Informed Search

Lecture 3

Uninformed and informed searches

- Based on the search problems we can classify the search algorithms into two major types:
  - Uninformed search (Blind Search) Algorithms.

    NB: “covered in the previous lecture”
  - Informed search (Heuristic Search) Algorithms.

    NB: “will be covered shortly”

Uninformed Search algorithms

- These kind of algorithms does not contain any Knowledge of the domain such as how closeness to the goal or the location of the goal.
- They operate in a brute force way as it only includes information about how to traverse the tree and how to identify leaf and goals.
- Applies away in which search tree is searched without any information about the goal.
- They examine each node until it reaches the goal node and it can stop after that.

Examples

- Depth first search (DFS)
- Breadth first search (BFS)
- Uniform cost search (UCS)
- Depth limited search
- Iterative Deeping depth first search
- Bidirectional search

Informed search Algorithms.

- They use domain knowledge, the problem information to the goal is available which can guide the search.
- Informed search strategies can find a solution more efficient than an uninformed search.
- Heuristic is away which might not always be guaranteed for the best solution but guaranteed to find a good solution in a reasonable time.
- They can solve much complex problem which could not be solved in other ways.

Examples

- Greedy Best First Search
- A* Search
- Hill climbing Search

Heuristic Function

- Is a function that estimates how close ia a state to a goal
- Designed for a particular search problem

  i.e: May differ from one problem to another

- Example: Euclidean distance or Manhattan distance for a path problem
- The accuracy of choosing the heuristic function affects the accuracy of the algorithm.

A heuristic function

- Let evaluation function *h(n)* (heuristic)
  - $h(n)$ = *estimated* cost of the cheapest path from node *n* to goal node.
  - If *n* is goal then $h(n)=0$

Best First Search

- Uses an evaluation function f(n) for each node and the node to be expanded is the node n with the smallest f(n)
- Example
  - A* Search
  - Greedy best first search

A Quick Review

- g(n) = cost from the initial state to the current state n

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

- f(n) = evaluation function to select a node for expansion (usually the lowest cost node)

Examples

- Greedy Best First Search
- A* Search
- Hill climbing Search

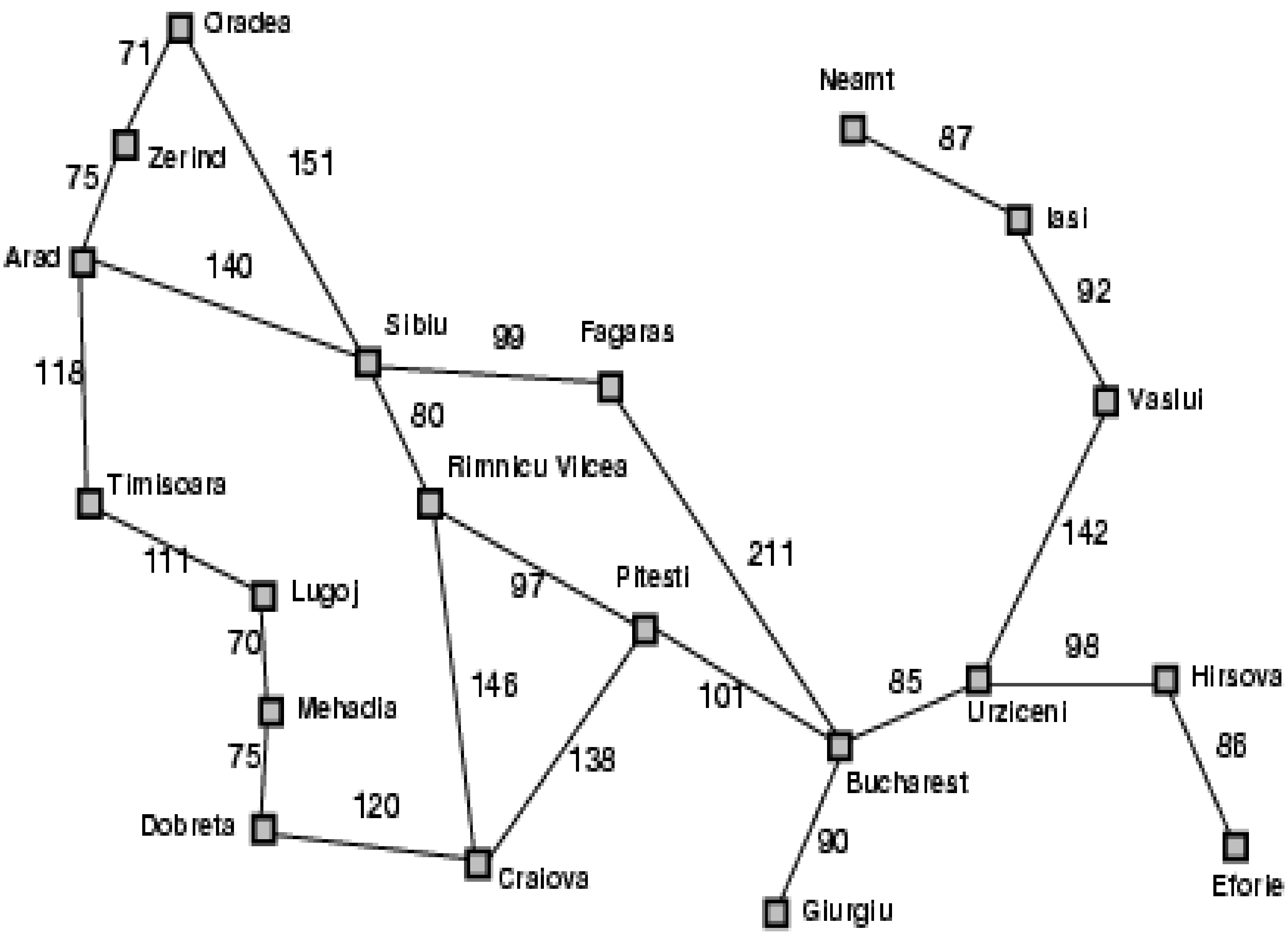
Greedy best-first search

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

= estimate of cost from *n* to *goal*

- Ignores the path cost
- Greedy best-first search expands the node that appears to be closest to goal

