CS 362: Intelligent Systems

UNIT 4

HEURISTIC SEARCH

Outline

4.0 Introduction

4.1 An Algorithm for Heuristic Search

4.2 Admissibility, Monotonicity, and Informedness

4.3 Using Heuristics I n Games

4.4 Complexity Issues

Heuristic definition

George Polya (1945) defines heuristic as :

"the study of the methods and rules of discovery and invention"

- The verb *eurisco*, which means *"I discover."*
- When Archimedes emerged from his famous bath clutching the golden crown,

  he shouted "Eureka!" meaning "I have found it!".

- In state space search,

  Heuristics are formalized as rules for choosing those branches in a state space that are most likely to lead to an acceptable problem solution.

Heuristic and AI problem solving

AI problem solvers employ heuristics in two basic situations:

1) A problem may not have an exact solution because of inherent ambiguities in the problem statement or available data.

- Medical diagnosis is an example of this. A given set of symptoms may have several possible causes; doctors use heuristics to choose the most likely diagnosis and formulate a plan of treatment.
- Vision is example of an inexact problem. Vision systems often use heuristics to select the most likely of several possible interpretations of as scene.

2) A problem may have an exact solution, but the computational cost of finding it may be prohibitive

The problem has an exact solution but is too complex to allow for a *brute force* solution.

In many problems (such as chess),

Heuristic and AI problem solving

Heuristic or informed search:

- **Exploits additional knowledge about the problem that helps direct search to more promising paths.**
- **problem-solving strategies which in many cases find a solution faster than uninformed search. However , this is not guaranteed.**
- **Could require a lot more time and can even result in the solution not being found.**
- **Fallible because they rely on limited information, they may lead to a suboptimal solution or to a dead end.**

Heuristic and AI problem solving

Heuristic is:

- **an AI search technique that employs heuristic for its moves.**
- **a rule of thumb that probably leads to a solution. Heuristics play a major role in search strategies because of exponential nature of the most problems.**
- **help to reduce the number of alternatives from an exponential number to a polynomial number.**

In Artificial Intelligence, heuristic search has a general meaning, and a more specialized technical meaning. In a general sense, the term heuristic is used for any advice that is often effective, but is not guaranteed to work in every case. Within the heuristic search architecture, however, the term heuristic usually refers to the special case of a heuristic evaluation function.

Informed or Heuristic search Strategies

Types of informed search strategies:

- Best-First Search
  - Greedy search
  - A* search
  - Best-First Search Analysis
  - Heuristic Functions
- ~~Memory Bounded Search~~
  - ~~Iterative Deepening A* Search~~
  - ~~Recursive best-first search~~
  - ~~Simplified Memory Bounded A*~~
- Iterative Improvement Algorithms
  - Hill-Climbing Search
  - ~~Simulated Annealing~~
  - ~~Tabu Search~~
  - ~~and many more.~~

Heuristic Information

- In order to solve larger problems, domain-specific knowledge must be added to improve search efficiency.

- Information about the problem include:
  - ✓ the nature of states,
  - ✓ cost of transforming from one state to another,
  - ✓ and characteristics of the goals.

This information can often be expressed in the form of heuristic evaluation function,

say f(n , g),

a function of the nodes n and/or the goals g.

Example (1): The game of tic-tac-toe

- Even if we use symmetry to reduce the search space of redundant moves,

- The number of possible paths through the search space is something like 12 x 7! = 60480. That is a measure of the amount of work that would have to be done by a brute-force search.

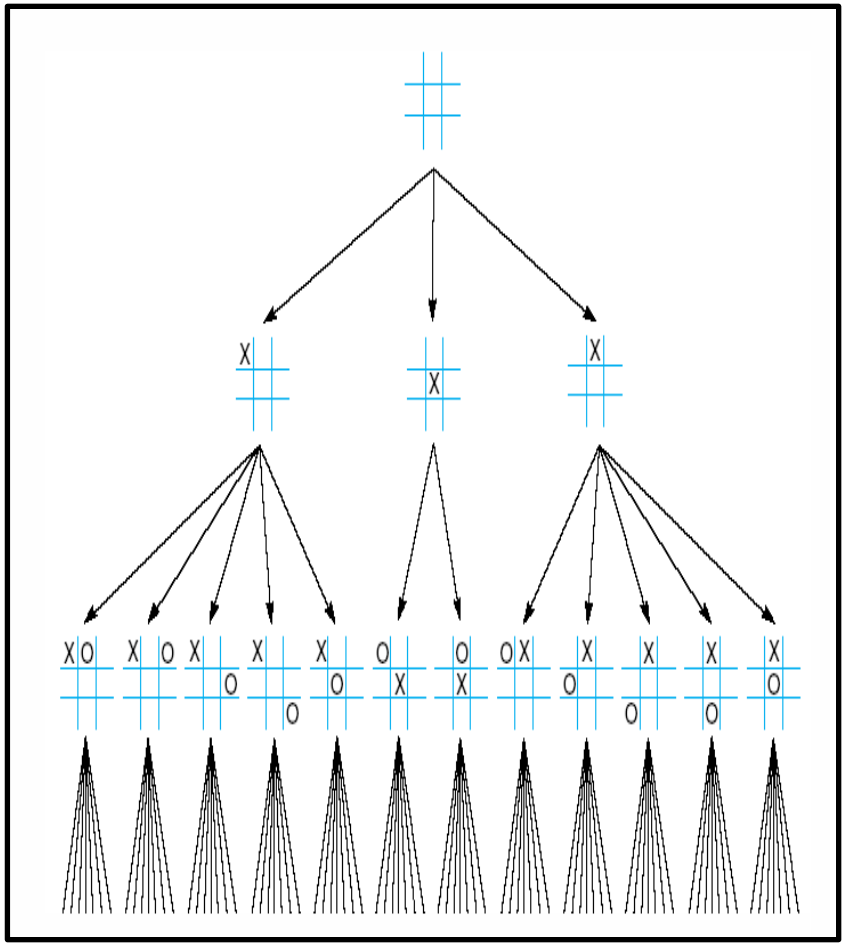


Fig 4.1 First three levels of the tic-tac-toe state space reduced by symmetry

Example (1)

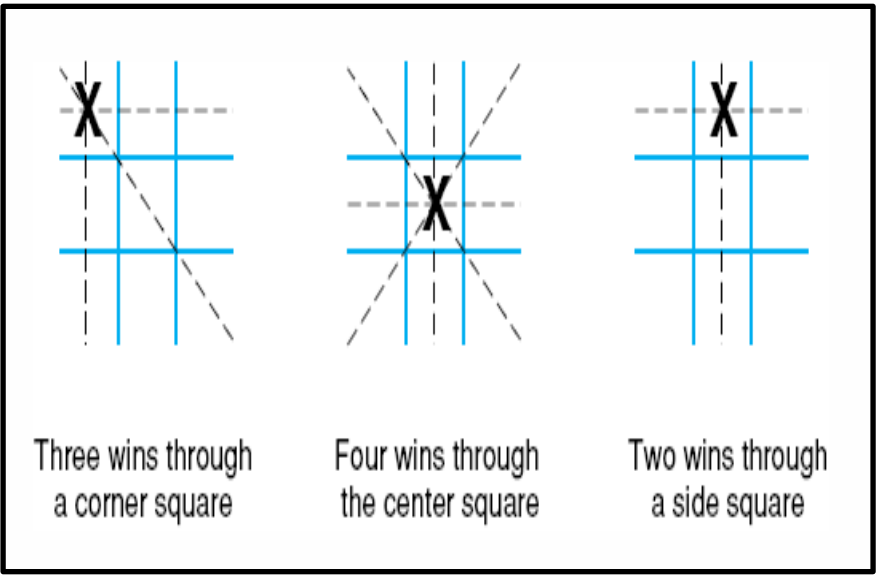


Fig 4.2 The "most wins" heuristic applied to the first children in tic-tac-toe.

