

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: $h_{SLD}(n)$ (never overestimates the actual road distance)
- Theorem: If $h(n)$ is admissible, A^* using `TREE-SEARCH` is optimal

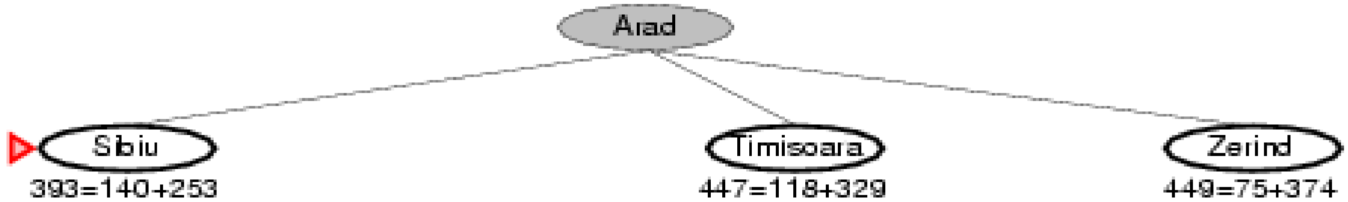
A^* search example

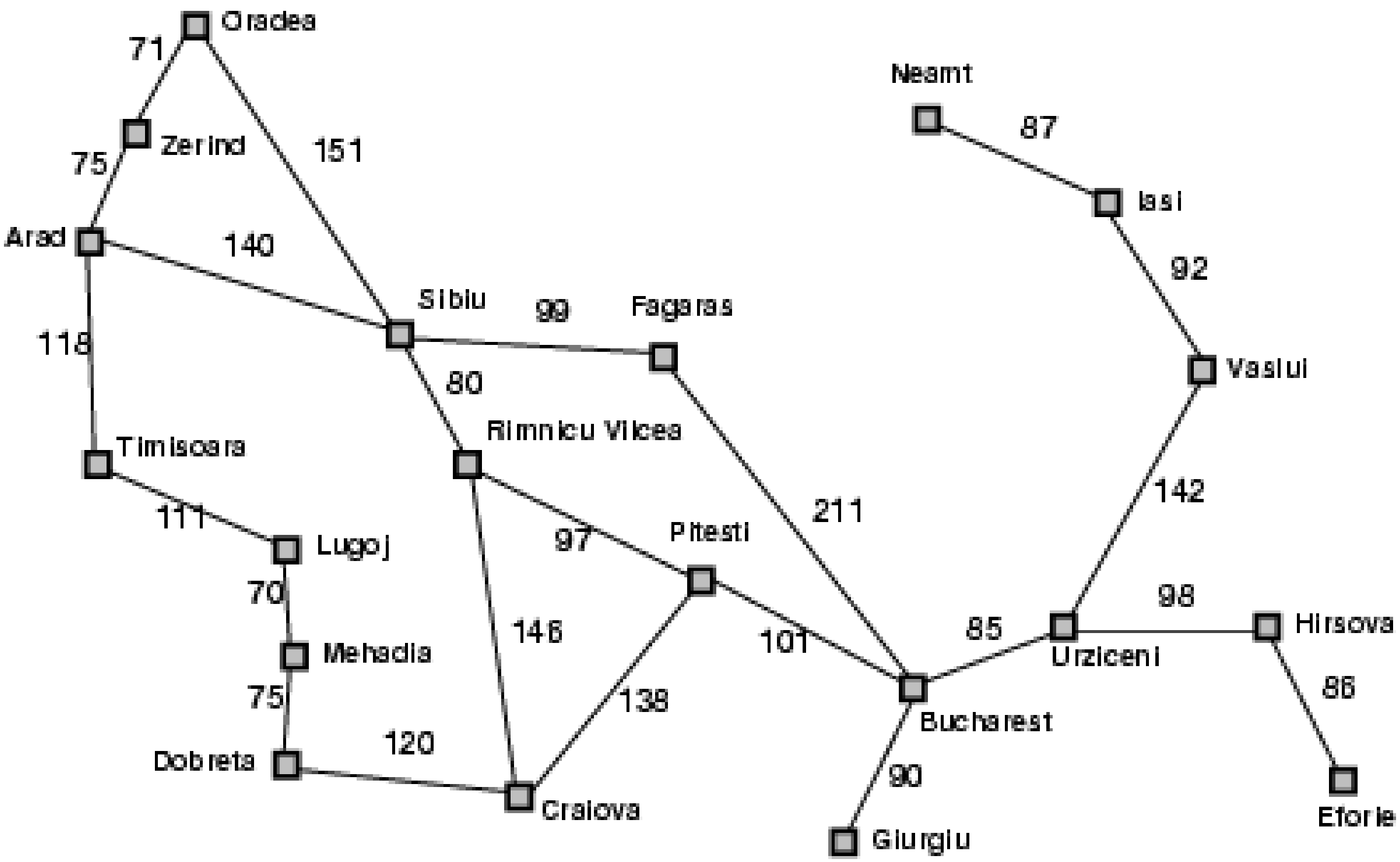


Straight–line distance
to Bucharest

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

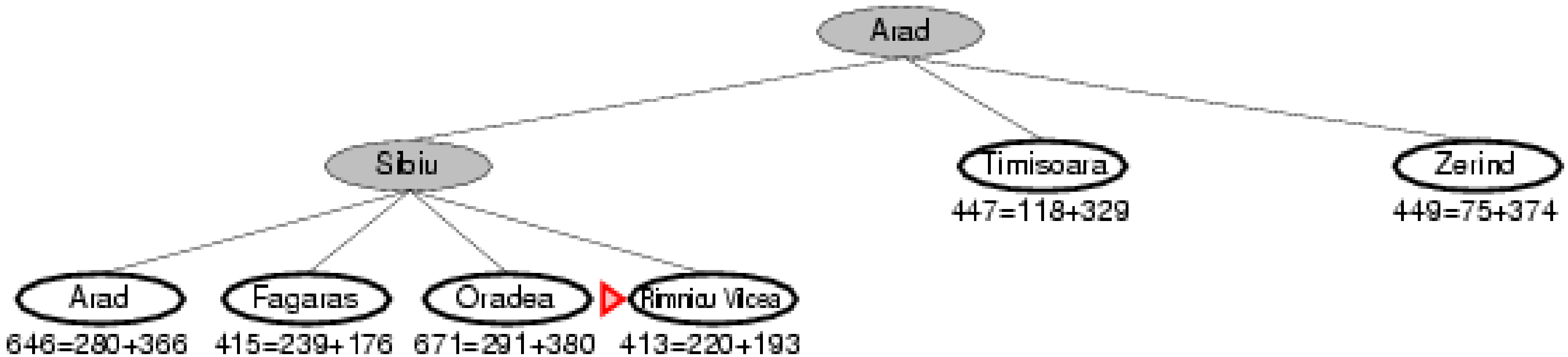


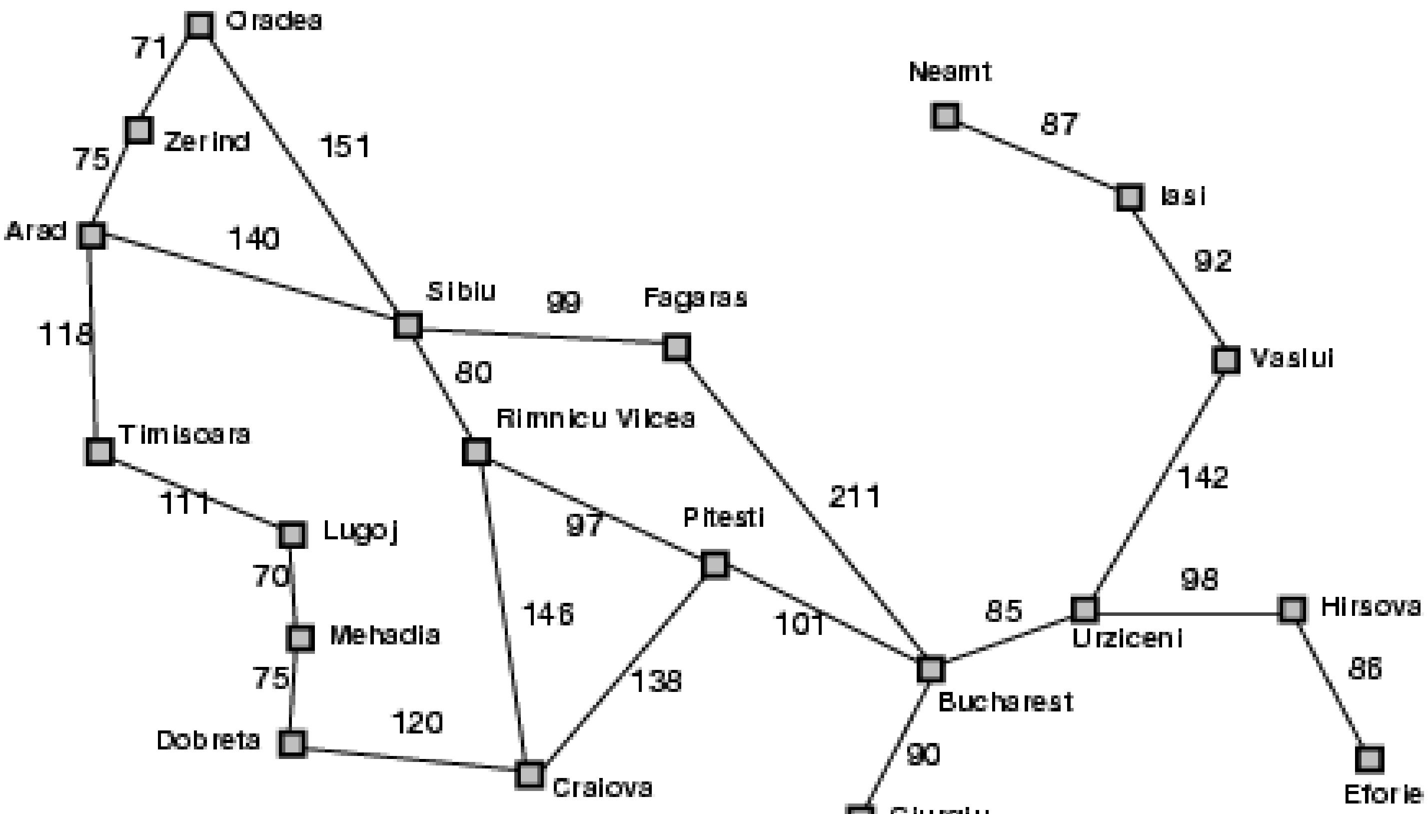


Straight–line distance to Bucharest