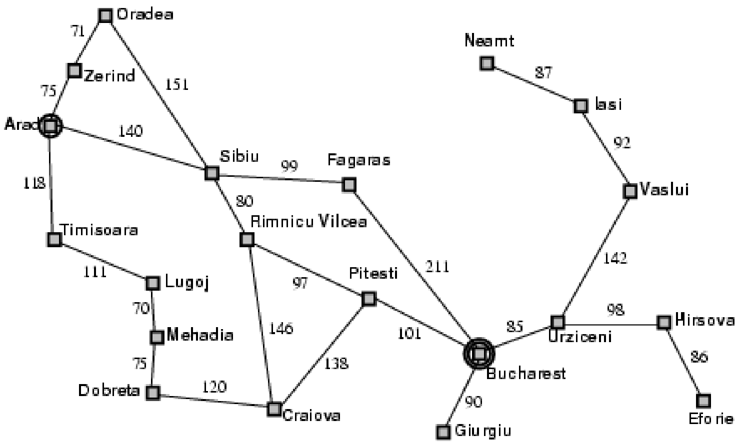
Romania with step costs in km



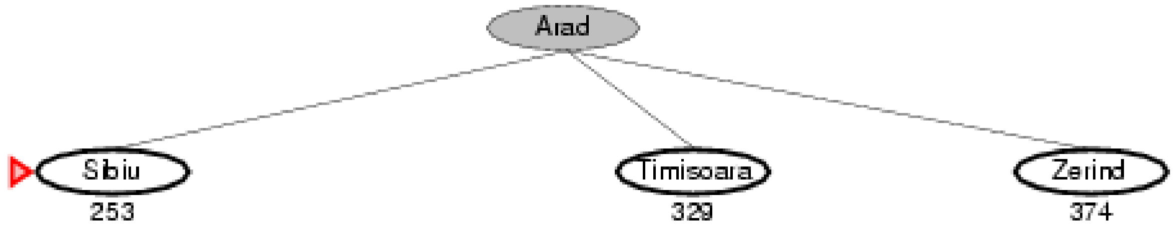
Straight–line distance to Bucharest

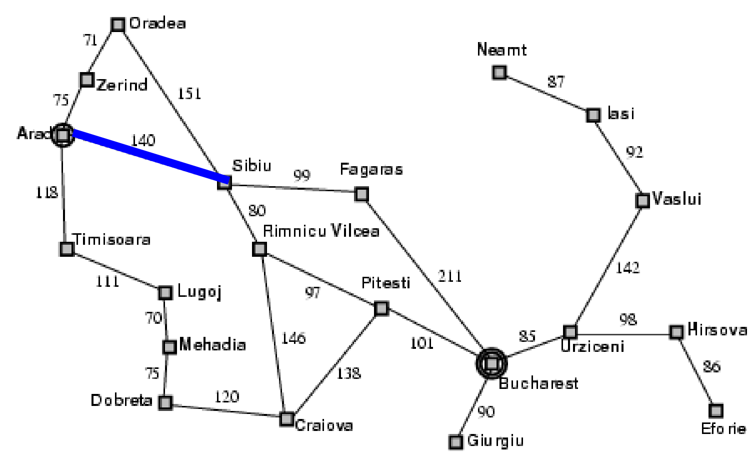
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy best-first search example