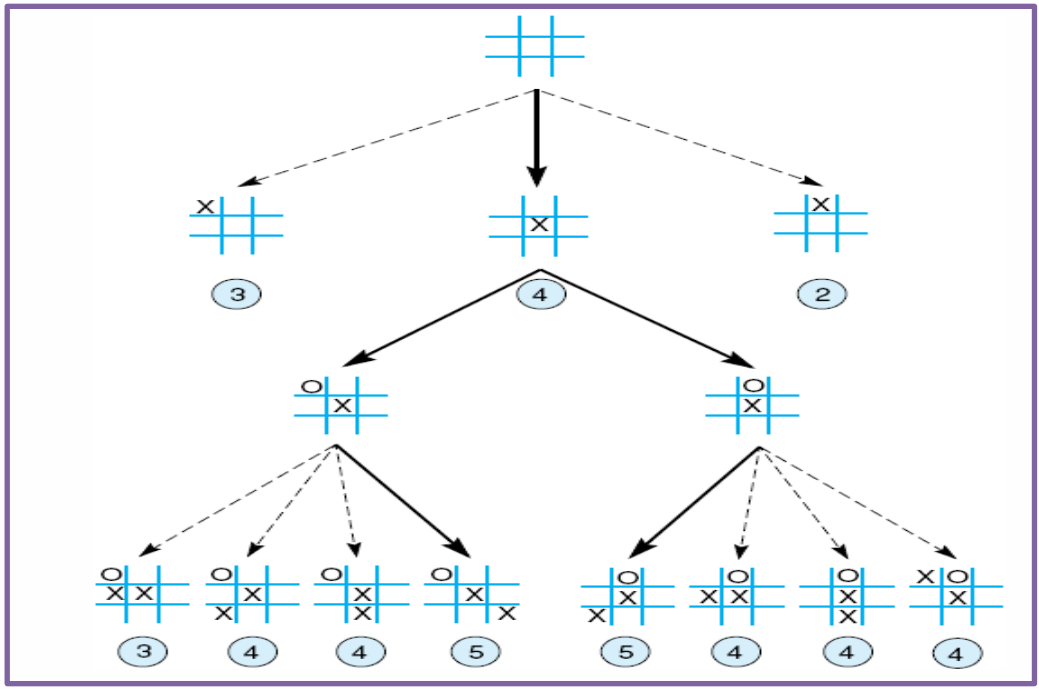
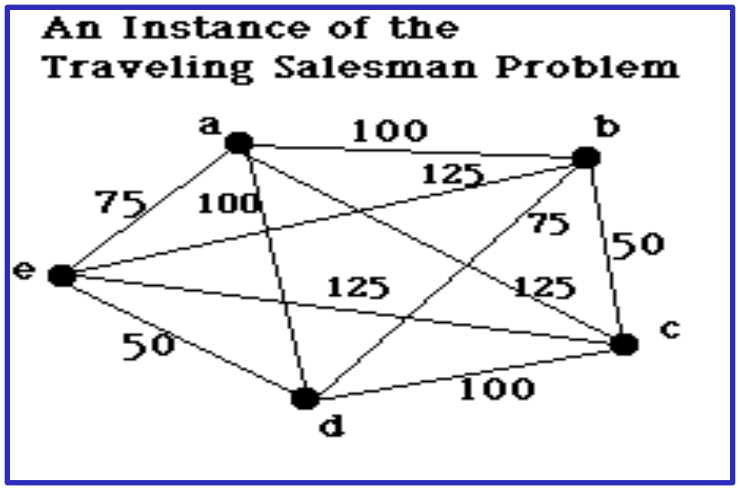


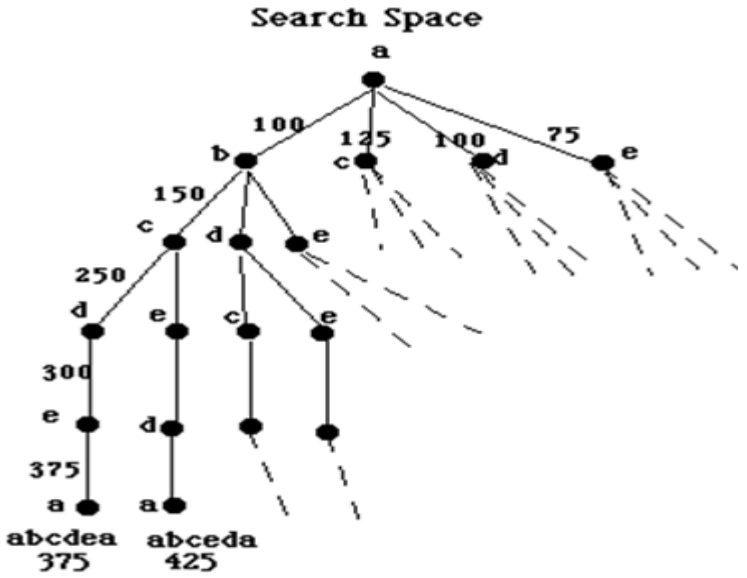
Fig 4.3 Heuristically reduced state space for tic-tac-toe.

- **In reality, the number of states is smaller, as the board fills and reduces options. This crude bound of 25 states is an improvement of four orders of magnitude over 9!.**

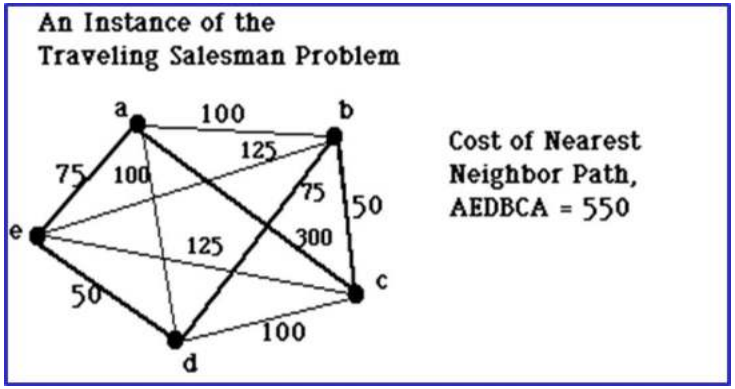
Example (2) The traveling salesman problem

- The traveling salesman problem is too complex to be solved via exhaustive search for large values of N.