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

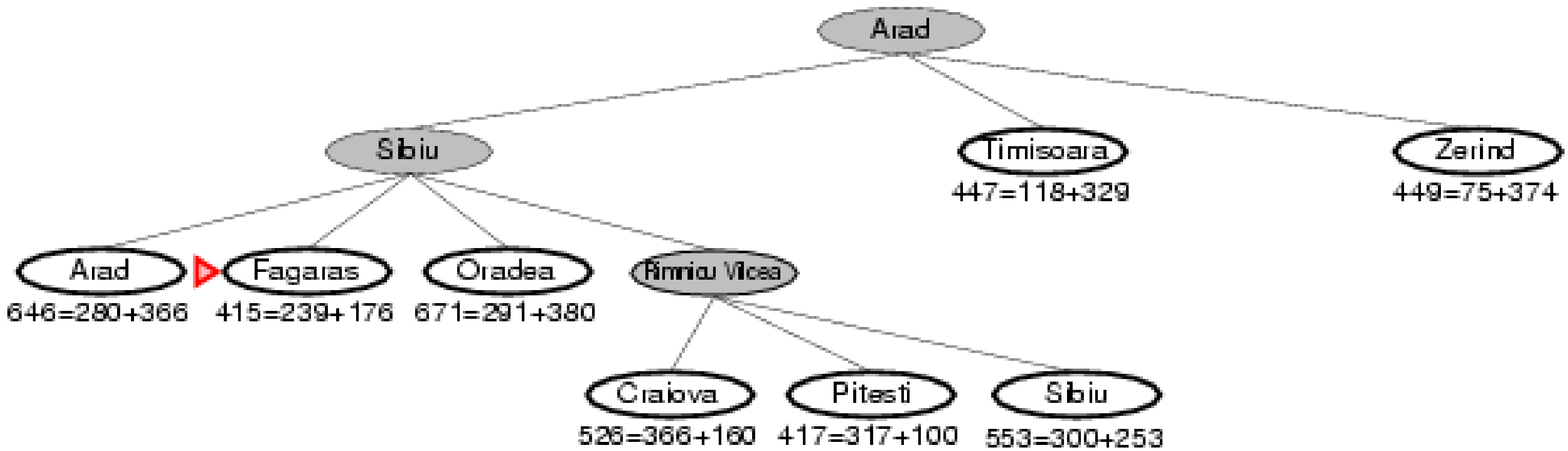
A* search example

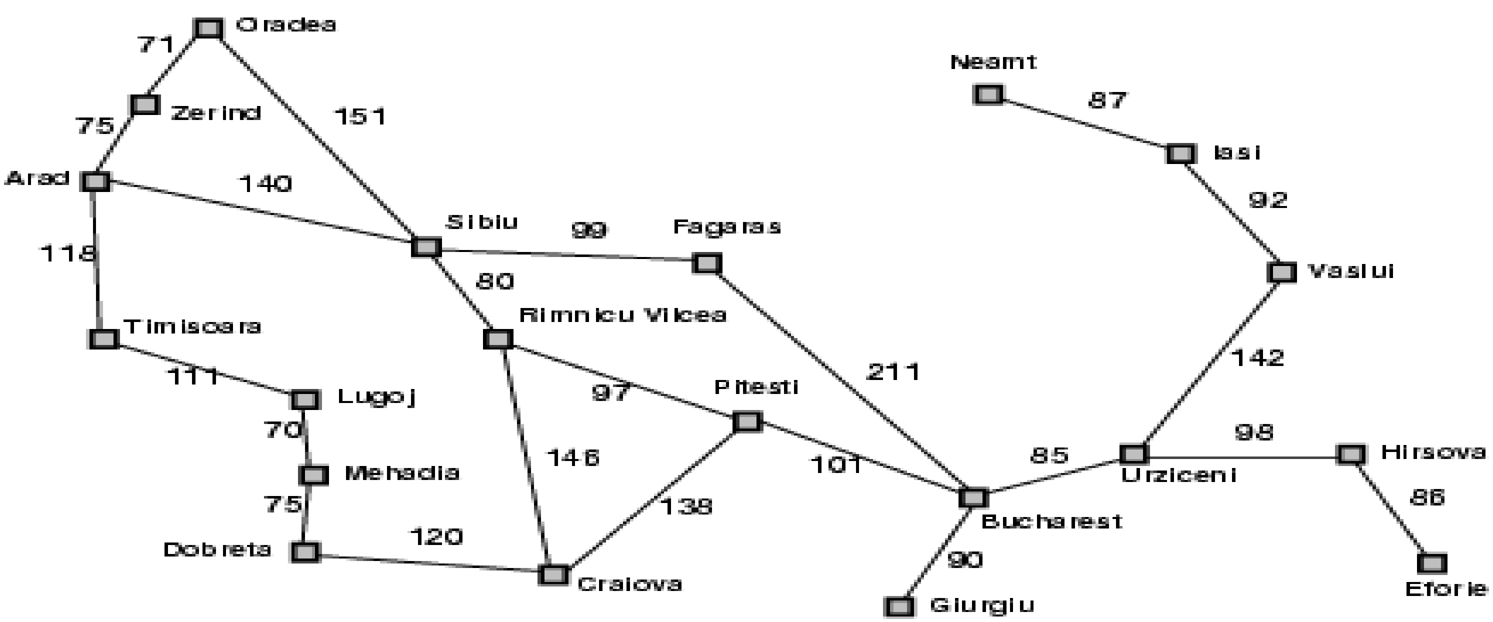




Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199

A* search example

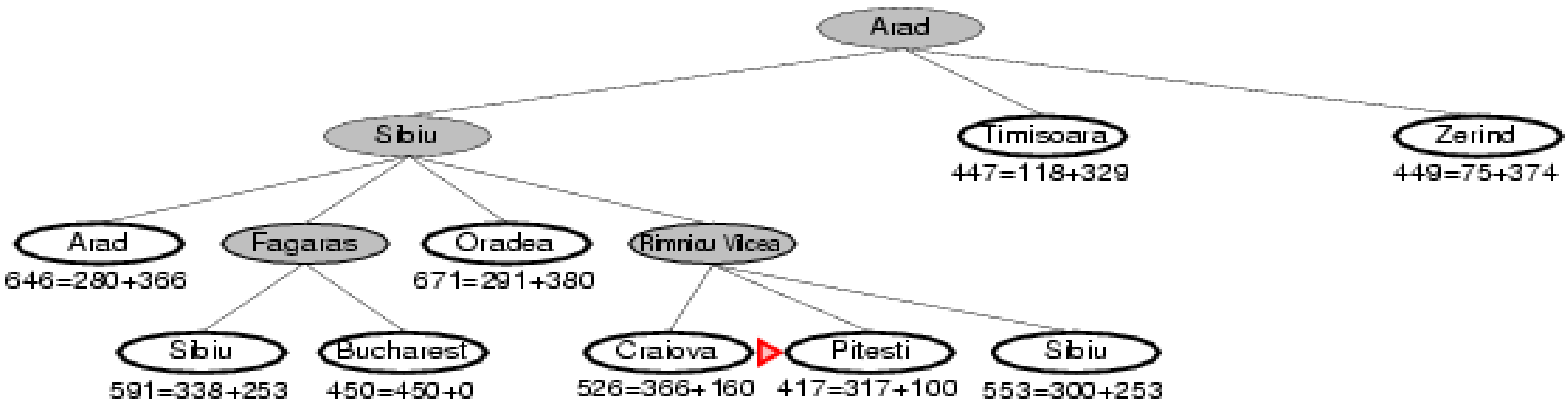


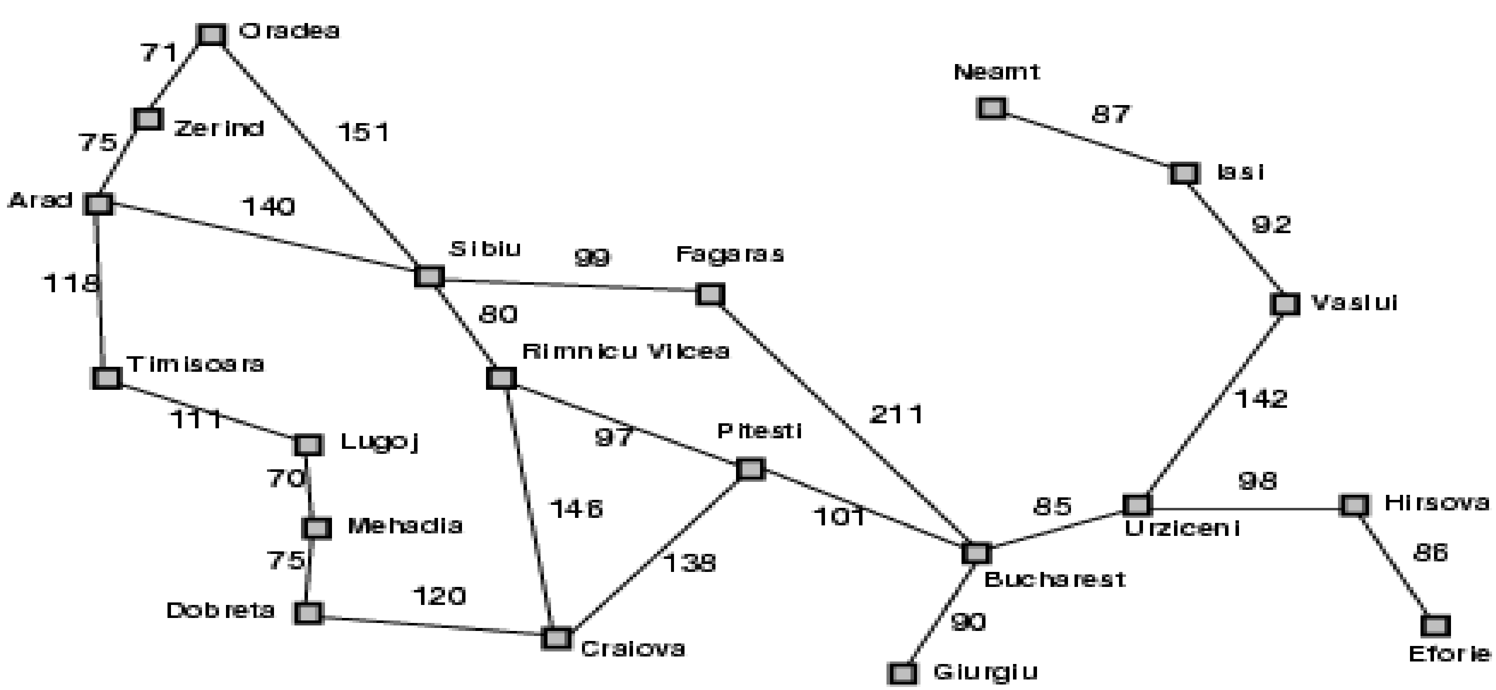


Straight–line distance to Bucharest

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

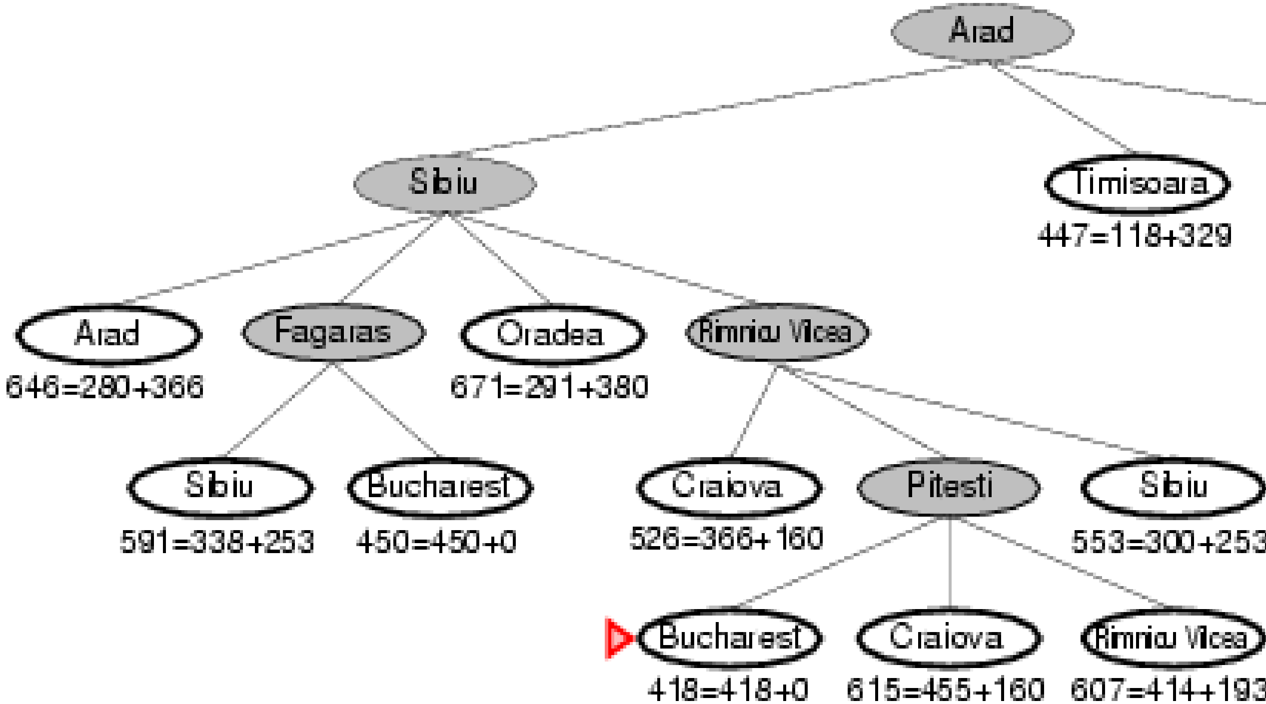


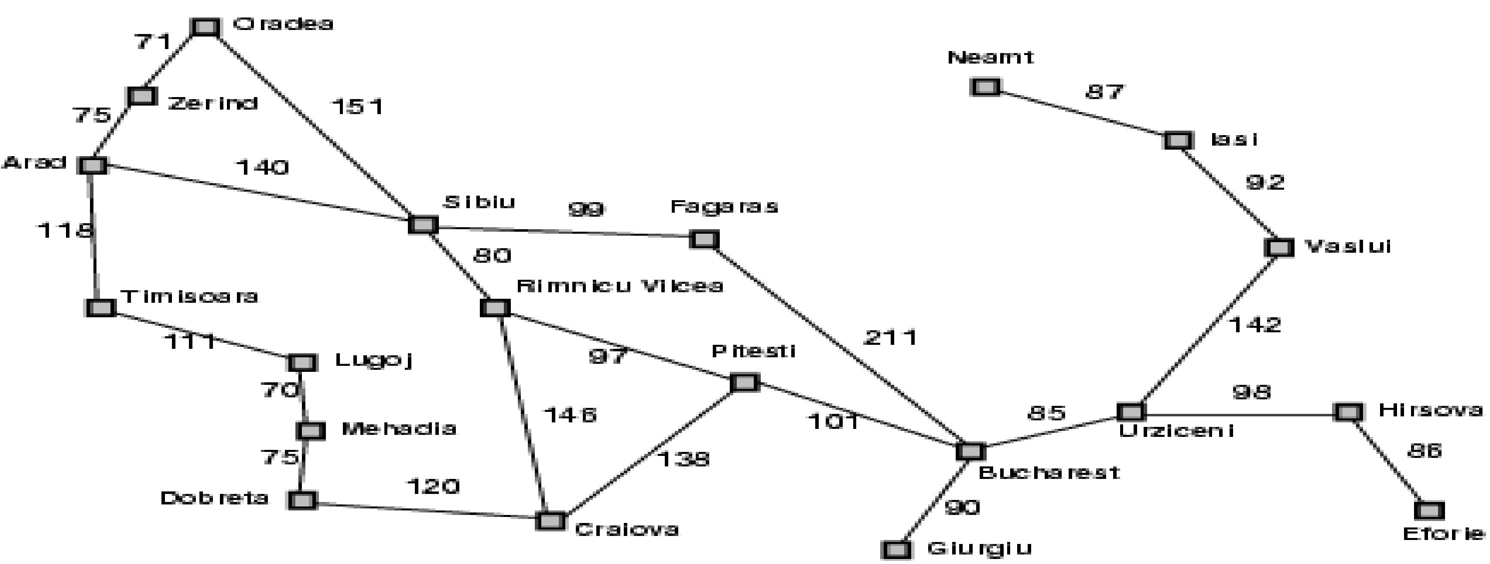