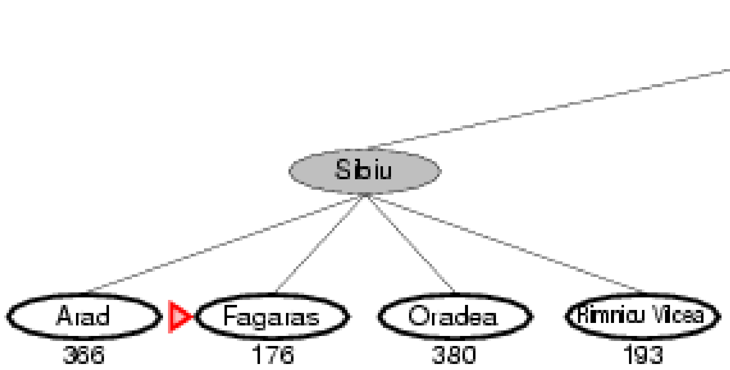


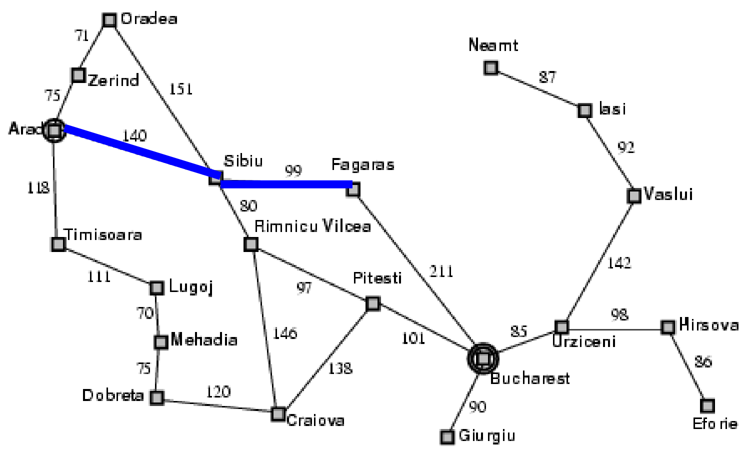
Greedy best-first search example



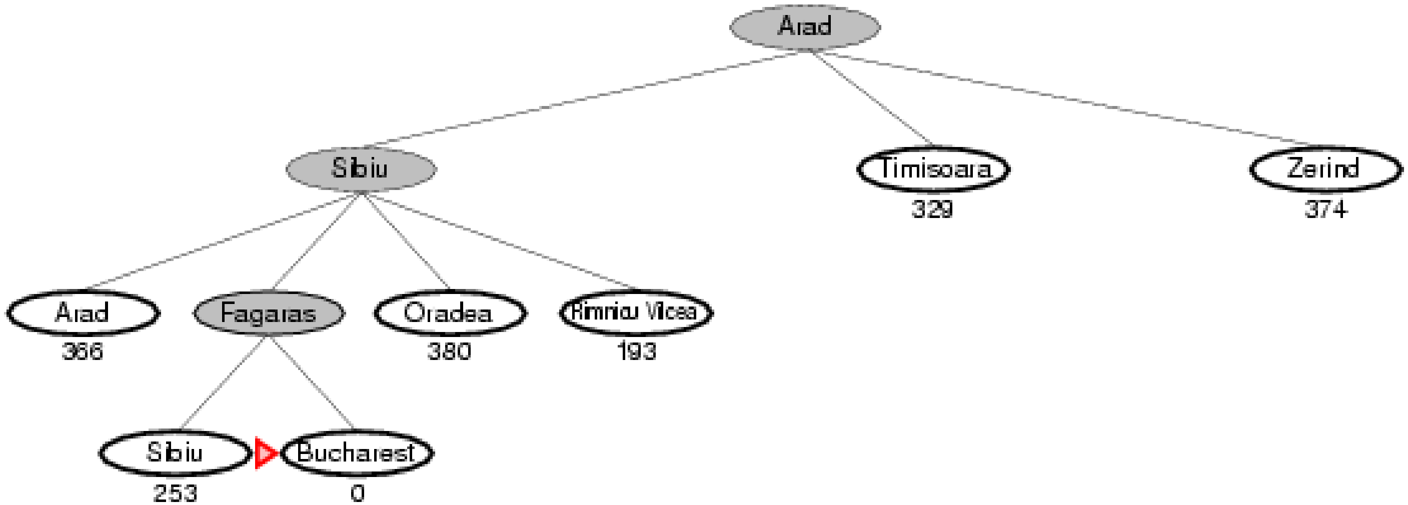


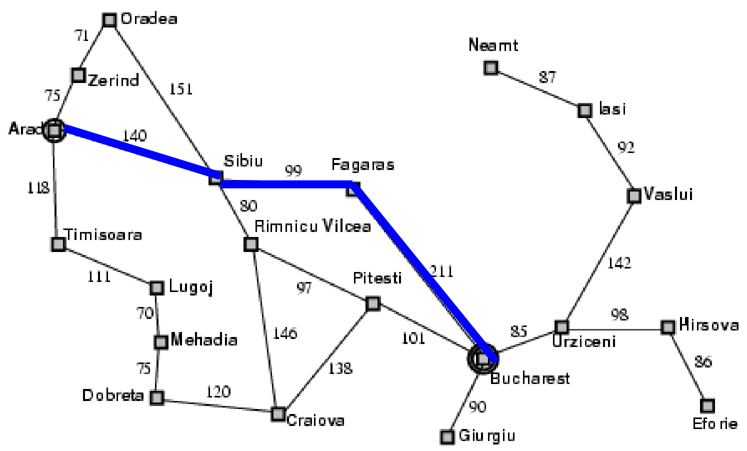
Greedy best-first search example



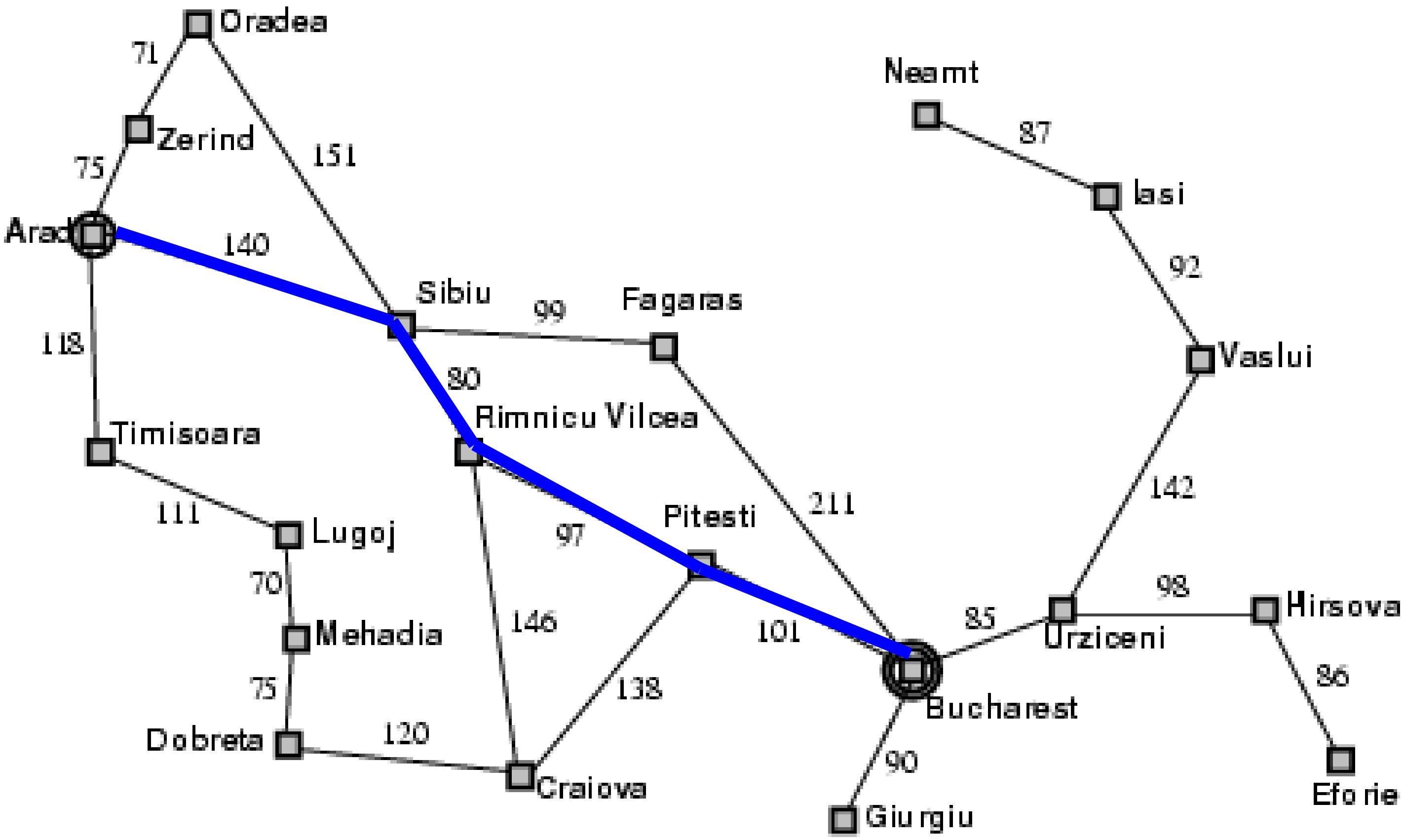


Greedy best-first search example





Optimal Path



Greedy Best-First Search Algorithm

Input: State Space

Ouput: *failure* or path from a start state to a goal state.

Assumptions:

- *L* is a list of nodes that have not yet been examined ordered by their *h* value.
- The state space is a tree where each node has a single parent.

1. Set L to be a list of the initial nodes in the problem.
2. While L is not empty
   1. Pick a node *n* from the front of L.
   2. If *n* is a goal node
      1. stop and return it and the path from the initial node to *n*.

      Else
      1. remove *n* from L.
      2. For each child c of *n*
         1. insert *c* into L while preserving the ordering of nodes in L and labelling *c* with its path from the initial node as well as its *h* value.

         End for

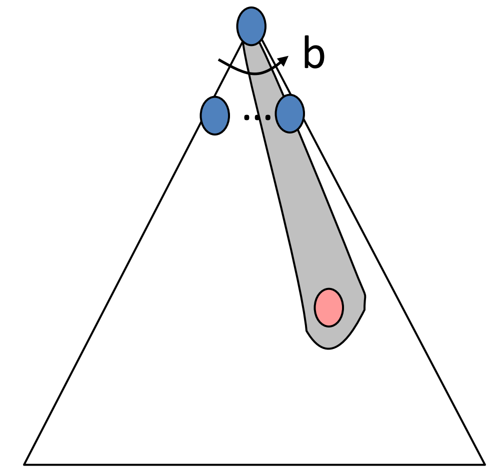
      End if

   End while

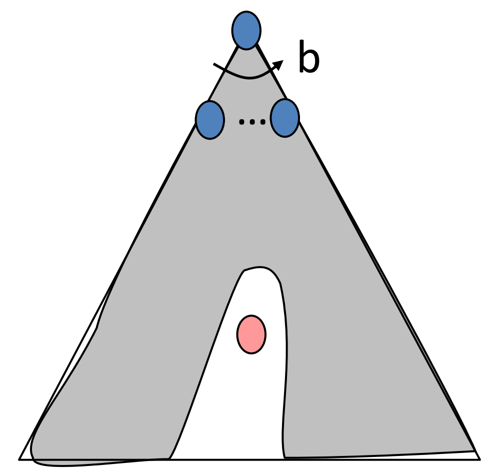
   Return *failure*

Greedy BF Search

- Strategy: expand a node that you think is closest to a goal state
  - Heuristic: estimate of distance to nearest goal for each state



- A common case:
  - Best-first takes you straight to the (wrong) goal



- Worst-case: like a badly-guided DFS

Properties of greedy best-first search

- Complete?
  - Not 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

Examples

- Greedy Best First Search
- A* Search
- Hill climbing Search

A* Search Algorithm

Evaluation function ***f(n)*** = ***h(n)*** + ***g(n)***

h(n) estimated cost to goal from *n*

g(n) cost so far to reach *n*

A* uses admissible heuristics, i.e.,

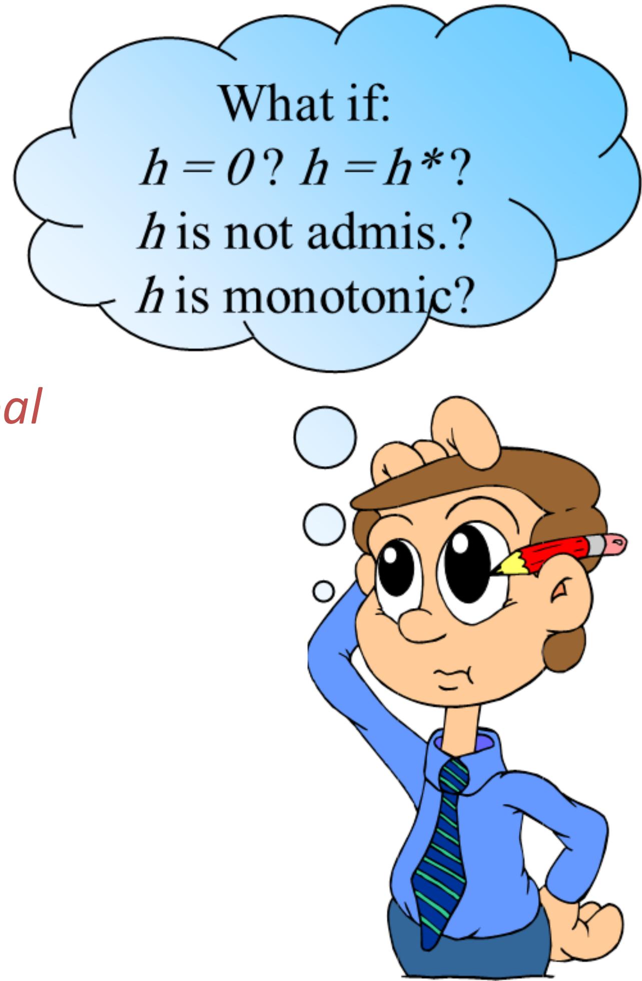
$h(n) \leq h^*(n)$ where $h^*(n)$ is the

true cost from *n*.