- The *nearest neighbor heuristic* work well most of the time, but with some arrangements of cities it does not find the shortest path. Consider the following two graphs.
- In the first, nearest neighbor will find the shortest path. In the second it does not:



Hill Climbing

- The simplest way to implement heuristic search is through a procedure called hill-climbing (Pearl 1984).
- Hill climbing (local search) is an iterative improvement algorithm that starts with an initial state to a problem, then attempts to find the best node for the current node to follow. And so on until goal node is found.
- Hill-climbing strategies expand the current state of the search and evaluate its children.
- The "best" child is selected for further expansion; neither its siblings nor its parent are retained.
- In hill climbing the basic idea is to always head towards a state which is better than the current one.
- So, if you are at town A and you can get to town B and town C (and your target is town D) then you should make a move IF town B or C appear nearer to town D than town A does.

Hill Climbing

Note that:

- **Backtracking is not permitted.**
- **At each step in the search, a single node is chosen to follow.**
- **The criterion for the node to follow is that it's the best state for the current state.**
- **Hill climbing do not care about path from initial state to goal**
- **In steepest ascent hill climbing you will always make your next state the best successor of your current state, and will only make a move if that successor is better than your current state. This can be described as follows:**

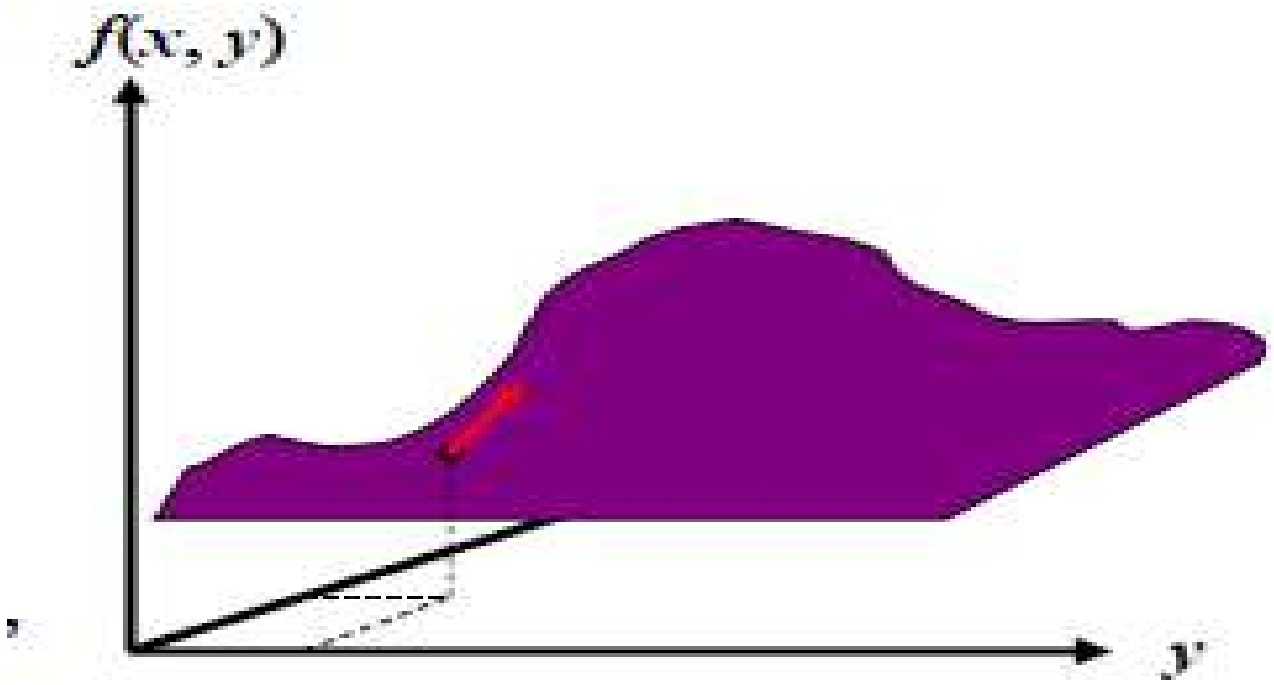
1. Start with current-state = initial-state.

2. Until current-state = goal-state OR there is no change in current-state do:

A. Get the successors of the current state and use the evaluation function to assign a score to each successor.

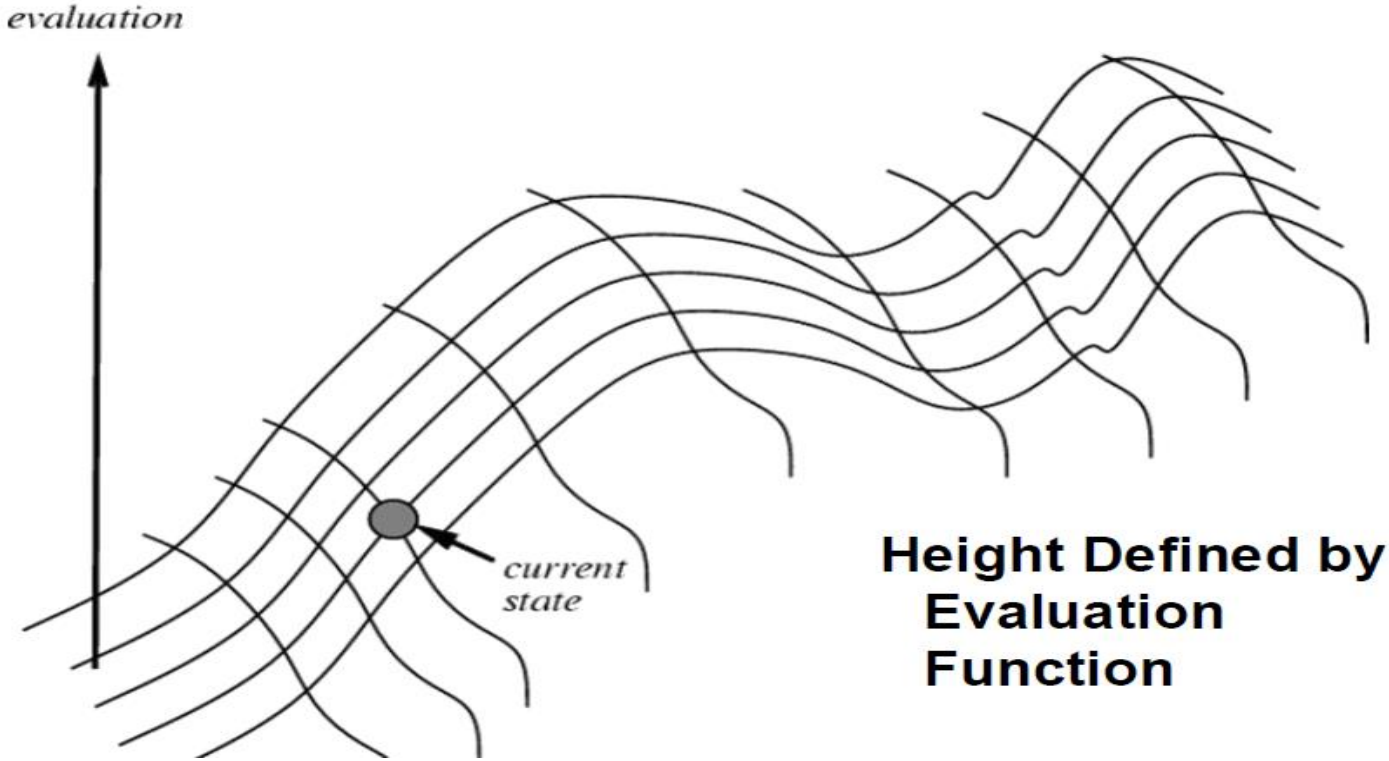
B. 2. If one of the successors has a better score than the current-state then set the new current-state to be the successor with the best score.