Straight–line distance
to Bucharest

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





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

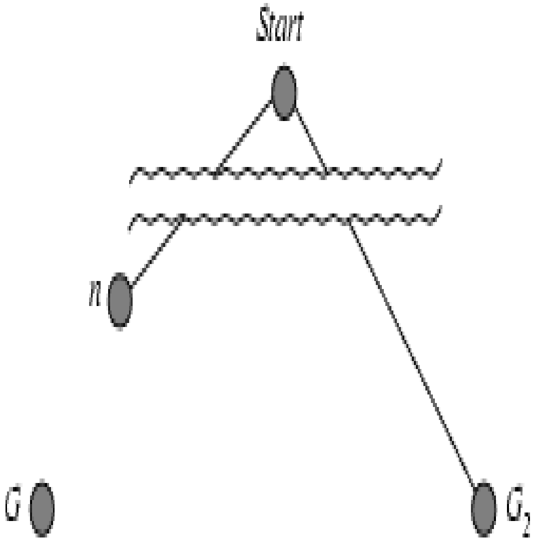
Properties of A*

- <u>Complete?</u> A* *using TREE-SEARCH is optimal if h(n) is admissible. (bucharest values>Pitesti values)*
- <u>Time/Space?</u> Exponential

  except if: $|h(n) - h^{*}(n)| \leq O(\log h^{*}(n))$
- <u>Optimal?</u> A* *using TREE-SEARCH is optimal if h(n) is admissible*
- *<u>Optimally Efficient</u>*: Yes (no algorithm with the same heuristic is guaranteed to expand fewer nodes)

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*.
- let the cost of the optimal solution be C*.



We want to prove:
f(n) < f(G2)
(then A* will prefer n over G2)

f (G2) = g(G2) + h(G2) = g(G2) > C .*

because *G2 is suboptimal and because h(G2)=0*

(true for any goal node),

f (n) = g(n) + h(n) <= C .*

- *f(n) <=C* < f (G2)* s,o *G2 will not be expanded and A* must*
- return an optimal solution.