A* Search finds the optimal path

A* search

- Best-known form of best-first search.
- Idea: avoid expanding paths that are already expensive.
- Combines uniform-cost and greedy search
- Evaluation function $f(n)=g(n) + h(n)$
  - $g(n)$ the cost (so far) to *reach* the node
  - $h(n)$ estimated cost to *get from the node to the goal*
  - $f(n)$ estimated *total cost* of path through *n* to goal
- Implementation: Expand the node n with minimum $f(n)$

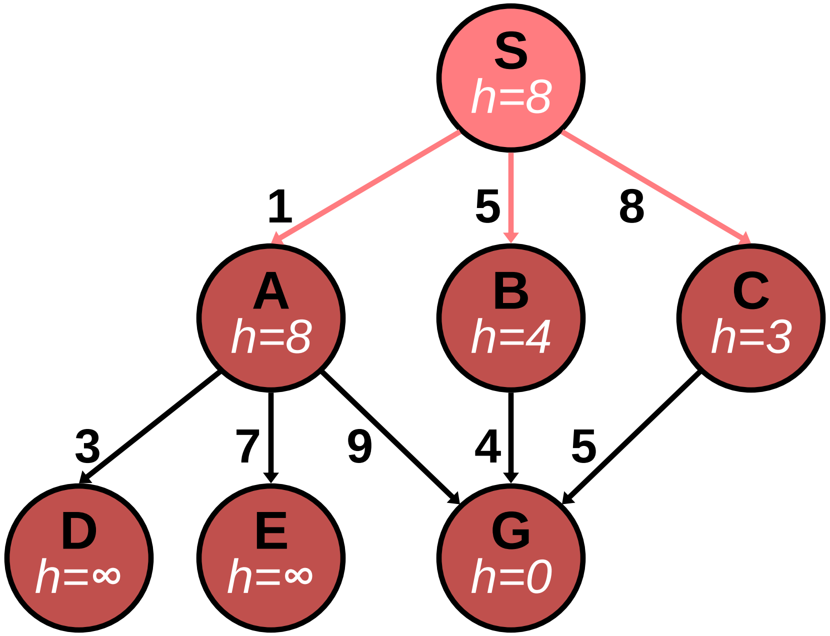


A* Search

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

# of nodes tested: 1, expanded: 1

expnd. node	**Frontier**
	{S:8}
S not goal	{A:1+8,B:5+4,C:8+3}

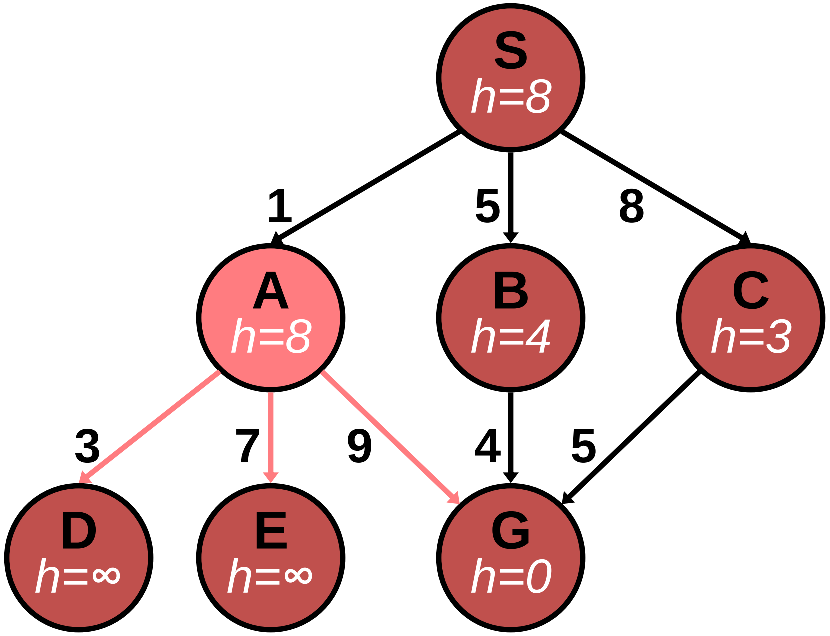


A* Search

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

# of nodes tested: 2, expanded: 2

expnd. node	**Frontier**
	{S:8}
S	{A:9,B:9,C:11}
A not goal	{B:9,G:1+9+0,C:11, D:1+3+∞,E:1+7+∞}

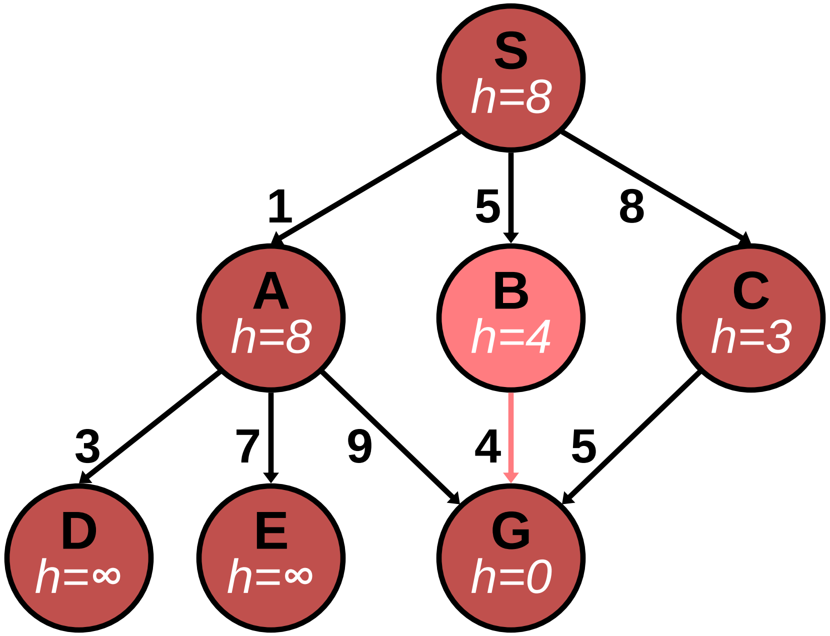


A* Search

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

# of nodes tested: 3, expanded: 3

expnd. node	**Frontier**
	{S:8}
S	{A:9,B:9,C:11}
A	{B:9,G:10,C:11,D:∞,E:∞}
B not goal	{G:5+4+0,~~G:10~~,C:11, D:∞,E:∞} replace

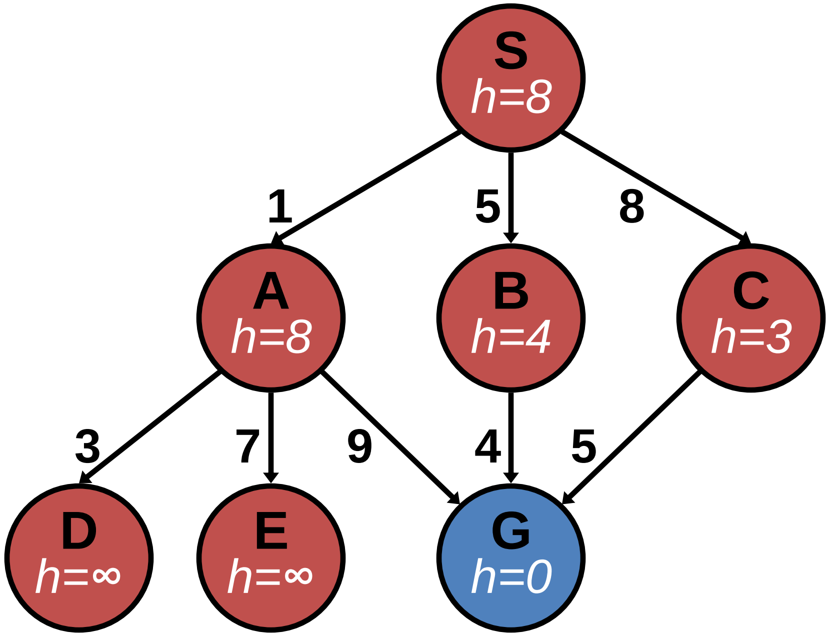


A* Search

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

# of nodes tested: 4, expanded: 3

expnd. node	Frontier
	{S:8}
S	{A:9,B:9,C:11}
A	{B:9,G:10,C:11,D:∞,E:∞}
B	{G:9,C:11,D:∞,E:∞}
G goal	{C:11,D:∞,E:∞} not expanded