Hill Climbing



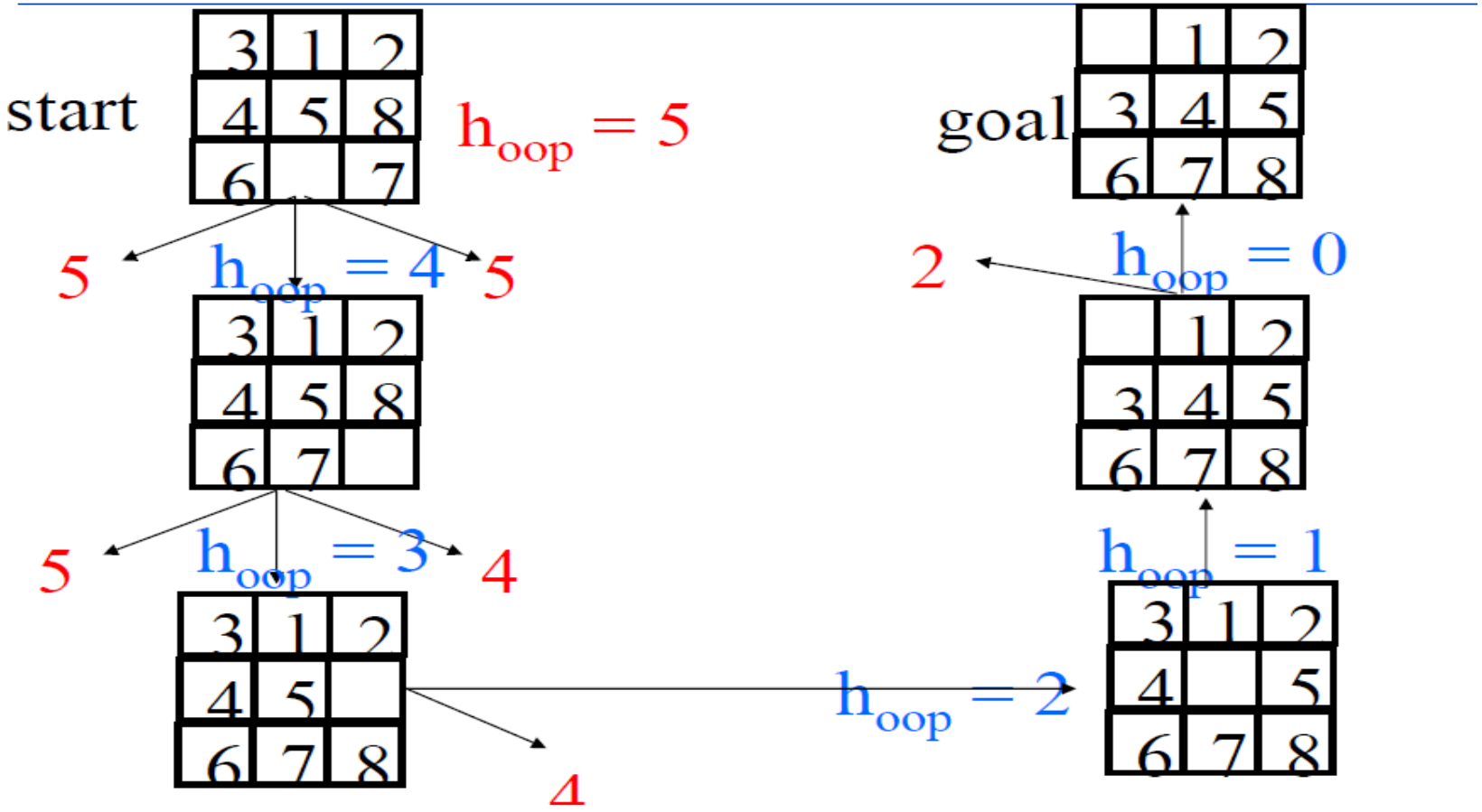
Hill Climbing

Hill climbing on a surface of states



Hill Climbing Example

Hill climbing Example I

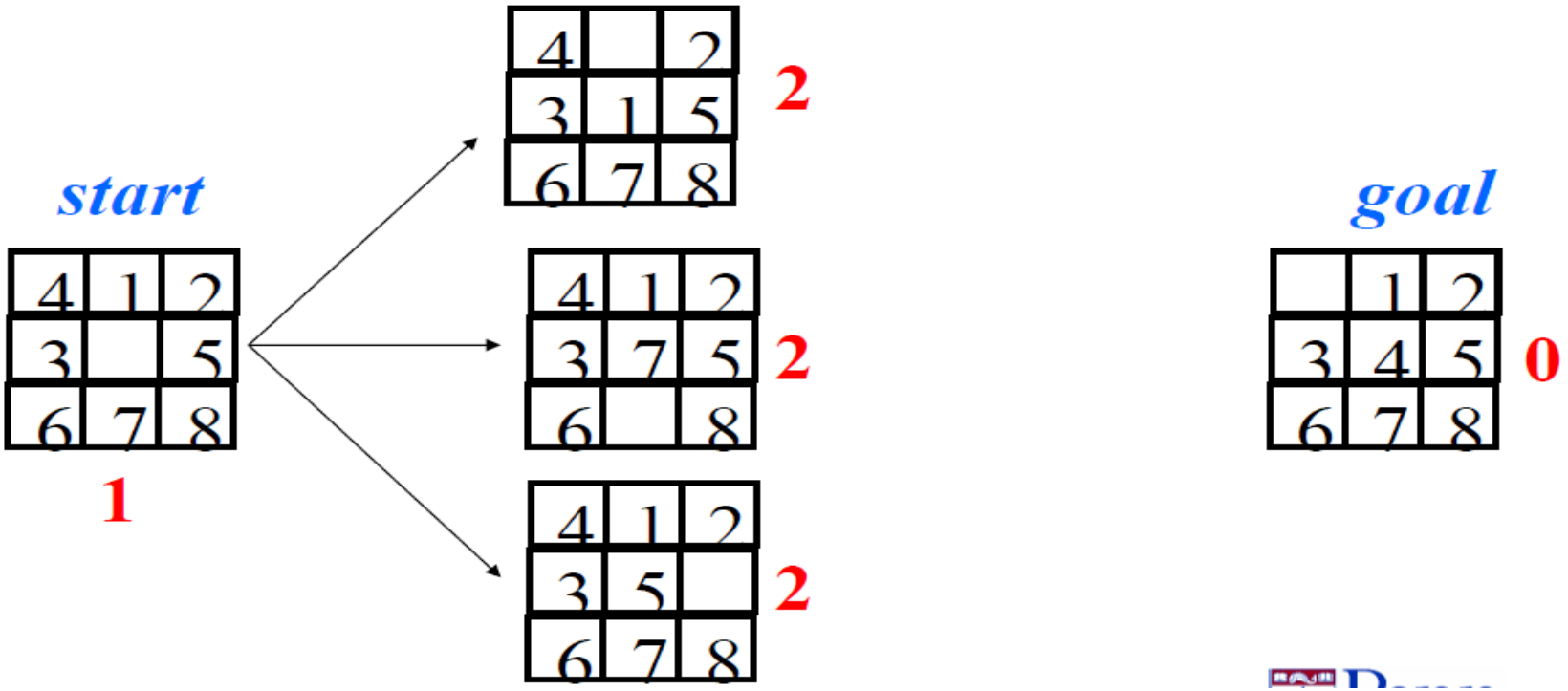


Hill Climbing

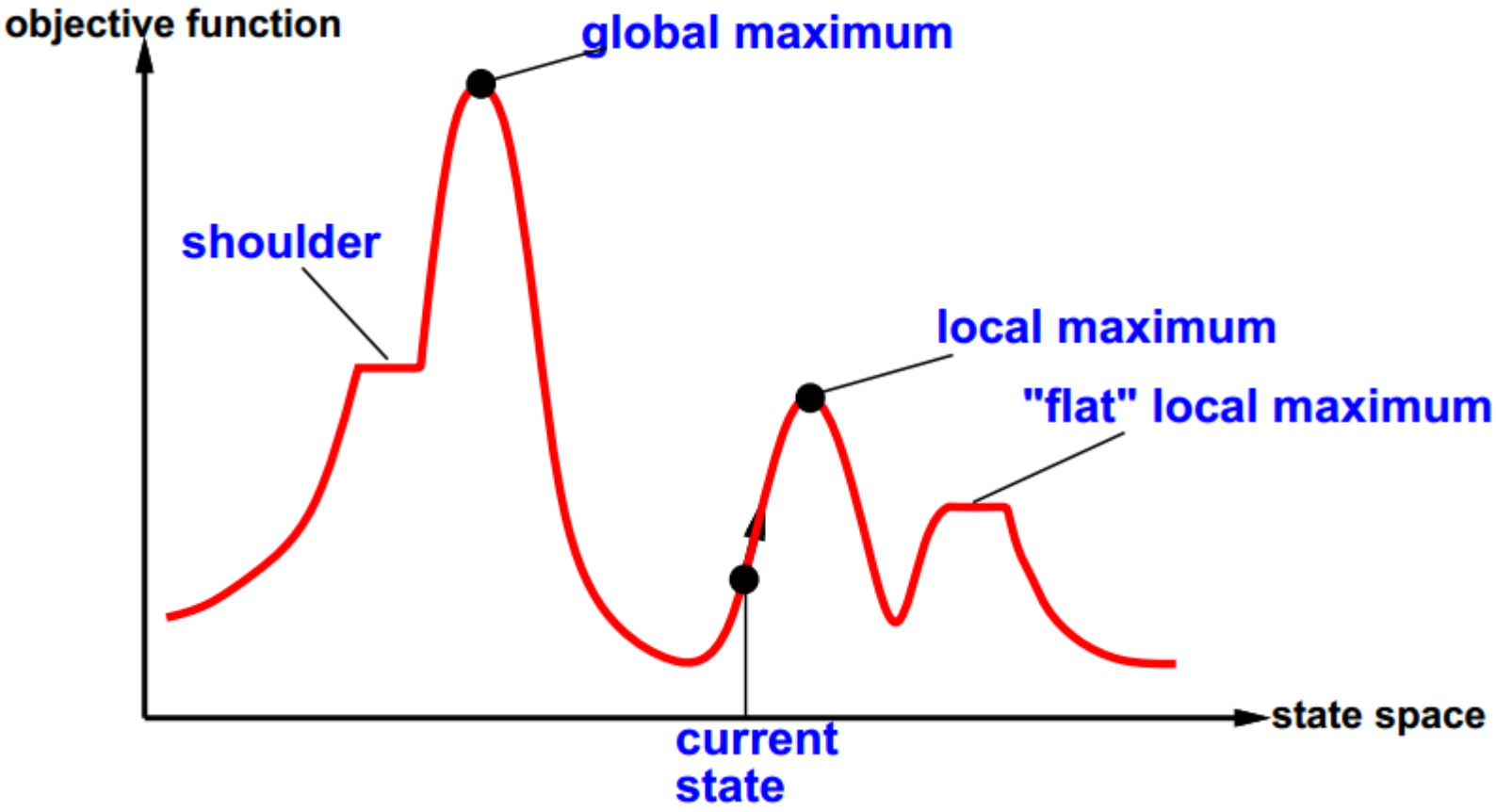
- Terminates when a peak is reached.
- Does not look ahead of the immediate neighbors of the current state.
- Chooses randomly among the set of best successors, if there is more than one.
- Doesn't backtrack, since it doesn't remember where it's been
- The problem with hill climbing is that the best node to enumerate locally may not be the best node globally.
  - For this reason, hill climbing can lead to local optimums, but not necessarily the global optimum (the best solution available).
- Hill-climbing can get stuck without finding optimum
  - Terminates when it reaches a peak (where no neighboring state has a higher value)
  - Does not look beyond immediate neighbors of current state
  - Like climbing Everest in thick fog with amnesia