Graph Search

- If we use the GRAPH-SEARCH algorithm instead of TREE-SEARCH, then this proof breaks down.
- Suboptimal solutions can be returned because GRAPH-SEARCH can discard the optimal path to a repeated state if it is not the first one generated.

two ways to fix this problem

- The first solution is to extend GRAPH-SEARCH so that it discards the more expensive of any two paths found to the same node.
- The extra bookkeeping is messy, but it does guarantee optimality.

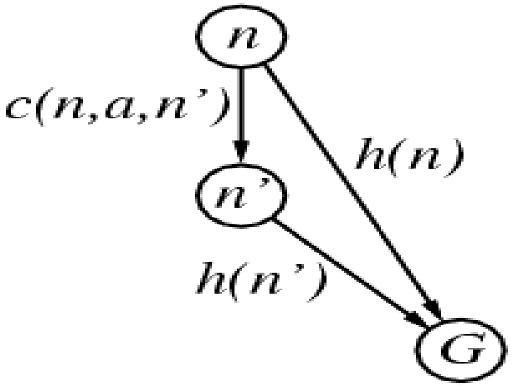
two ways to fix this problem

- The second solution is to ensure that the optimal path to any repeated state is always the first one followed-as is the case with uniform-cost search.
- This property holds if we impose an extra requirement on *h(n),namely the requirement of* ***consistency (also called*** **monotonicity).**

Consistent heuristics

- A heuristic is consistent if for every node *n*, every successor *n'* of *n* generated by any action *a*, the estimated cost of reaching the goal from n is no greater than the step cost of getting to n plus the estimated cost of reaching the goal from n :

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



each side of a triangle cannot be longer than the sum of the other two sides.

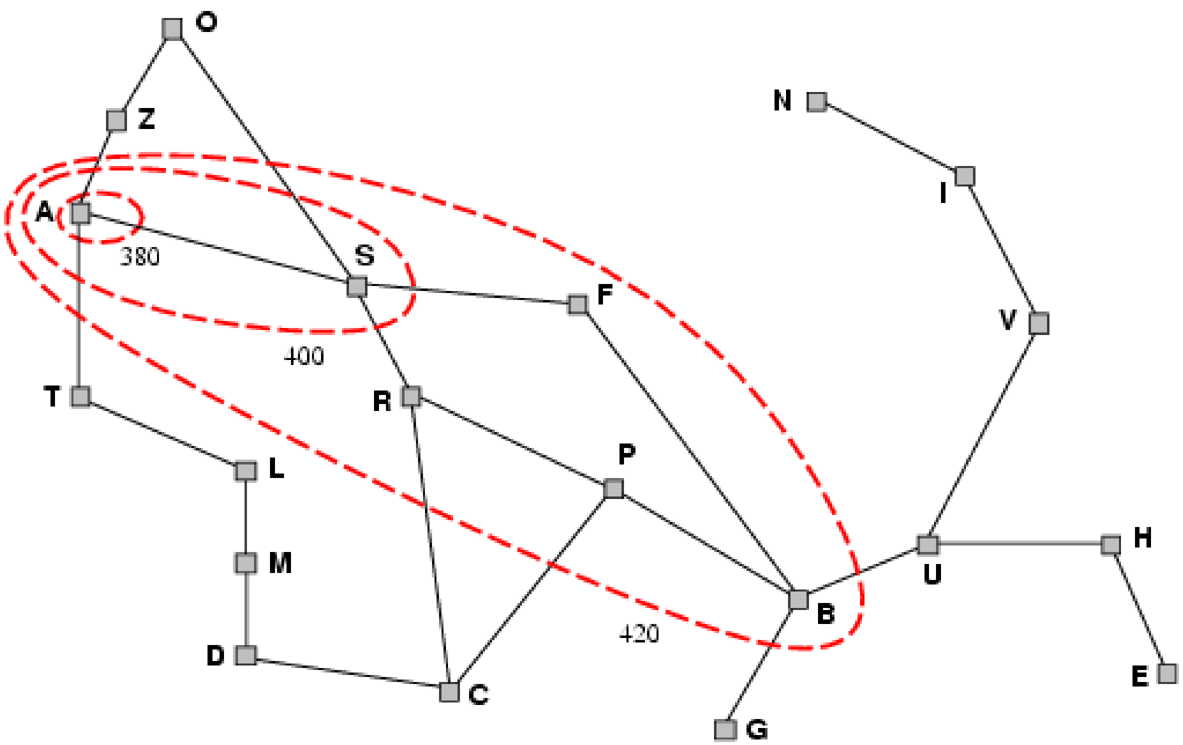
For an **admissible heuristic**, the inequality makes perfect sense: if there were a route from n to Gn via n that was cheaper than h(n), that would violate the property that h(n) is a lower bound on the cost to reach Gn.

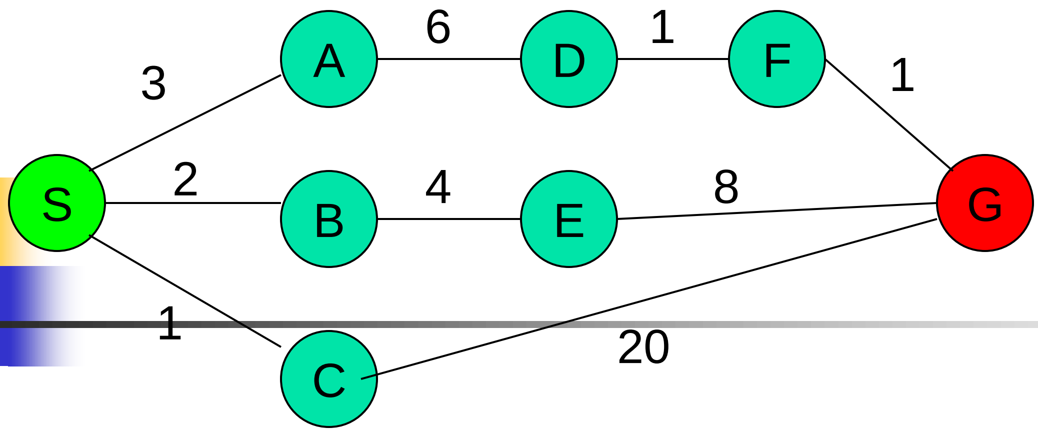
It's the triangle inequality !

A* *using* GRAPH-SEARCH *is optimal if h(n) is consistent.*

Optimality of A*

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





straight-line distances

h(S-G)=10
h(A-G)=7
h(D-G)=1
h(F-G)=1
h(B-G)=10
h(E-G)=8
h(C-G)=20

try yourself

The graph above shows the step-costs for different paths going from the start (S) to the goal (G). On the right you find the straight-line distances.

1. Draw the search tree for this problem. *Avoid repeated states*.

2. Give the order in which the tree is searched (e.g. S-C-B...-G) for A* search.
Use the straight-line dist. as a heuristic function, i.e. h=SLD,
and indicate for each node visited what the value for the evaluation function, f, is.