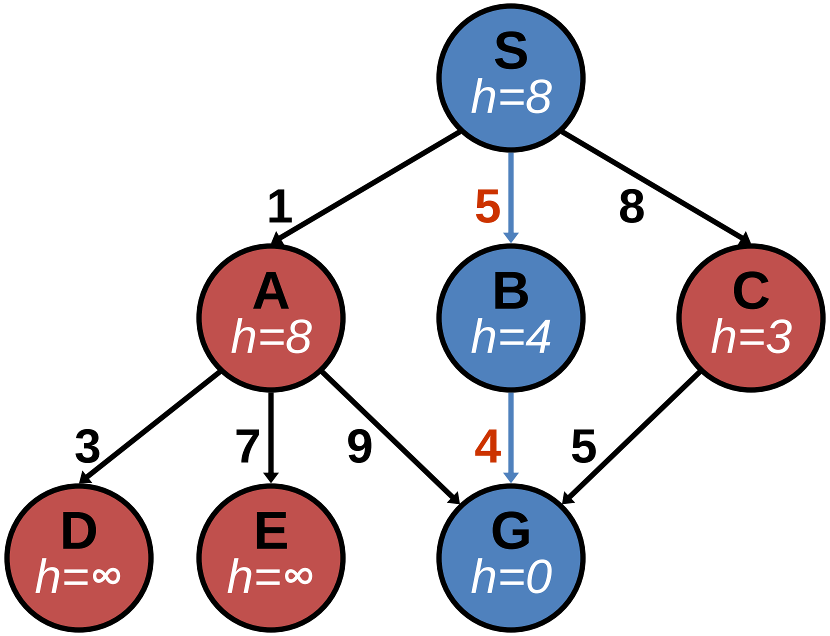


A* Search

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

# of nodes tested: 4, expanded: 3

expnd. node	Frontier
	{S:8}
S	{A:9,B:9,C:11}
A	{B:9,G:10,C:11,D:∞,E:∞}
B	{G:9,C:11,D:∞,E:∞}
G	{C:11,D:∞,E:∞}

