2802ICT Intelligent Systems

Informed search

Outline

- Heuristics
- Best-first search
  - Greedy search
  - A* search
- Admissible heuristics

Reading: textbook AIMA Chapter 3, pages 92-109

Heuristics

- A heuristic is a rule or principle used to guide a search
  - It provides a way of giving additional knowledge of the problem to the search algorithm
  - Must provide a reasonably reliable estimate of how far a state is from a goal, or the cost of reaching the goal via that state

- A heuristic evaluation function is a way of calculating or estimating such distances/cost

Heuristics and algorithms

- A correct algorithm will find you the best solution given good data and enough time
  - It is precisely specified

- A heuristic gives you a workable solution in a reasonable time
  - It gives a guided or directed solution

Evaluation function

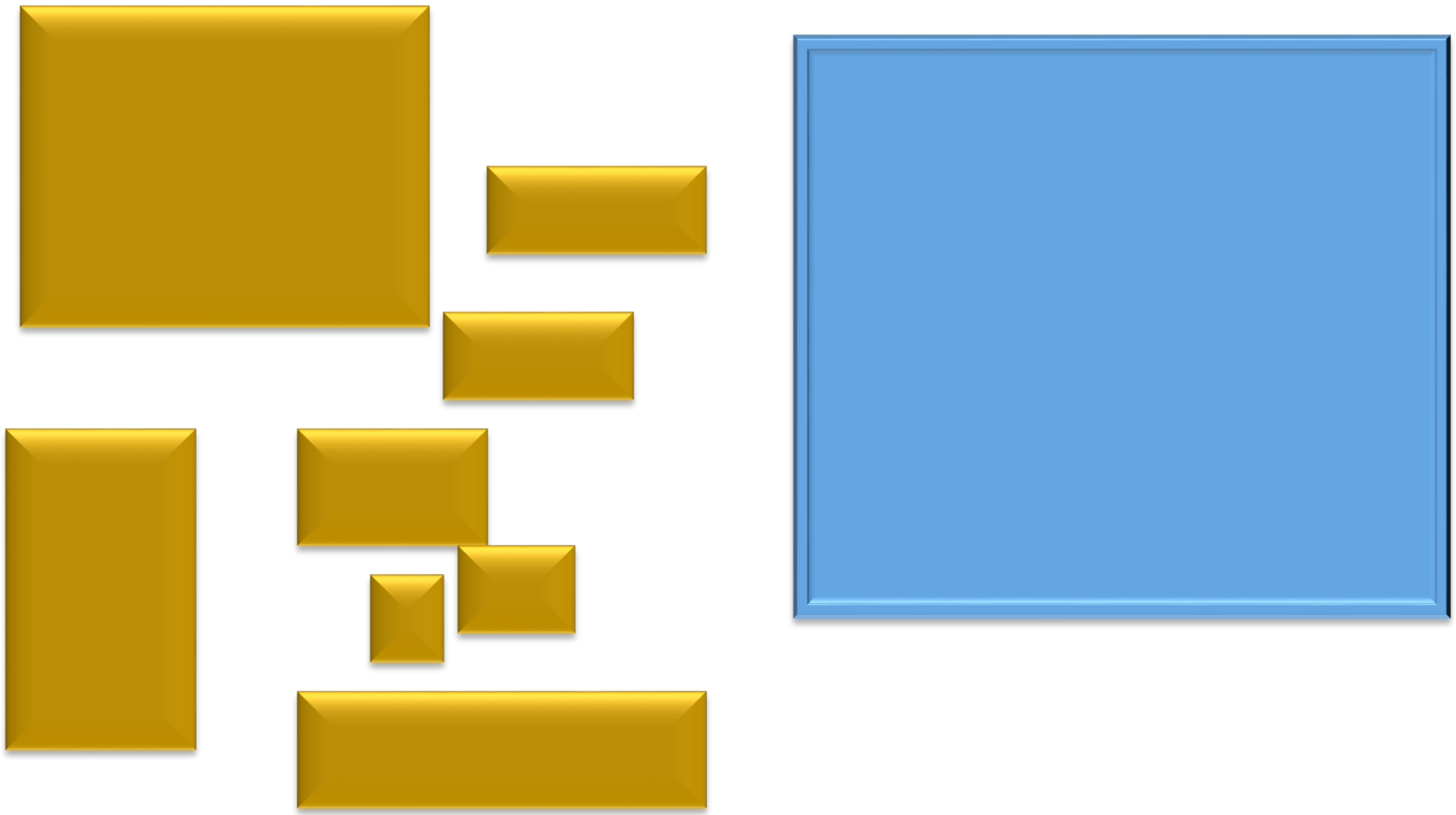
- There are an infinite number of possible heuristics
  - Criteria is that it returns an assessment of the point in the search

- If an evaluation function is accurate, it will lead directly to the goal

- More realistically, this usually ends up as “seemingly-best-search”

- Traditionally, the lowest value after evaluation is chosen as we usually want the lowest cost or nearest

Heuristics?



Problem: Pack blocks as compactly as possible into the space provided

Recall that ...

```
function TREE-SEARCH(problem) returns a solution, or failure
  initialize the frontier using the initial state of problem
  loop do
     if the frontier is empty then return failure
     choose a leaf node and remove it from the frontier
     if the node contains a goal state then return the corresponding solution
     expand the chosen node, adding the resulting nodes to the frontier
```

A strategy is defined by picking the order of node expansion, i.e. Depth-first, breadth-first, etc.