Memory Bounded Heuristic Search: Recursive BFS

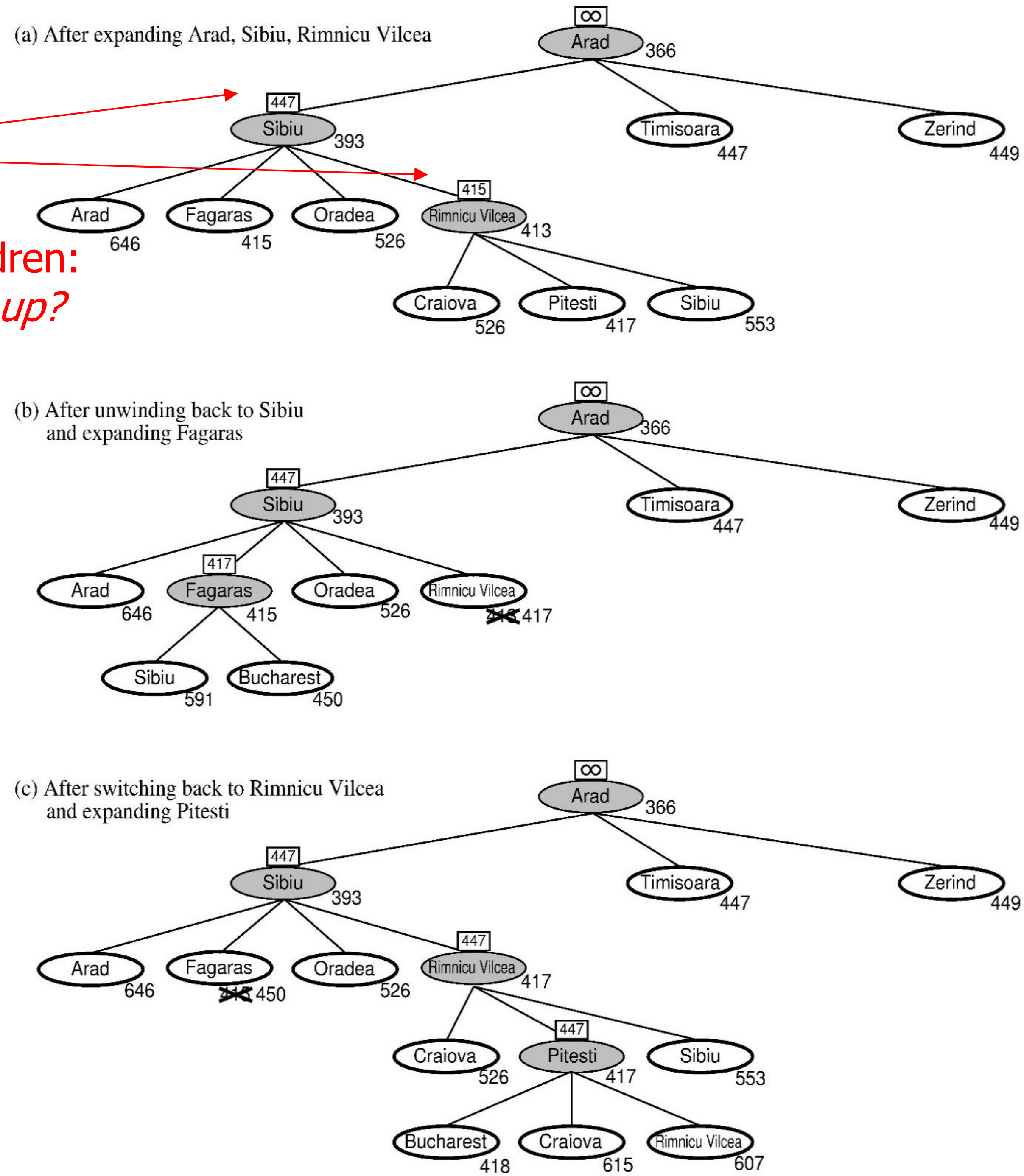
- How can we solve the memory problem for A* search?
- Idea: Try something like depth first search, but let’s not forget everything about the branches we have partially explored.
- *We remember the best f-value we have found so far in the branch we are deleting*.

Memory Bounded Heuristic Search

- 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.*

RBFS:

best alternative
over fringe nodes,
which are not children:
do I want to back up?

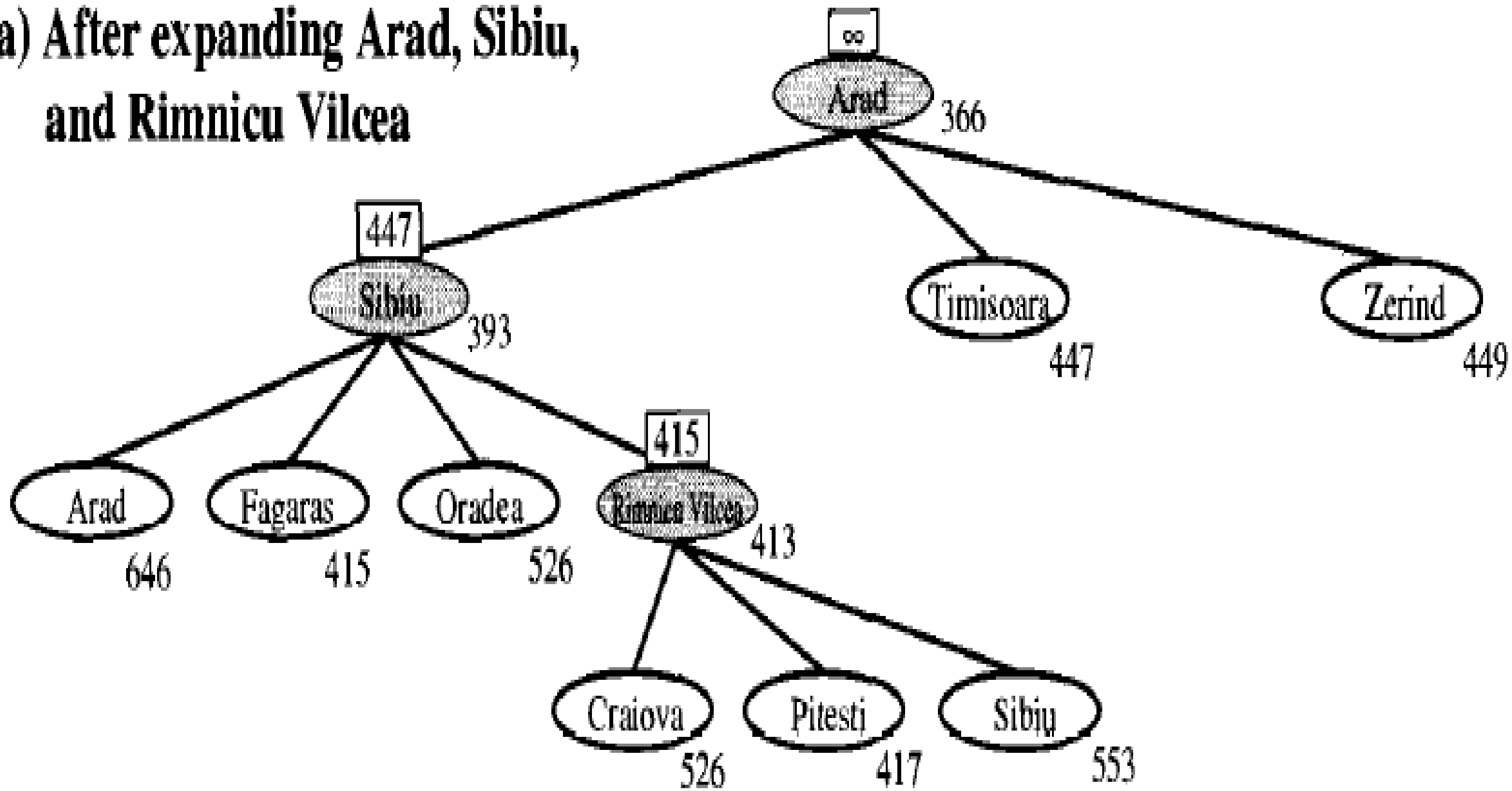


RBFS changes its mind
very often in practice.