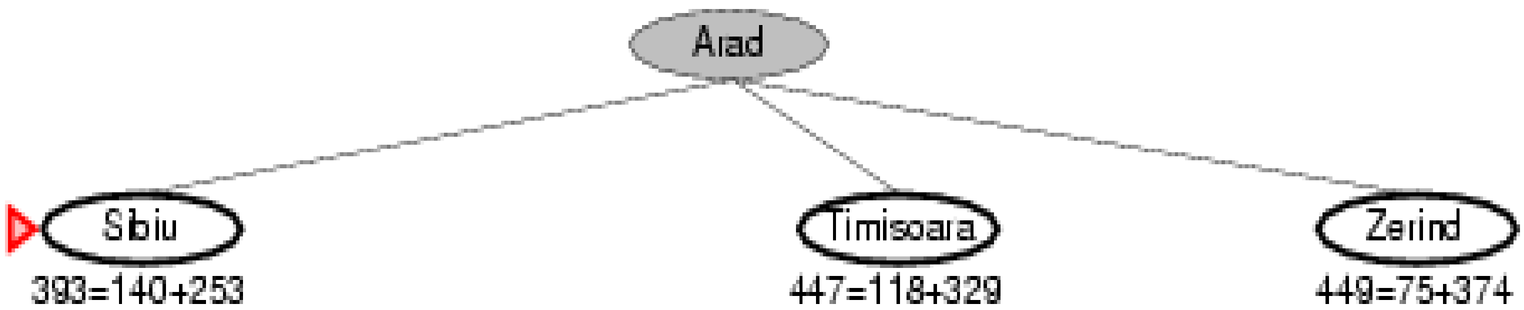


- *Pretty fast and optimal*

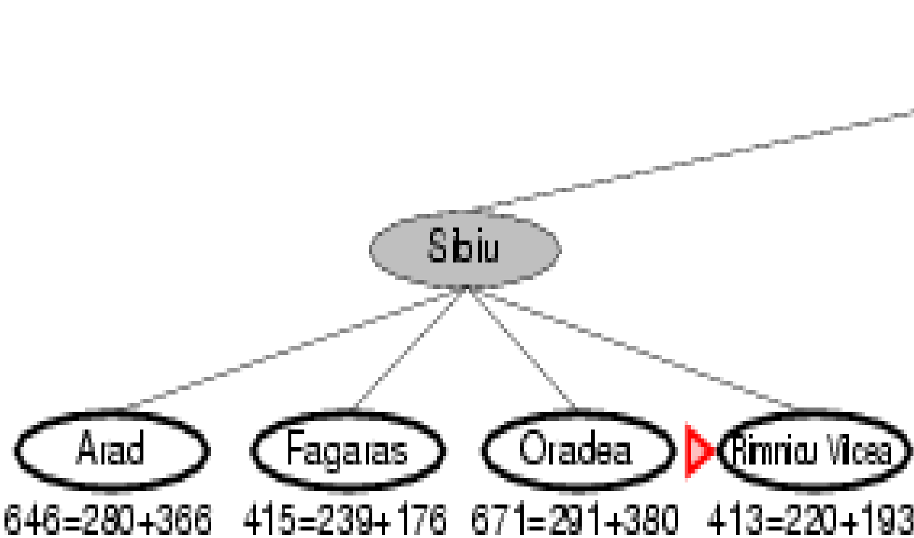
path: S,B,G
cost: 9

A^* search example

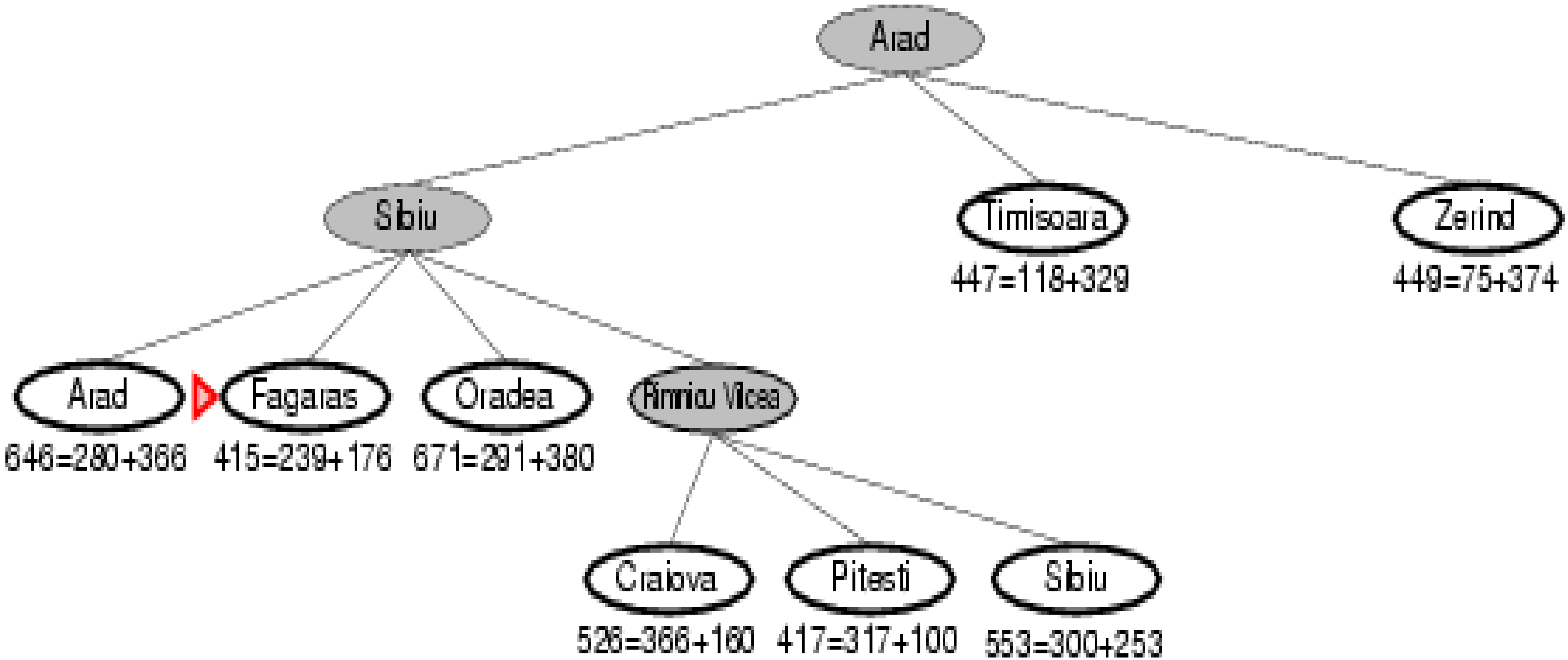
A^* search example



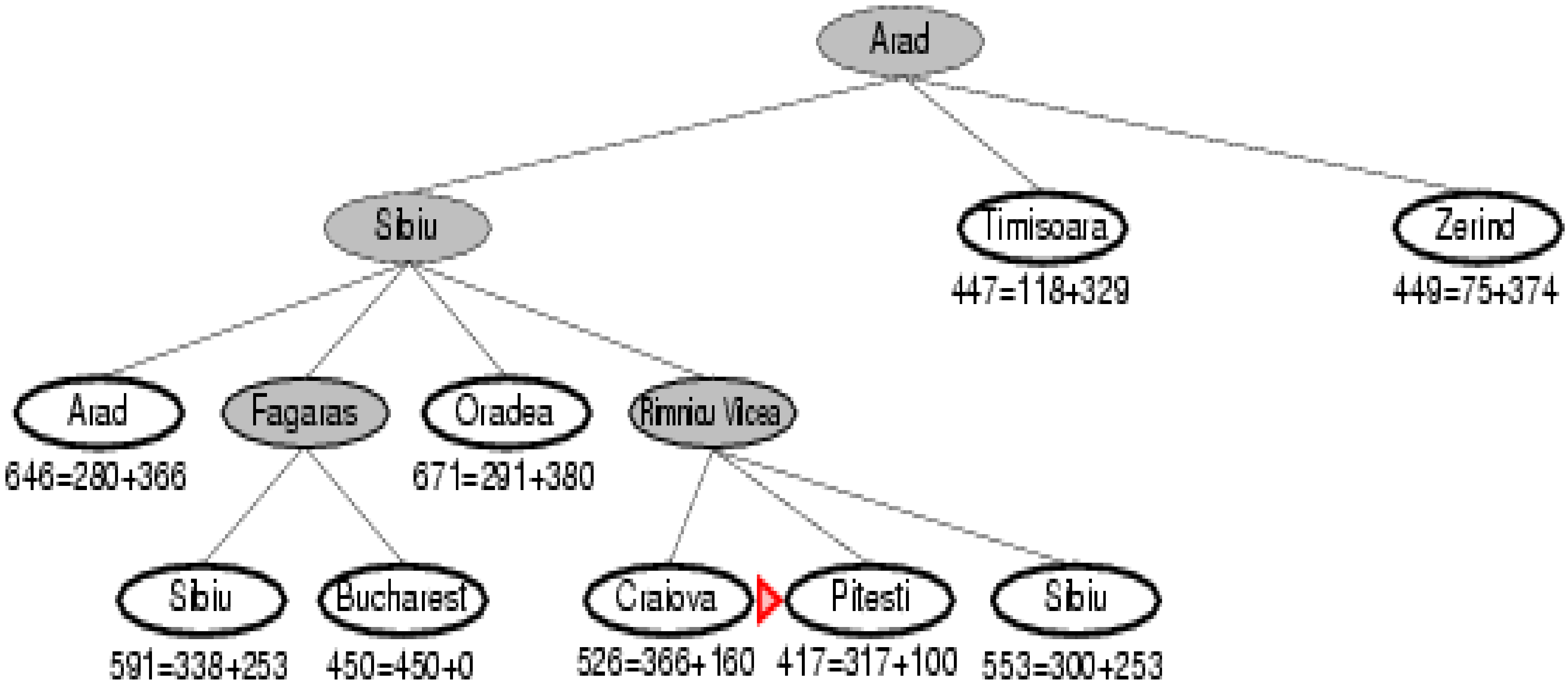
A* search example



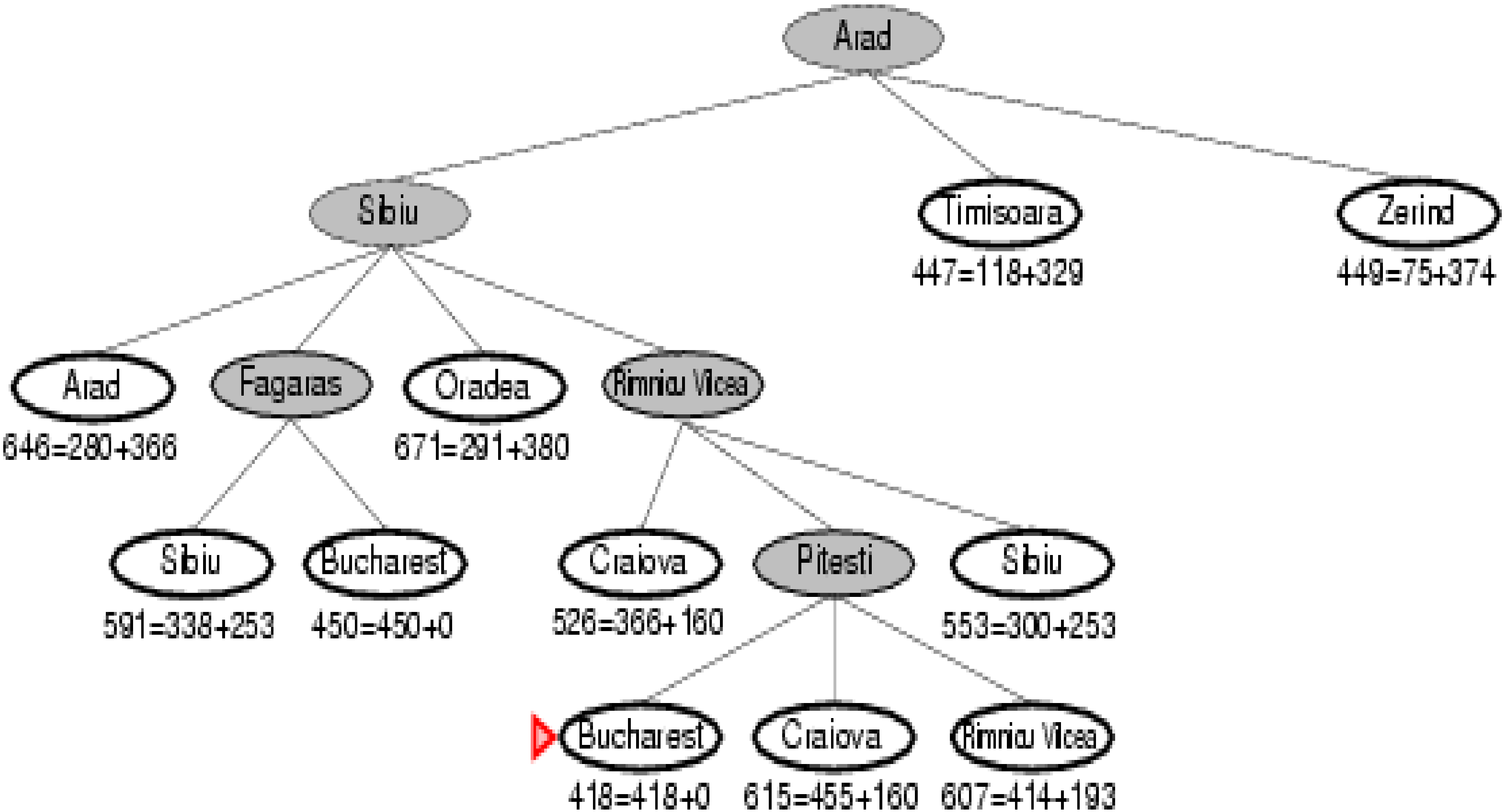
A* search example

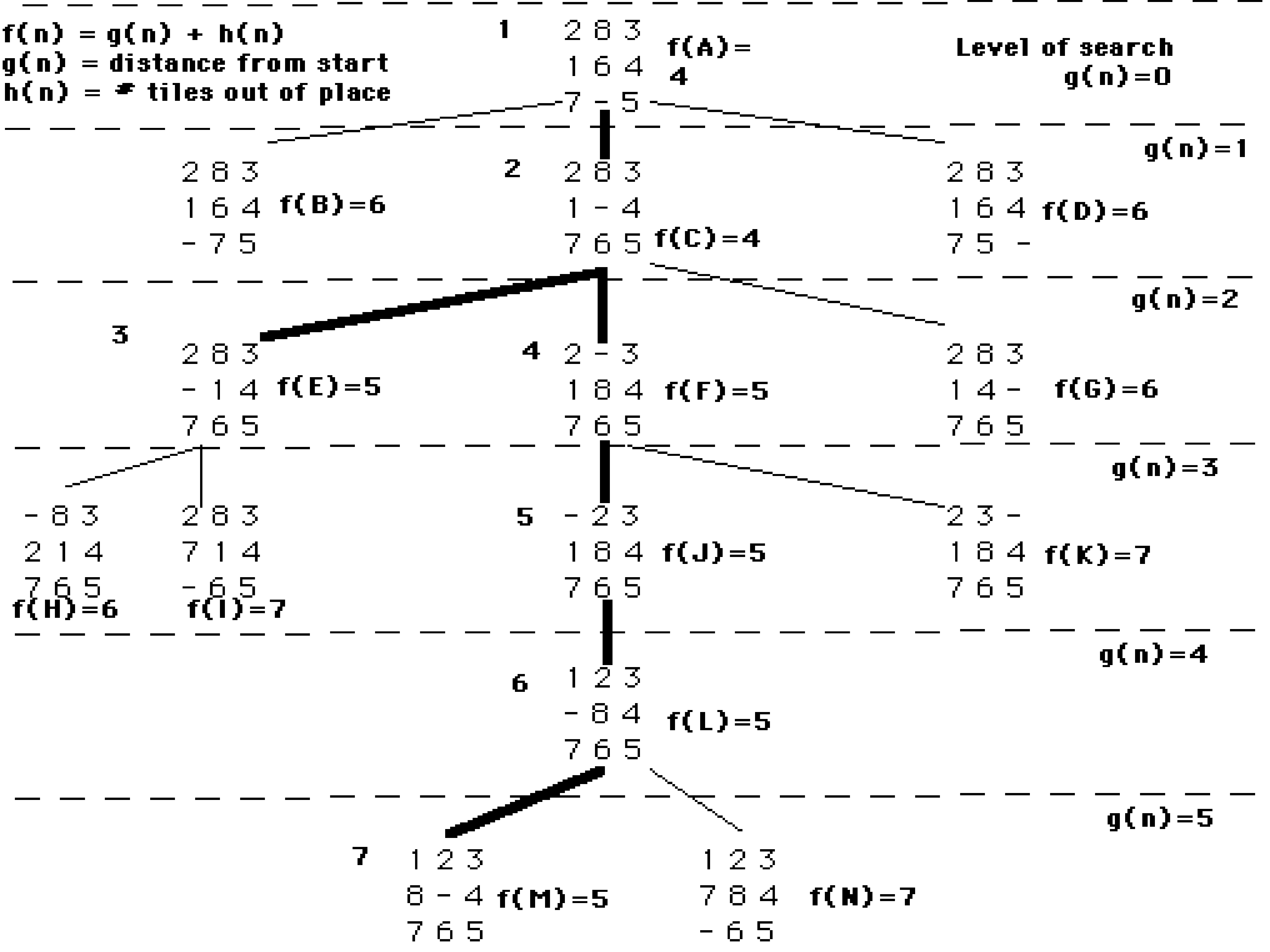


A* search example



A* search example




f(n) = g(n) + h(n)
g(n) = distance from start
h(n) = # tiles out of place
Level of search
g(n)=0
1
2 8 3
1 6 4
7 - 5
f(A)= 4
g(n)=1
2 8 3
1 6 4
- 7 5
f(B)=6
2
2 8 3
1 - 4
7 6 5
f(C)=4
2 8 3
1 6 4
7 5 -
f(D)=6
g(n)=2
3
2 8 3
- 1 4
7 6 5
f(E)=5