Hill Climbing Example

Example of a local maximum



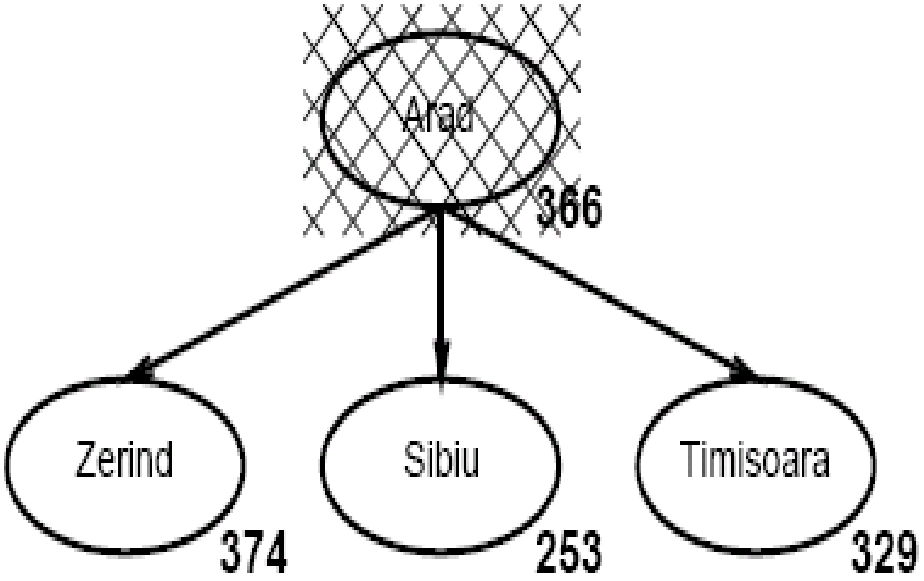
Problem of HILL-CLIMBING SEARCH

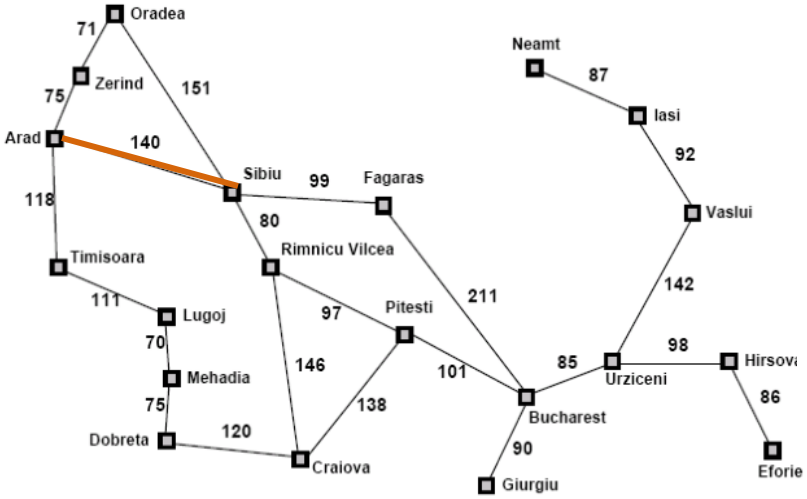


The Best-First Search Algorithm (BEST-FS)

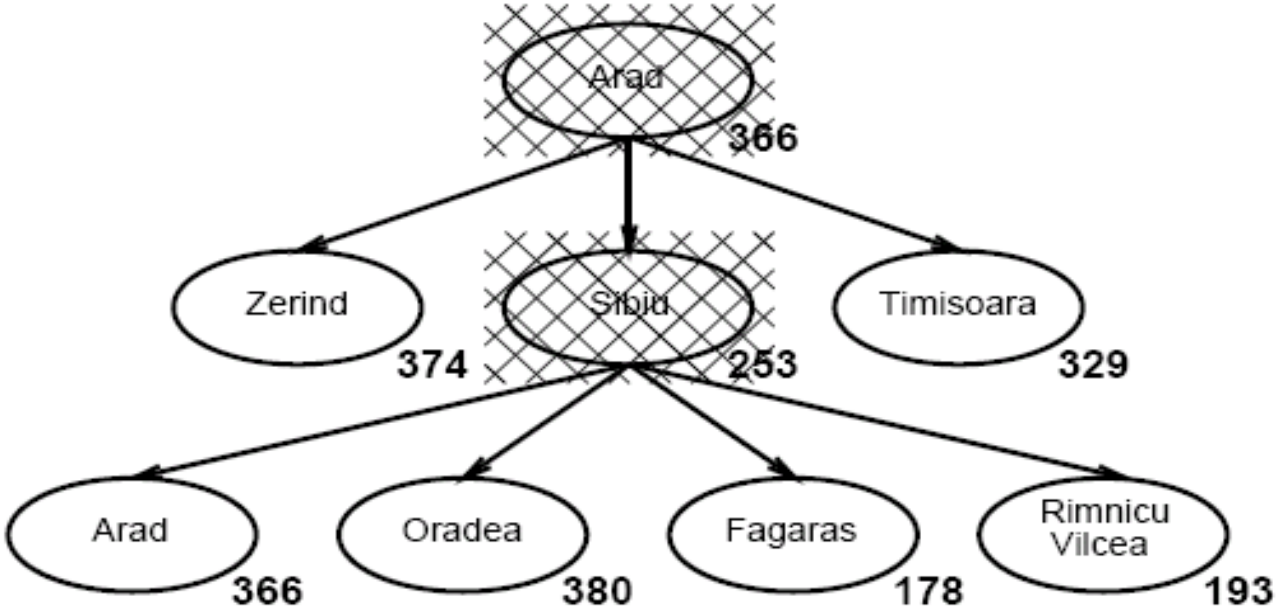
- **Best First Search (or Greedy Best First Search) is a combination of depth first and breadth first searches.**
  - **Depth first is good because a solution can be found without computing all nodes and**
  - **Breadth first is good because it does not get trapped in dead ends.**
- **The best first search allows us to switch between paths thus gaining the benefit of both approaches.**
- **Best First Search described as follows:**
  - **At each step the most promising node is chosen.**
  - **If one of the nodes chosen generates nodes that are less promising it is possible to choose another at the same level and in effect the search changes from depth to breadth.**
  - **If on analysis these are no better then this previously unexpanded node and branch is not forgotten and the search method reverts to the descendants of the first choice and proceeds, backtracking as it were.**