Best-first search

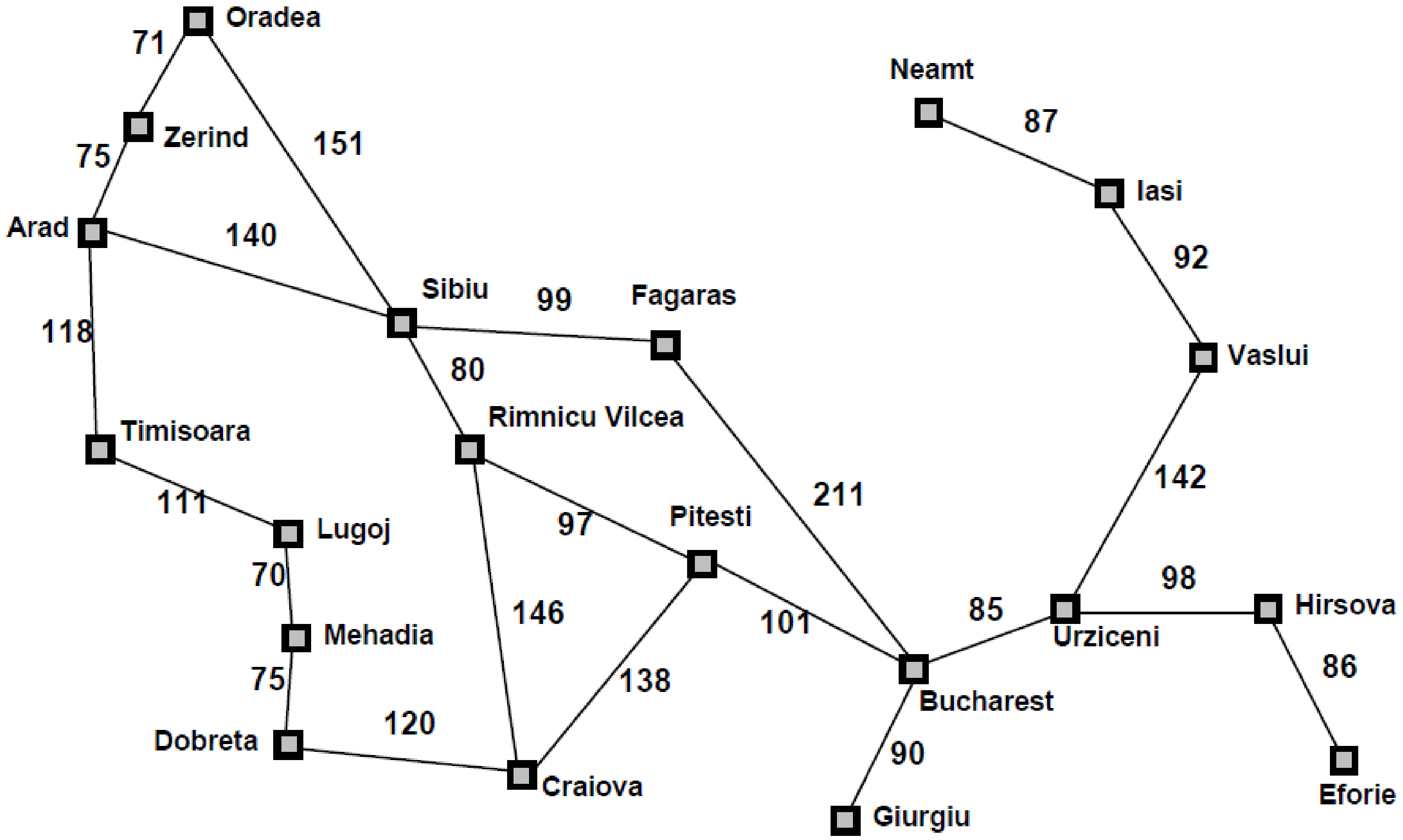
- Idea: use an evaluation function for each node
  - estimate of “desirability”

  $\Rightarrow$ Expand most desirable unexpanded node

- Implementation:
  - Frontier is a queue sorted in decreasing order of desirability

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

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

Greedy best-first search

- Evaluation function $h(n)$ (heuristic)

  = estimate of cost from n to the closest goal

  E.g., $h_{SLD}(n)$ = straight-line distance from n to Bucharest

- Greedy search expands the node that appears to be closest to goal

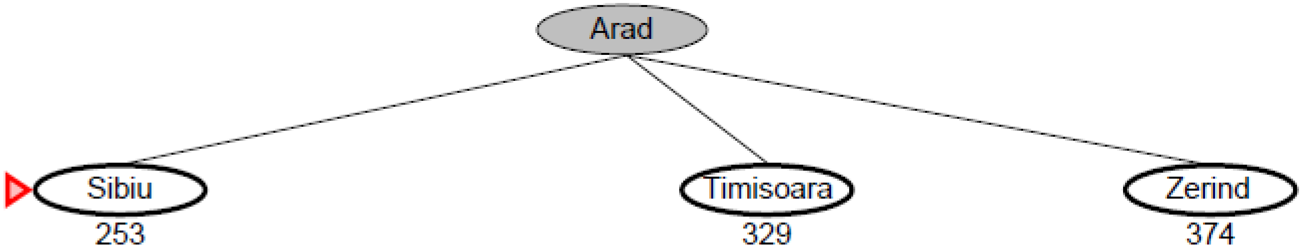
Greedy best-first search example

The initial state:

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

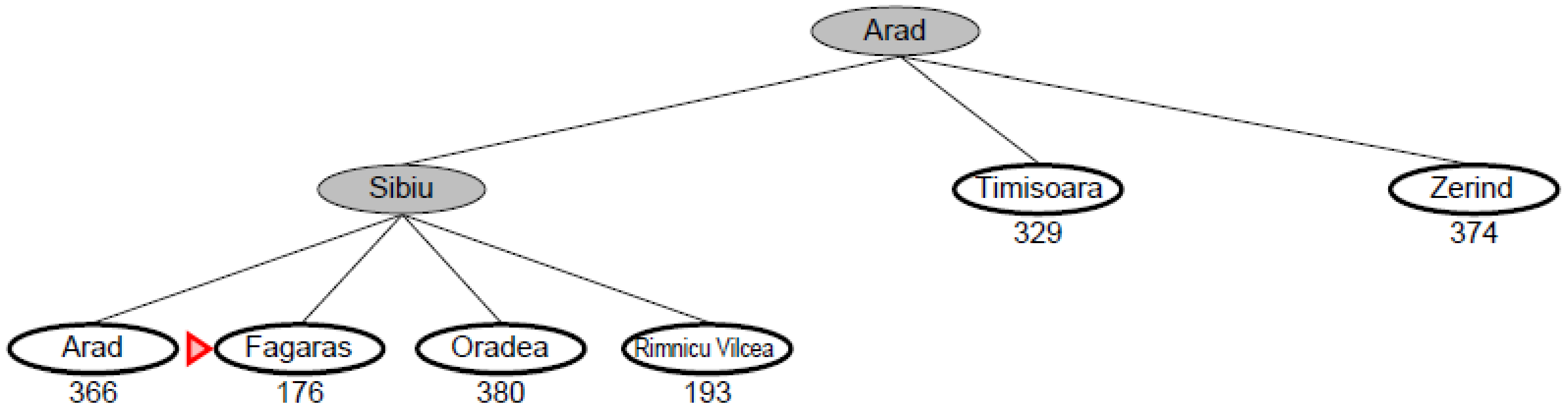
Greedy best-first search example

After expanding Arad:



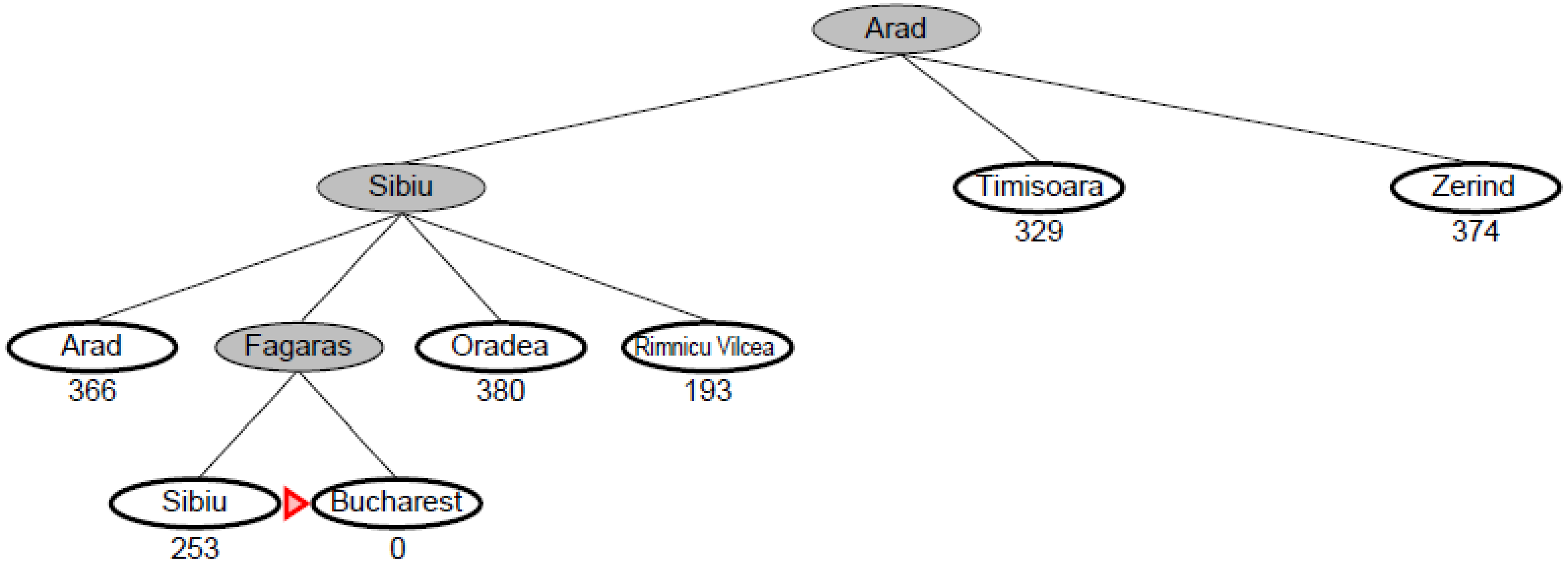
Greedy best-first search example

After expanding Sibiu:



Greedy best-first search example

After expanding Fagaras:



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??** $O(b^m)$, but a good heuristic can give dramatic improvement

- **Space??** $O(b^m)$ - keeps all nodes in memory

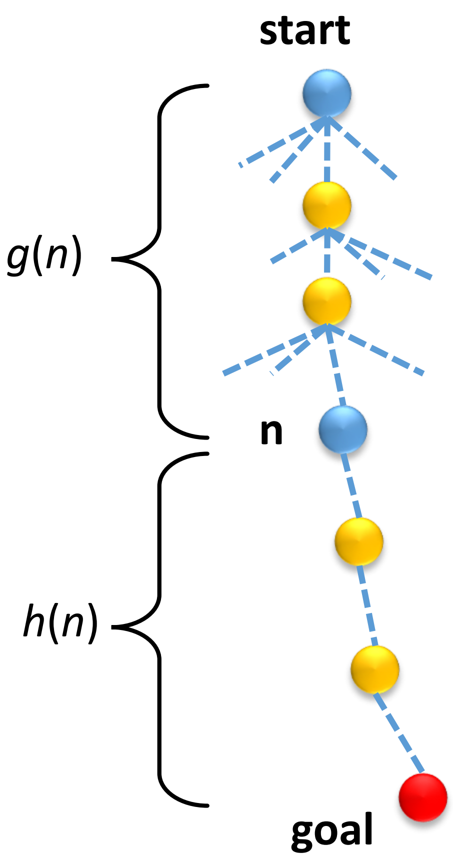
- **Optimal??** No. (A-S-F-B = 450, shorter journey is possible, ie A-S-R-P-B = 418)