4
2 - 3
1 8 4
7 6 5
f(F)=5
2 8 3
1 4 -
7 6 5
f(G)=6
g(n)=3
- 8 3
2 1 4
7 6 5
f(H)=6
2 8 3
7 1 4
- 6 5
f(I)=7
5
- 2 3
1 8 4
7 6 5
f(J)=5
2 3 -
1 8 4
7 6 5
f(K)=7
g(n)=4
6
1 2 3
- 8 4
7 6 5
f(L)=5
g(n)=5
7
1 2 3
8 - 4
7 6 5
f(M)=5
1 2 3
7 8 4
- 6 5
f(N)=7

Properties of A*

- Complete? Yes (unless there are infinitely many nodes with f $\leq$ *f(G)*)

- Optimal? Yes

A*: Summary

- A* uses both backward costs and (estimates of) forward costs
- A* is optimal with admissible / consistent heuristics
- Heuristic design is key: often use relaxed problems

Examples

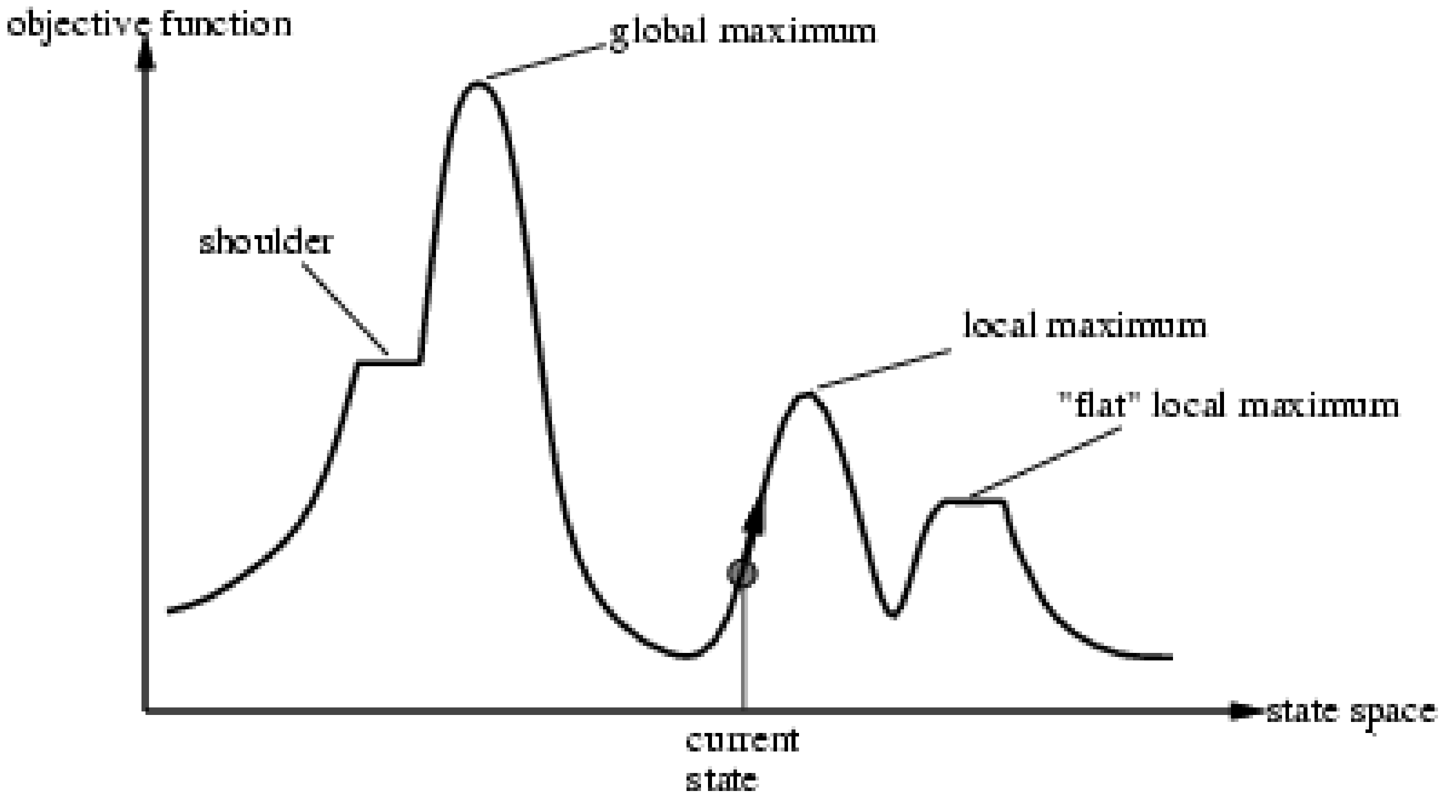
- Greedy Best First Search
- A* Search
- Hill climbing Search