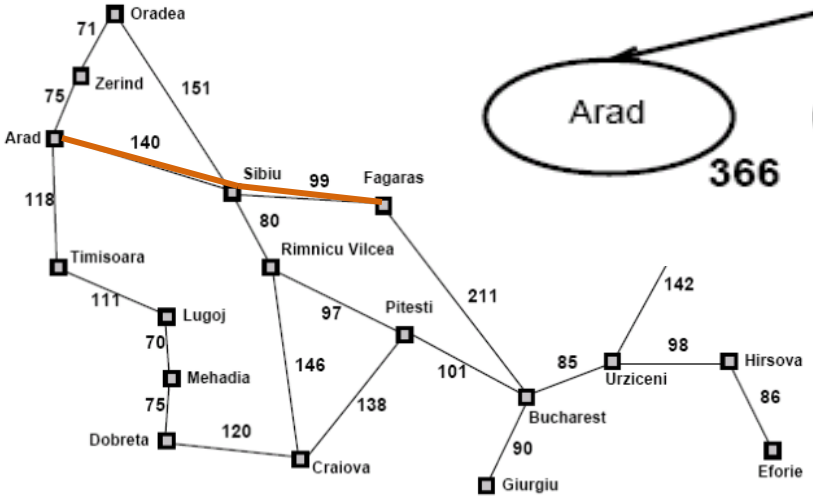
Greedy best-first search



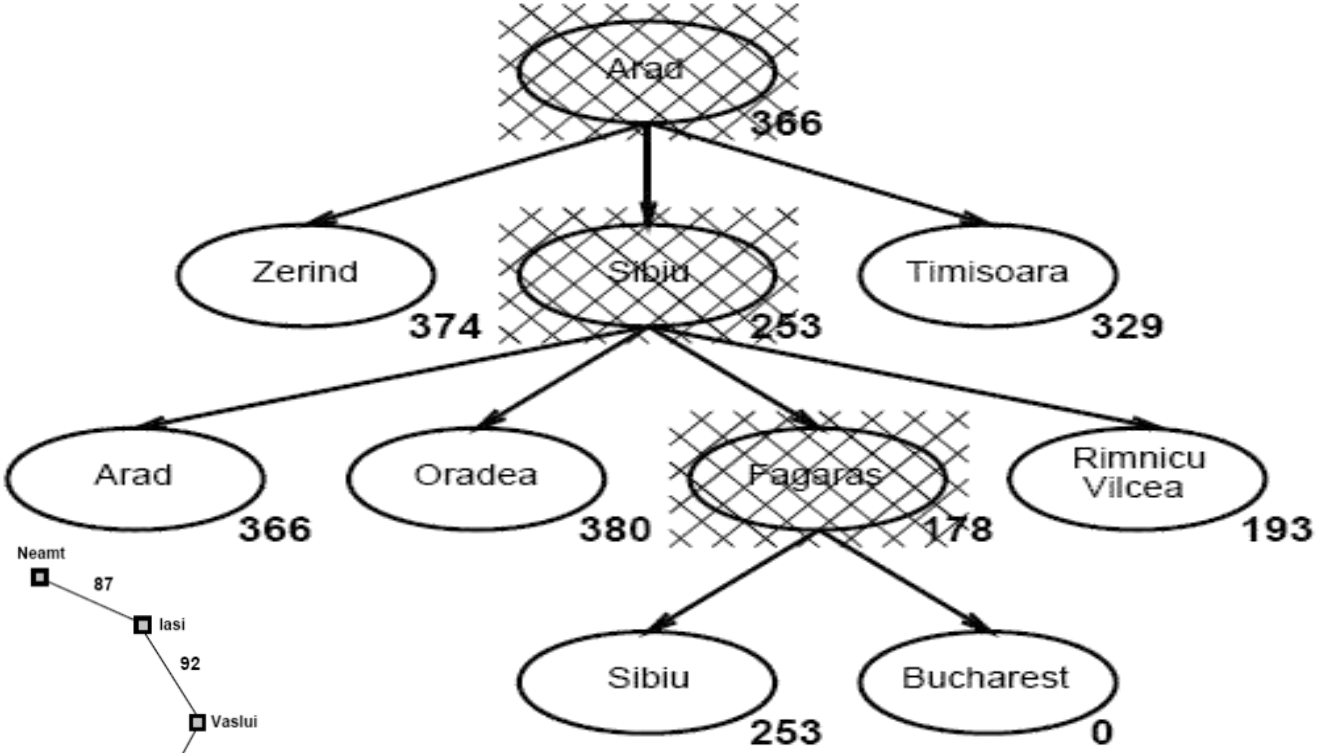


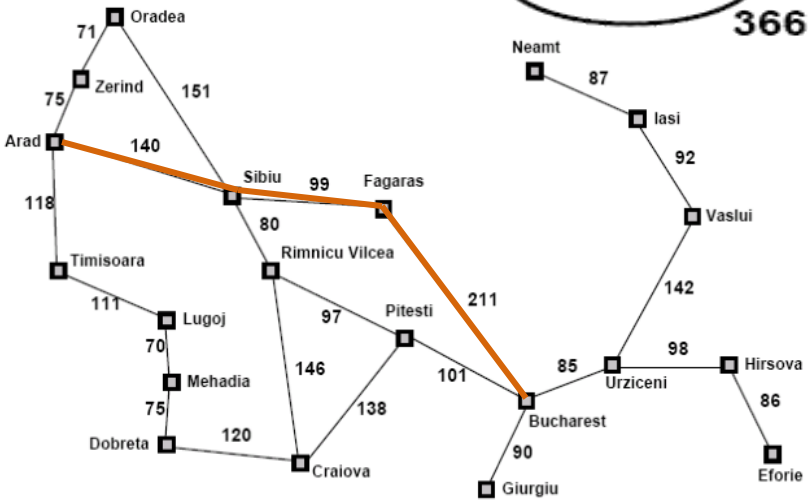
Greedy Best-first search





Greedy Best-first search





The Best-First Search Algorithm(BEST-FS)

Best First Search Algorithm:

1- Start with OPEN holding the initial state

2- Pick the best node on OPEN

3- Generate its successors

4- For each successor Do

If it has not been generated before evaluate it add it to OPEN and record its parent

If it has been generated before change the parent if this new path is better and in that case update the cost of getting to any successor nodes

5- If a goal is found or no more nodes left in OPEN, quit, else return to 2.

```
function best_first_search;

begin
  open := [Start];                                          % initialize
  closed := [ ];
  while open ≠ [ ] do                                       % states remain
    begin
      remove the leftmost state from open, call it X;
      if X = goal then return the path from Start to X
      else begin
            generate children of X;
            for each child of X do
            case
                the child is not on open or closed:
                  begin
                      assign the child a heuristic value;
                      add the child to open
                  end;
                the child is already on open:
                  if the child was reached by a shorter path
                  then give the state on open the shorter path
                the child is already on closed:
                  if the child was reached by a shorter path then
                    begin
                      remove the state from closed;
                      add the child to open
                    end;
            end;                                            % case
            put X on closed;
            re-order states on open by heuristic merit (best leftmost)
          end;
return FAIL                                                 % open is empty
end.
```

The Best-First Search Algorithm(BEST-FS)

- The best first search algorithm will involve an OR graph which avoids the problem of node duplication and assumes that each node has a parent link to give the best node from which it came and a link to all its successors.
- In this way if a better node is found this path can be propagated down to the successors. This method of using an OR graph requires 2 lists of nodes :
  - OPEN is a priority queue of nodes that have been evaluated by the heuristic function but which have not yet been expanded into successors. The most promising nodes are at the front.
  - CLOSED are nodes that have already been generated and these nodes must be stored because a graph is being used in preference to a tree.

BEST-FS – Example 1

If the root is A and the goal is P

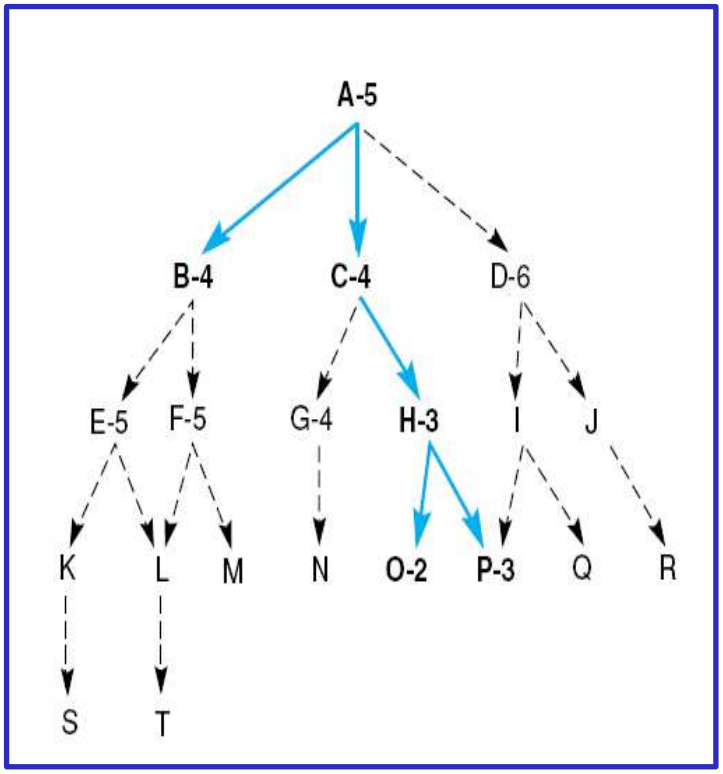


Fig 4.10: Heuristic search of a hypothetical state space.

A trace of the execution of best_first_search for Figure 4.10

1.open = [A5]; closed = [ ]
2.evaluate A5; open = [B4,C4,D6]; closed = [A5]
3.evaluate B4; open = [C4,E5,F5,D6];
closed = [B4,A5]
4.evaluate C4; open = [H3,G4,E5,F5,D6];
closed = [C4,B4,A5]
5.evaluate H3; open = [O2,P3,G4,E5,F5,D6];
closed = [H3,C4,B4,A5]
6.evaluate O2; open = [P3,G4,E5,F5,D6];
closed = [O2,H3,C4,B4,A5]
7.evaluate P3; the solution is found!