A* search

- Idea: avoid expanding paths that are already expensive

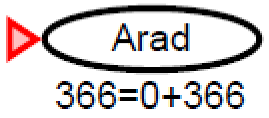
- Evaluation function $f(n) = g(n) + h(n)$
  $g(n)$ = cost so far to reach n
  $h(n)$ = estimated cost to goal from n
  $f(n)$ = estimated total cost of path through n to goal

- A* search uses an admissible heuristic
  i.e., $h(n) \leq h^*(n)$ where $h^*(n)$ is the true cost from n.
  (Also require $h(n) \geq 0$, so $h(G) = 0$ for any goal G.)
  E.g., $h_{SLD}(n)$ never overestimates the actual road distance



A* search example

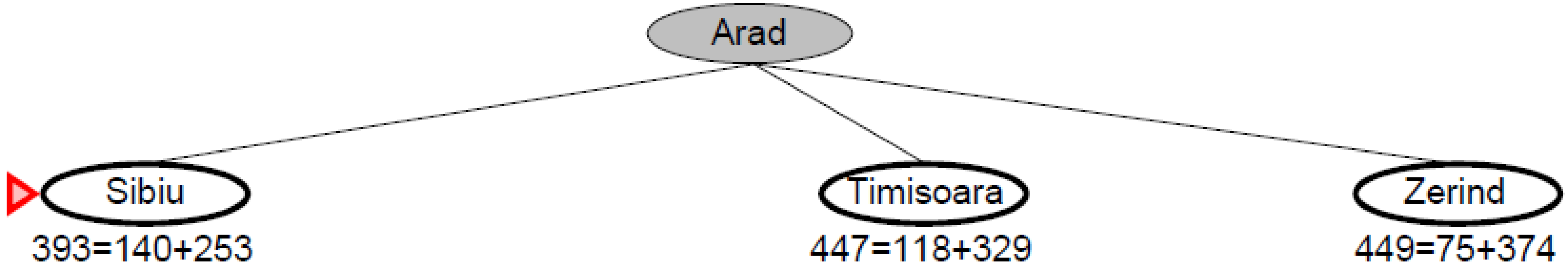
The initial state:

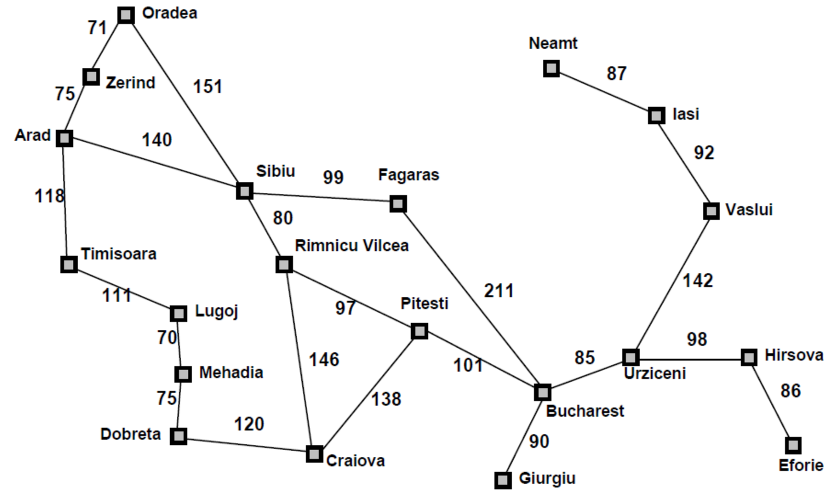


Stages in an A∗ search for Bucharest. Nodes are labeled with f = g + h. The h values are the straight-line distances to Bucharest

A* search example

After expanding Arad:



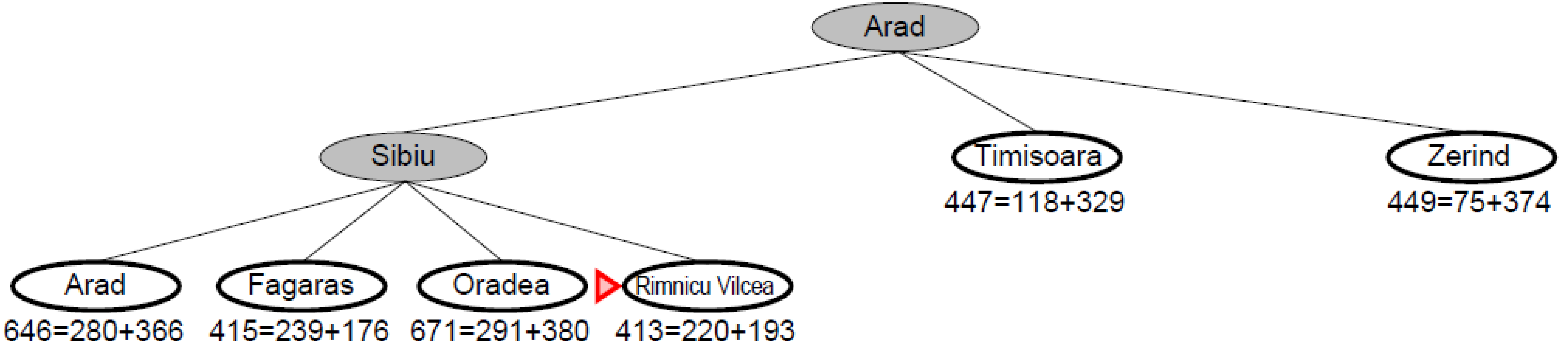


Straight−line distance to Bucharest

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

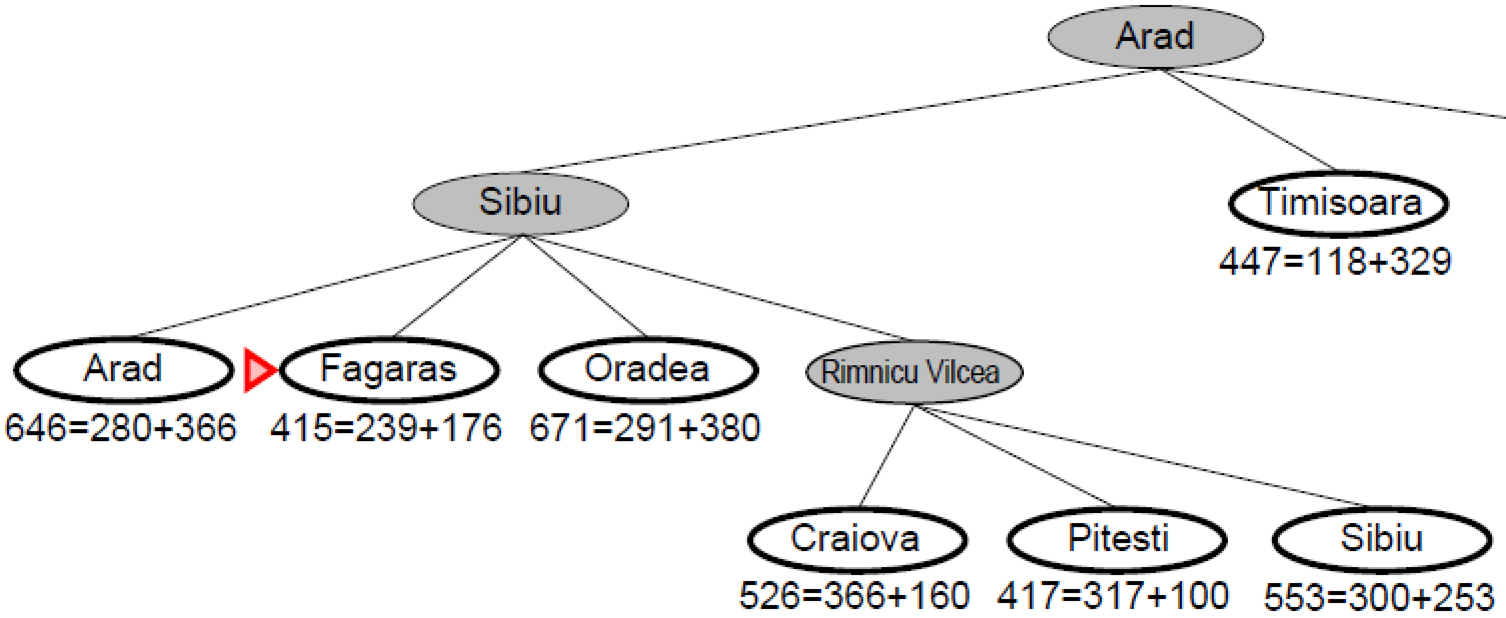
A* search example

After expanding Sibiu:



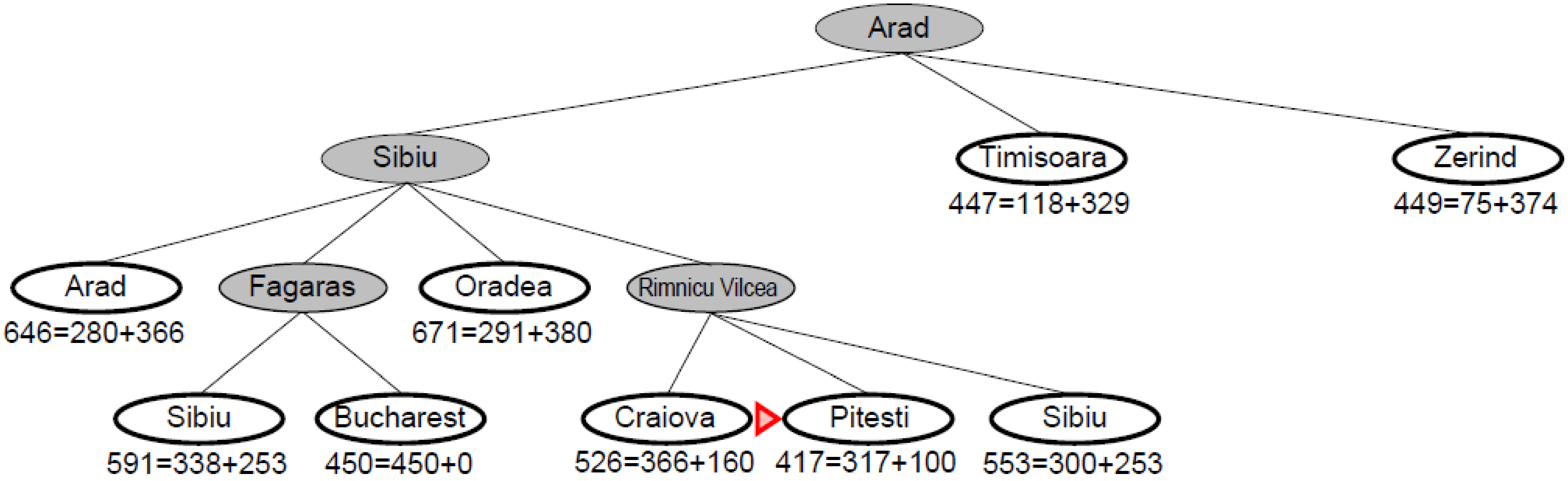
A* search example

After expanding Rimnicu Vilcea:



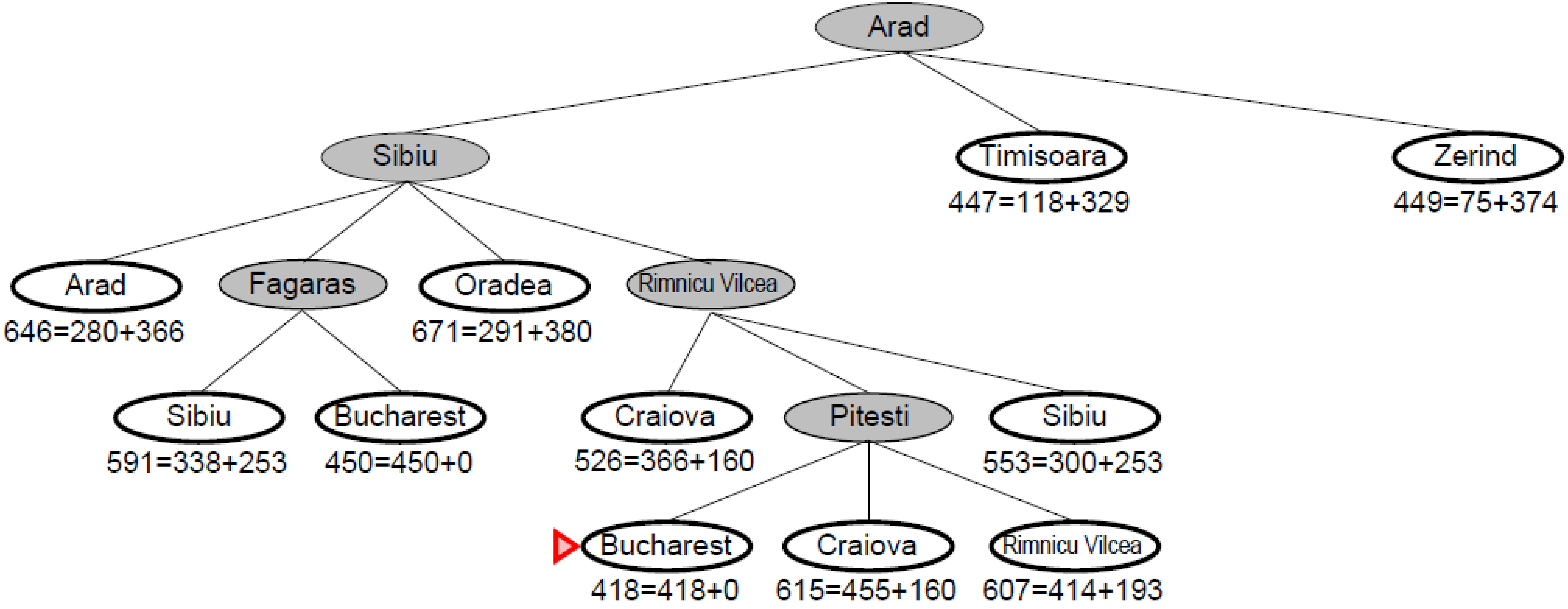
A* search example

After expanding Fagaras:



A* search example

After expanding Pitesti:



Properties of A* search

- Complete and optimal if $h(n)$ does not overestimate the true cost of a solution through n
- Time complexity
  - Exponential in [relative error of h x length of solution]
  - 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, UCS
- Space complexity
  - 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

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
  - Example: $h_{SLD}(n)$ (never overestimates the actual road distance)

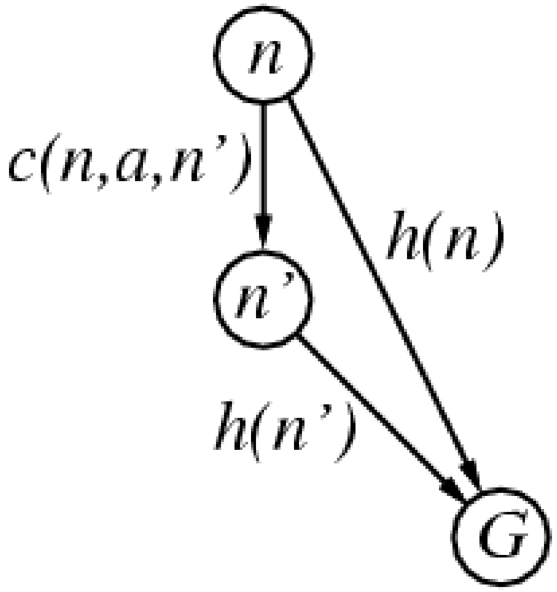
- **Theorem**: If $h(n)$ is admissible, A^* using `TREE-SEARCH` is optimal

Consistent heuristics (consistent => admissible)

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

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

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



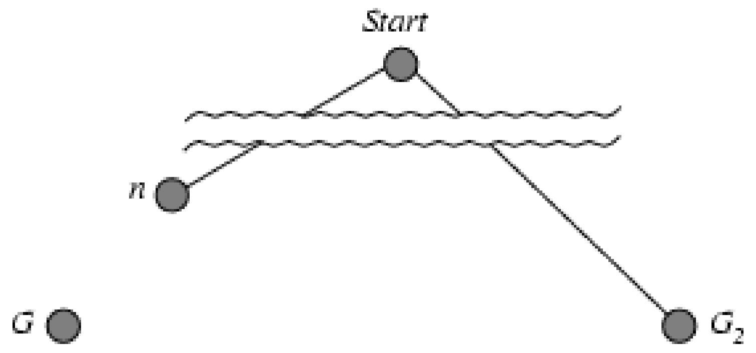
- If h is consistent, we have

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

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

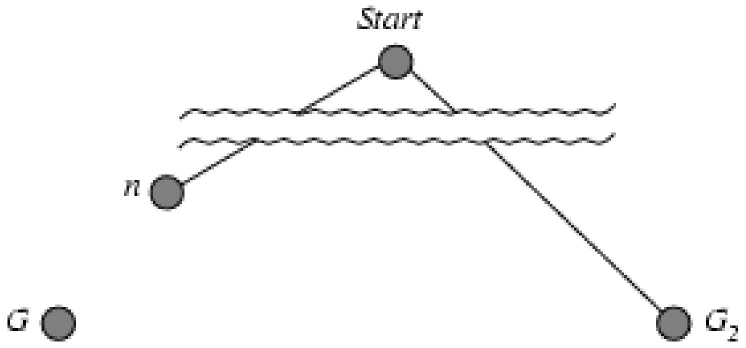
Optimality of A* (proof)