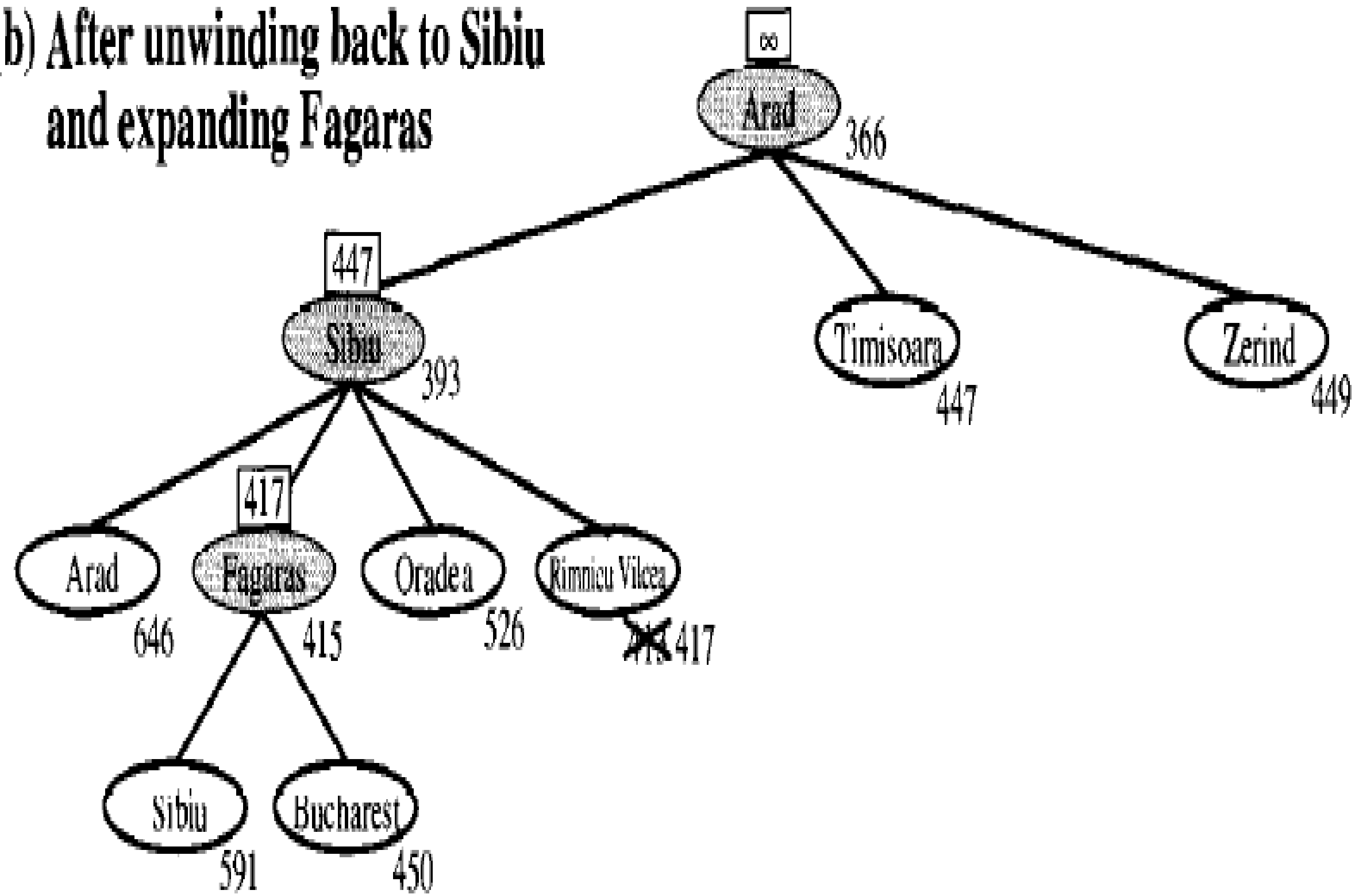
This is because the
f=g+h become more
accurate (less optimistic)
as we approach the goal.
Hence, higher level nodes
have smaller f-values and
will be explored first.

Problem: We should keep
in memory whatever we can.

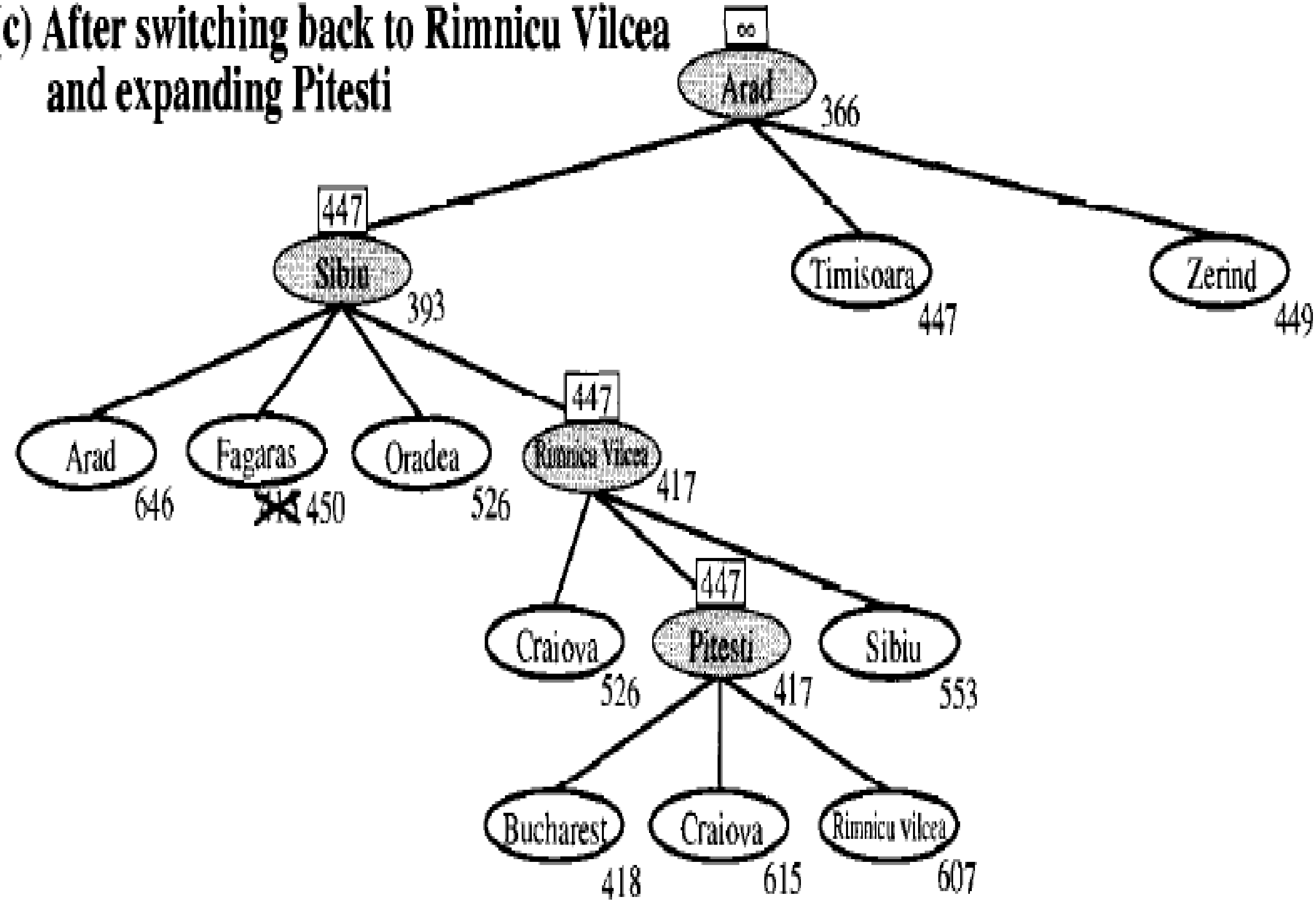
(a) After expanding Arad, Sibiu, and Rimnicu Vilcea