Hill Climbing Search

- For artefact-only problems (don’t care about the path)
- Depends on some e(state)
  - Hill climbing tries to maximise score e
- Randomly choose a state
  - Only choose actions which improve e
  - If cannot improve e, then perform a **random restart**
    - Choose another random state to restart the search from
- Only ever have to store one state (the present one)
  - Can’t have cycles as e always improves

Hill-climbing search

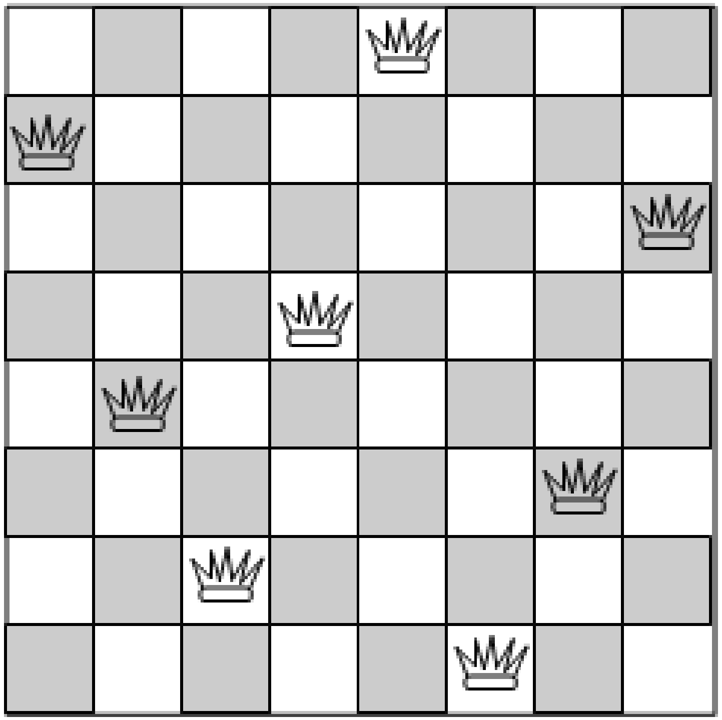
- Problem: depending on initial state, can get stuck in local maxima



Hill Climbing - Algorithm

1. Pick a random point in the search space
2. Consider all the neighbors of the current state
3. Choose the neighbor with the best quality and move to that state
4. Repeat 2 thru 4 until all the neighboring states are of lower quality
5. Return the current state as the solution state

Example: 8 Queens



- Place 8 queens on board
  - So no one can “take” another
- Gradient descent search
  - Throw queens on randomly
  - e = number of pairs which can attack each other
  - Move a queen out of other’s way
    - Decrease the evaluation function
  - If this can’t be done
    - Throw queens on randomly again

Hill-climbing search

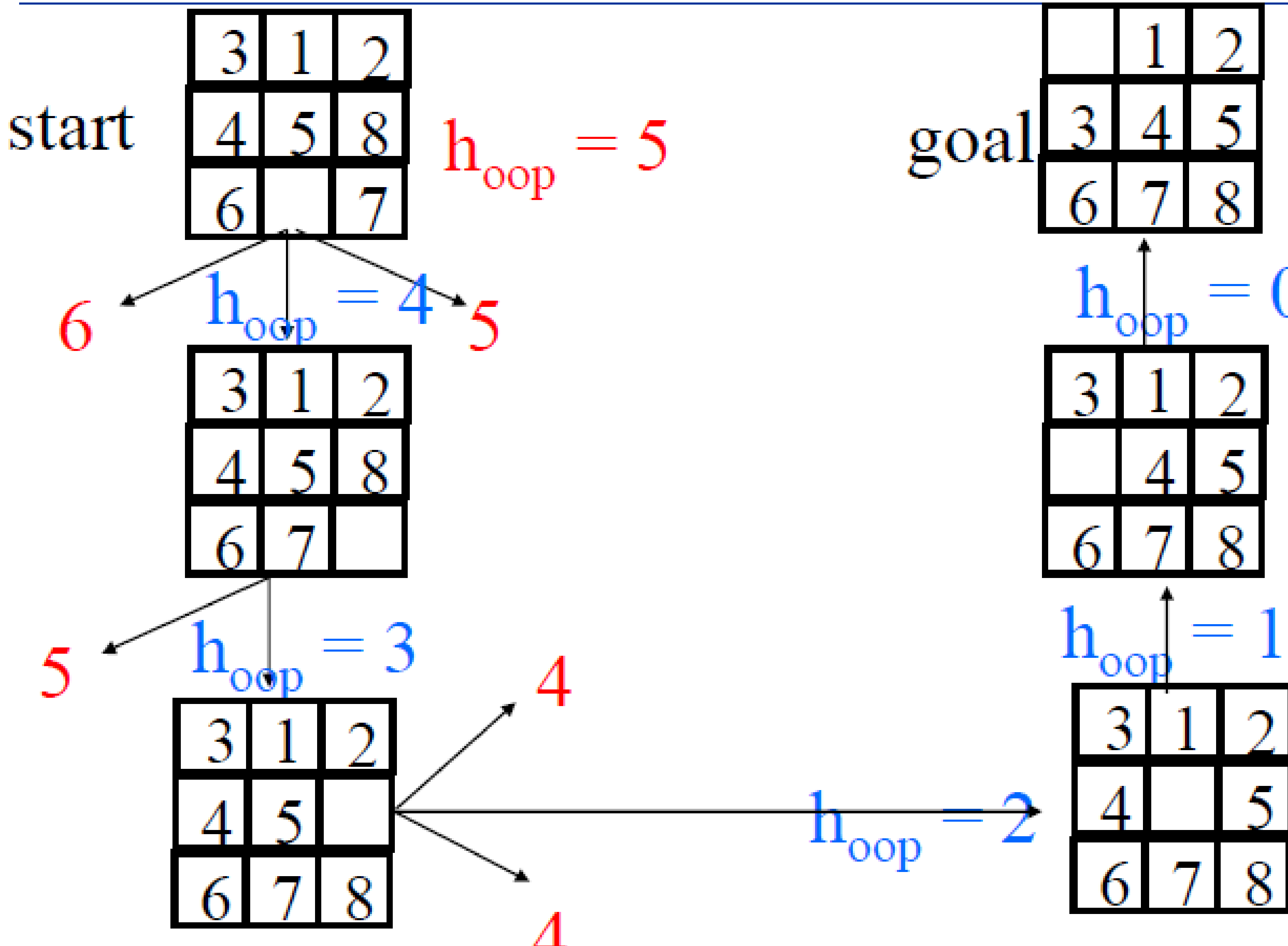
- Looks one step ahead to determine if any successor is better than the current state; if there is, move to the best successor.
- <u>**Rule: If**</u> there exists a successor s for the current state n such that
  - $h(s) < h(n)$ and
  - $h(s) \leq h(t)$ for all the successors t of n,

  then move from n to s. Otherwise, halt at n.

Hill-climbing search

- Similar to Greedy search in that it uses *h*(), but does not allow backtracking or jumping to an alternative path since it doesn't "remember" where it has been.
- If Hill climbing failed to find the goal it will start from another random node searching for the goal node(may choose the same failed node ☹).

Hill climbing example I (*minimizing* h)



Properties of Hill Climbing

- Complete? NO

- Optimal? NO