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



- $f(G_2) = g(G_2)$ since $h(G_2) = 0$ (true for any goal state)
- $g(G_2) > g(G)$ since G_2 is suboptimal
- $f(G) = g(G)$ since $h(G) = 0$
- $f(G_2) > f(G)$ from above

Optimality of A* (proof)



- $f(G_2) > f(G)$
- $h(n) \leq h^*(n)$ since h is admissible
- $g(n) + h(n) \leq g(n) + h^*(n)$
- $f(n) \leq f(G)$ ($f(G) = g(G) = g(n) + h^*(n)$ since n is on the shortest path to G)
- $f(n) < f(G_2)$
- Hence $f(G_2) > f(n)$, and A* will never select G_2 for expansion

Memory-bounded heuristic search

- To reduce memory requirements of A*
  - Iterative deepening A* (IDA*): cut-off used is the f-cost (rather than depth as in IDS)

- Other memory-bounded algorithms
  - Recursive best-first search (RBFS)
  - Simplified memory-bounded A* (SMA*)

- RBFS and SMA* are robust, optimal search algorithms that use limited amount of memory, and can often solve problems that A* can't as it runs out of memory

(Read textbook section 3.5.3 for details)

RBFS search example

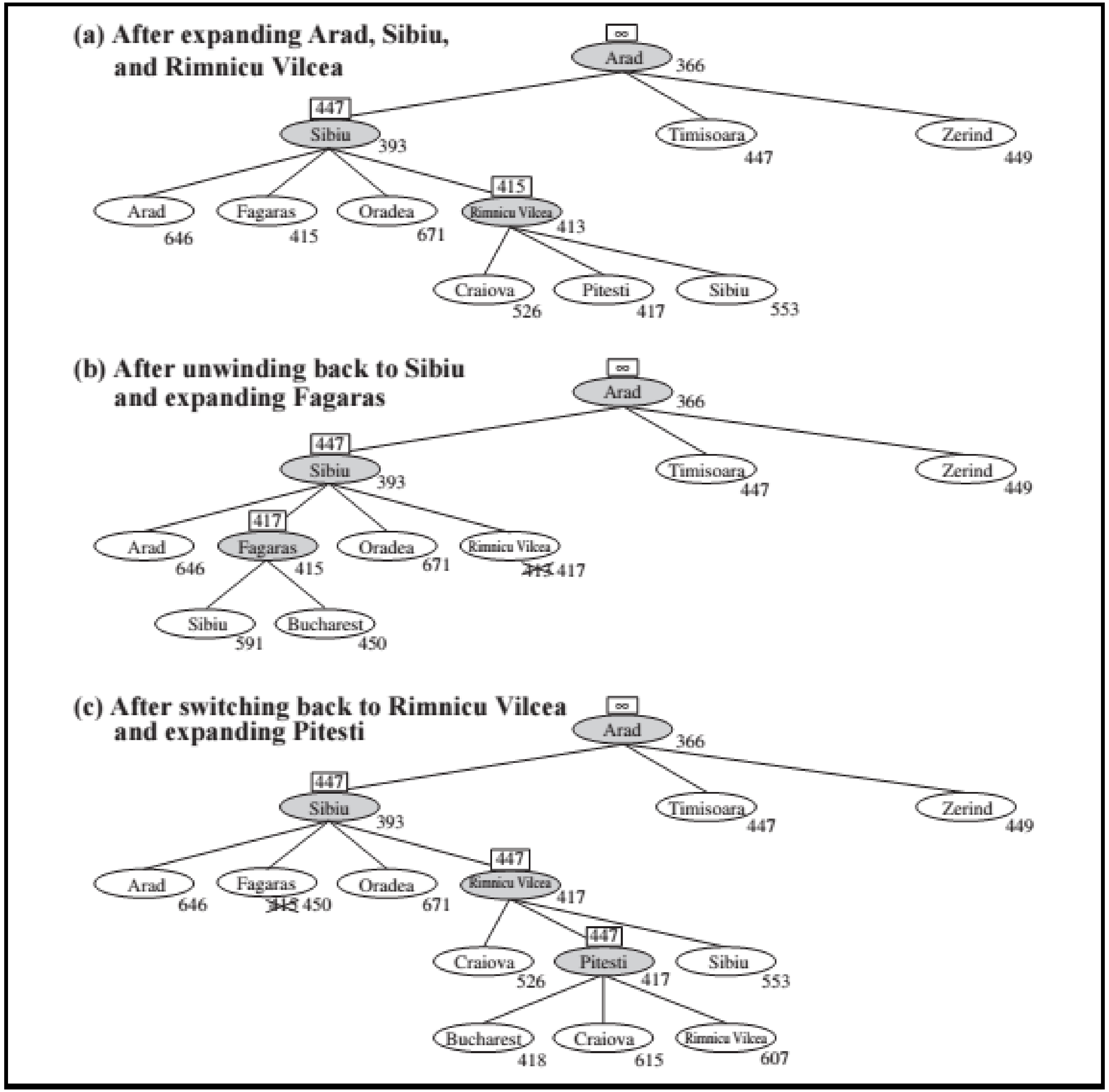


Figure 3.27 Stages in an RBFS search for the shortest route to Bucharest. The f-limit value for each recursive call is shown on top of each current node, and every node is labeled with its f-cost. (a) The path via Rimnicu Vilcea is followed until the current best leaf (Pitesti) has a value that is worse 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.