(b) After unwinding back to Sibiu and expanding Fagaras



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



Simple Memory Bounded A*

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

A Solution is not reachable
if a single path from root to goal
does not fit into memory

Admissible heuristics

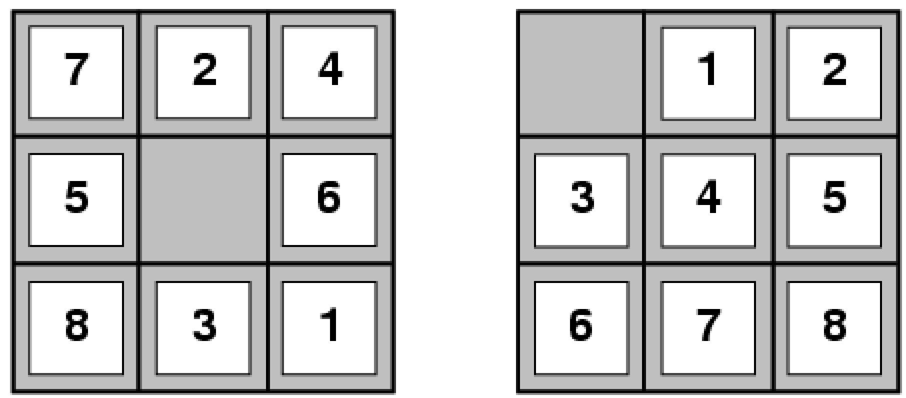
- h1 = the number of misplaced tiles. start state would have h1 = 8. h1 is an admissible heuristic because it is clear that any tile that is out of place must be moved at least once.
- h2 = the sum of the distances of the tiles from their goal positions. Because tiles cannot move along diagonals, the distance we will count is the sum of the horizontal and vertical distances.

Admissible heuristics

E.g., for the 8-puzzle:

- $h_1(n)$ = number of misplaced tiles
- $h_2(n)$ = total Manhattan distance

(i.e., no. of squares from desired location of each tile)



Start State   Goal State

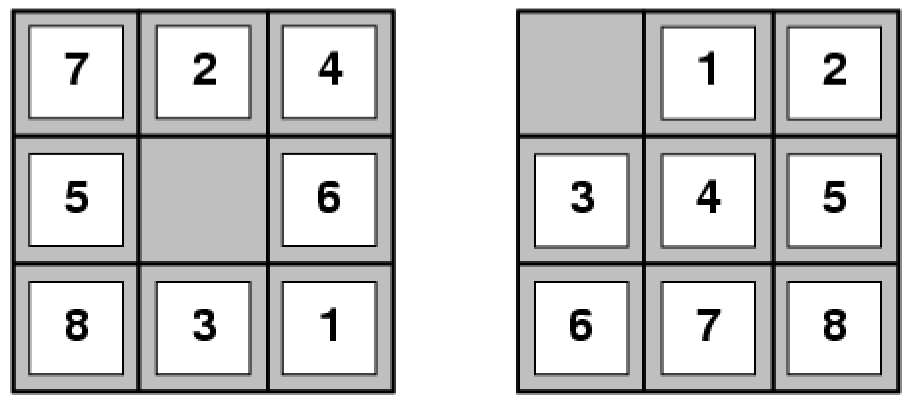
- $h_1(S)$ = ?
- $h_2(S)$ = ?

Admissible heuristics

E.g., for the 8-puzzle:

- $h_1(n)$ = number of misplaced tiles
- $h_2(n)$ = total Manhattan distance

(i.e., no. of squares from desired location of each tile)



Start State Goal State

- $h_1(S) = ?$ 8
- $h_2(S) = ?$ 3+1+2+2+2+3+3+2 = 18

Dominance

- If $h_2(n) \geq h_1(n)$ for all n (both admissible)
- then h_2 dominates h_1
- h_2 is better for search: it is guaranteed to expand less or equal nr of nodes.

- Typical search costs (average number of nodes expanded):

- *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

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

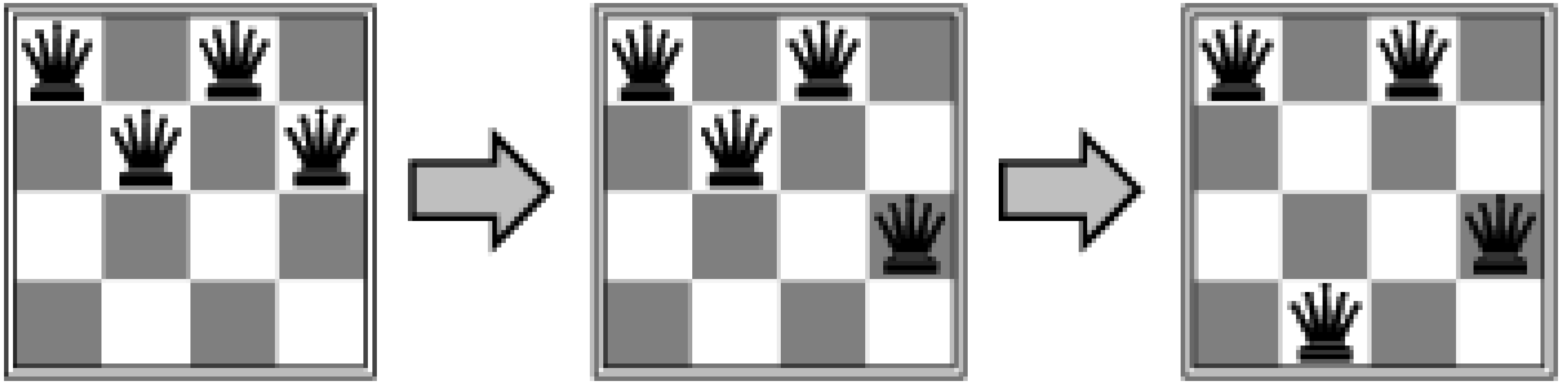
Local search algorithms

- In many optimization problems, the path to the goal is irrelevant; the goal state itself is the solution

- State space = set of "complete" configurations
- Find configuration satisfying constraints, e.g., n-queens
- In such cases, we can use local search algorithms
- keep a single "current" state, try to improve it.
- Very memory efficient (only remember current state)

Example: *n*-queens

- Put *n* queens on an *n* × *n* board with no two queens on the same row, column, or diagonal



Note that a state cannot be an incomplete configuration with m<n queens