Admissible heuristics for the 8-puzzle

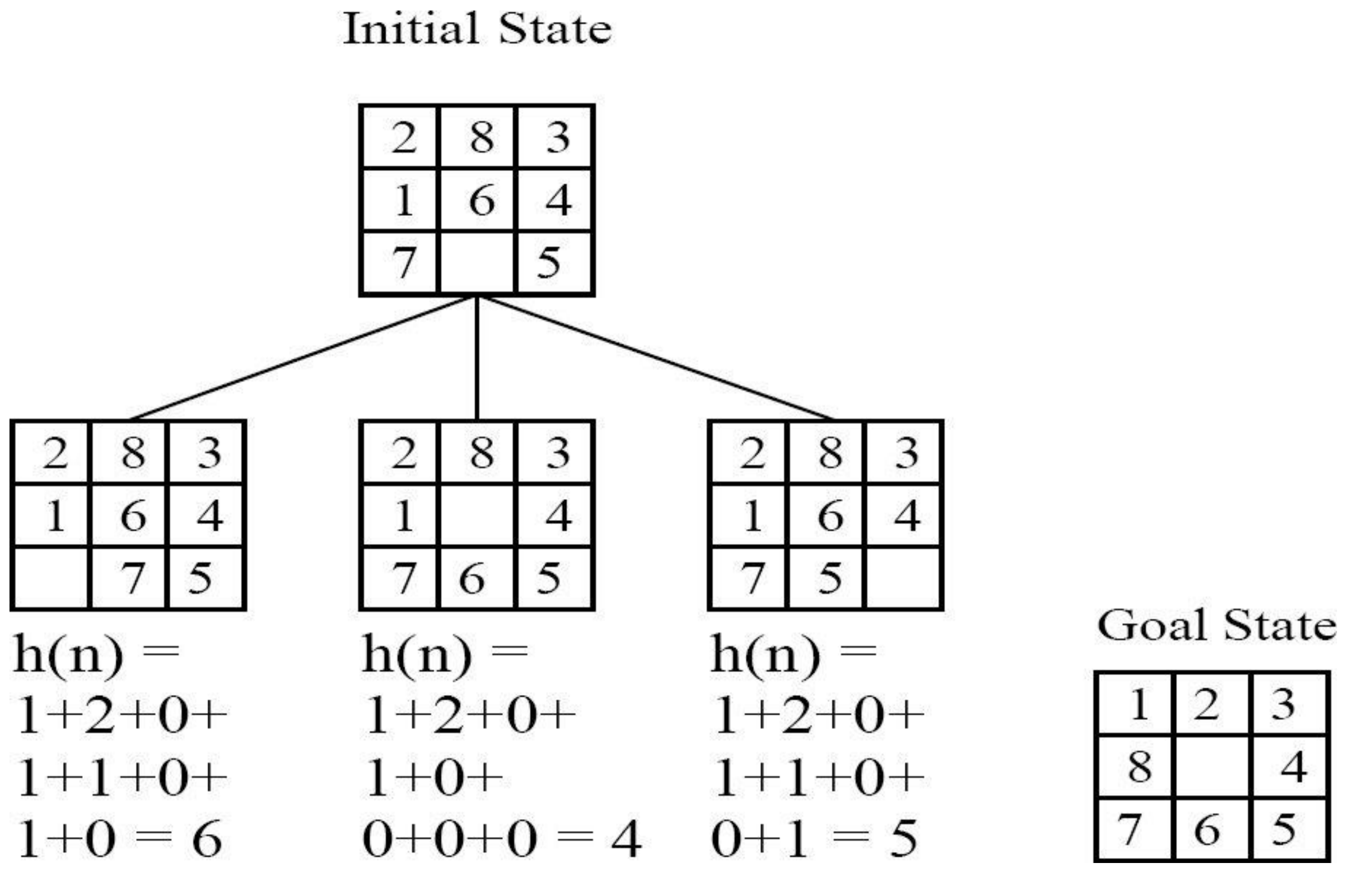
- Number of tiles out of place (h_1)

- Manhattan distance (h_2)
  - Sum of the distance of each tile from its goal position
  - Tiles can only move up or down → city blocks

1	2	3
4	5	6
7	8	

The 8-puzzle

- Using a heuristic evaluation function:
  - $h_2(n)$ = sum of the distance each tile is from its goal position



Goal state

1	2	3
4	5	6
7	8	

Current state

1	2	3
4	5	6
7		8

h_1=1

h_2=1

Current state

1	3	6
4	2	8
7		5

h_1=5

h_2=1+1+1+2+2=7

Heuristic functions

- Dominance/Informedness
  - if $h_2(n) \geq h_1(n)$ for all n (both admissible)
    then h_2 dominates h_1 and is better for search

- Typical search costs: (8 puzzle, d = solution length)
  - d = 12 IDS = 3,644,035 nodes
    $A^*(h_1)$ = 227 nodes
    $A^*(h_2)$ = 73 nodes

  - d = 24 IDS $\approx$ 54,000,000,000 nodes
    $A^*(h_1)$ = 39,135 nodes
    $A^*(h_2)$ = 1,641 nodes

Relaxed problems

- Admissible heuristics can be derived from the exact solution cost of a relaxed version of the problem
  - E.g. 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

- Key point: the optimal solution cost of a relaxed problem is not greater than the optimal solution cost of the real problem

Next…

- Local search
  - Hill climbing
  - Simulated annealing
  - Genetic algorithms