BEST-FS – Example 1

- open and closed states at level 3

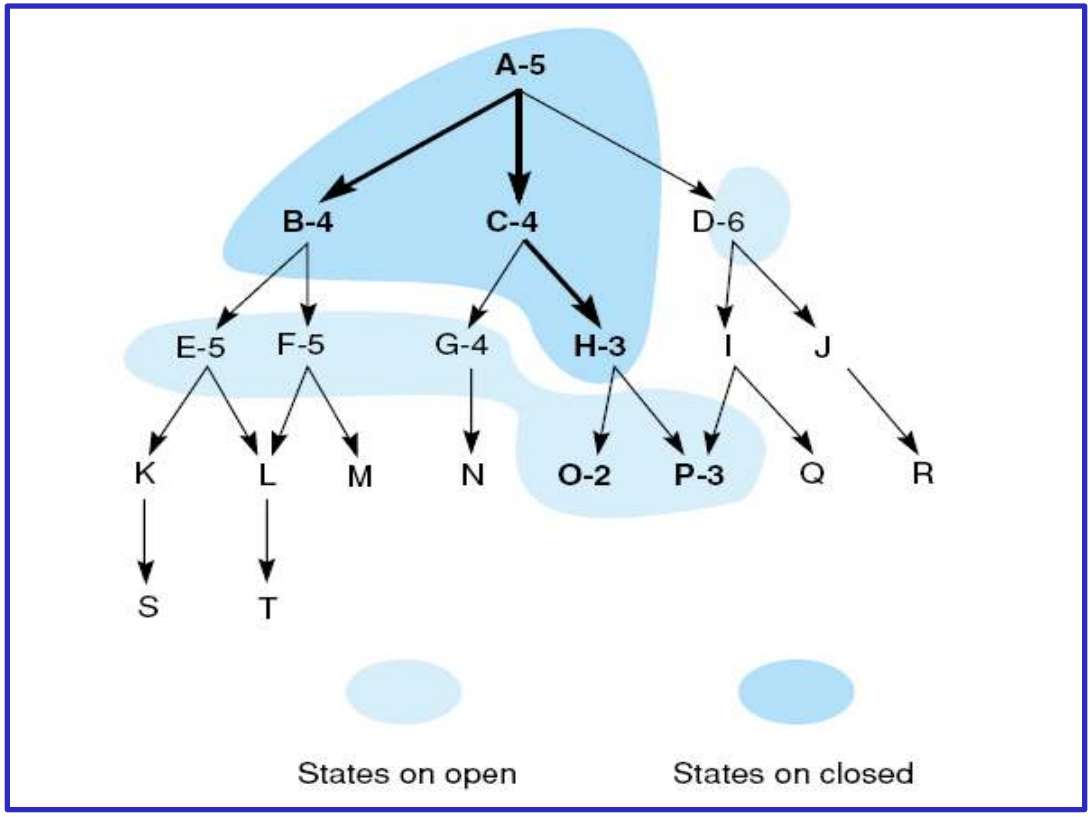
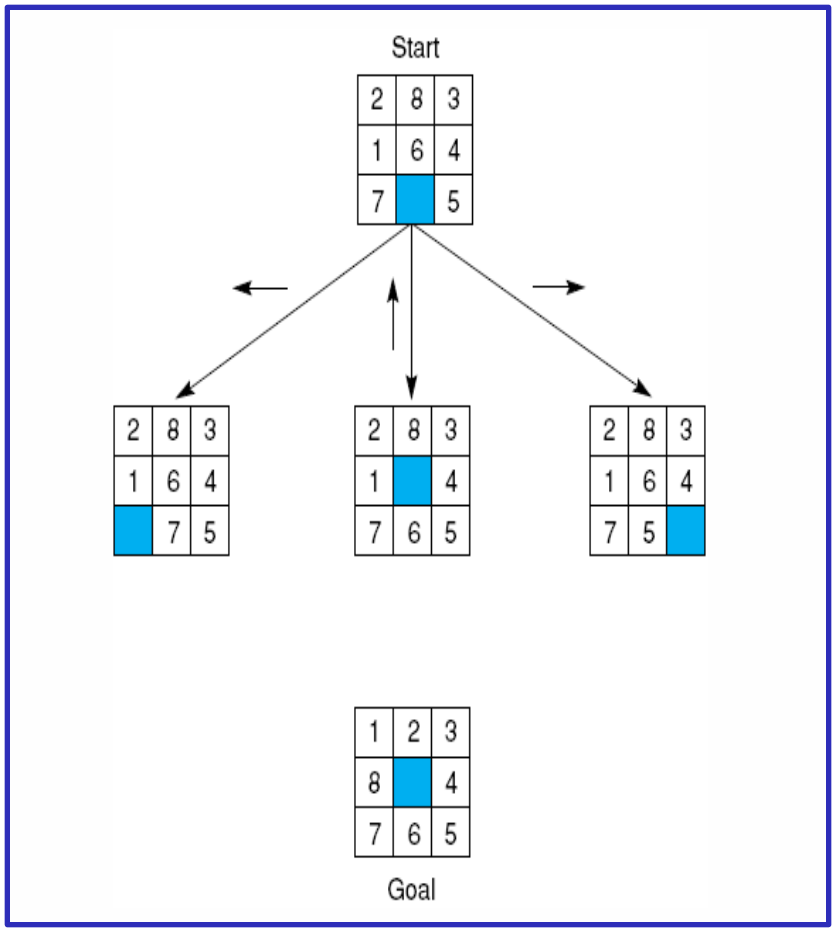


Fig 4.11: Heuristic search of a hypothetical state space with open and closed states highlighted.

BEST-FS – Example 2

- The start state, first moves, and goal state for an example-8 puzzle.
- The simplest heuristic counts the tiles out of place in each state when compared with the goal.
- This heuristic does not use all of the information available in a board configuration, because it does not take into account the distance the tiles must be moved.
- A "better" heuristic would sum all the distances by which the tiles are out of place, one for each square a tile must be moved to reach its position in the goal state.



BEST-FS – Example 2

- Three heuristics applied to states in the 8-puzzle.

2 8 3 1 6 4 _ 7 5	tiles 2,8,1,6,7 **5**	Sum=1+2+1+1+1 **6**	**0**
2 8 3 1 _ 4 7 6 5	tiles 2,8,1 **3**	Sum=1+2+1 **4**	**0**
2 8 3 1 6 4 7 5 _	tiles 2,8,1,6,5 **5**	**6**	**0**
	Tiles out of place	Sum of distances out of place	2 x the number of direct tile reversals

1 2 3
8 _ 4
7 6 5

Goal

The Best-First Search Algorithm

- If two states have the same or nearly the same heuristic evaluations, it is generally preferable to examine the state that is nearest to the root state of the graph.
- This state will have a greater probability of being on the shortest path to the goal.
- The distance from the starting state to its descendants can be measured by maintaining a depth count for each state.
- This count is 0 for the beginning state and is incremented by 1 for each level of the search.
- This makes our evaluation function, f , the sum of two components:

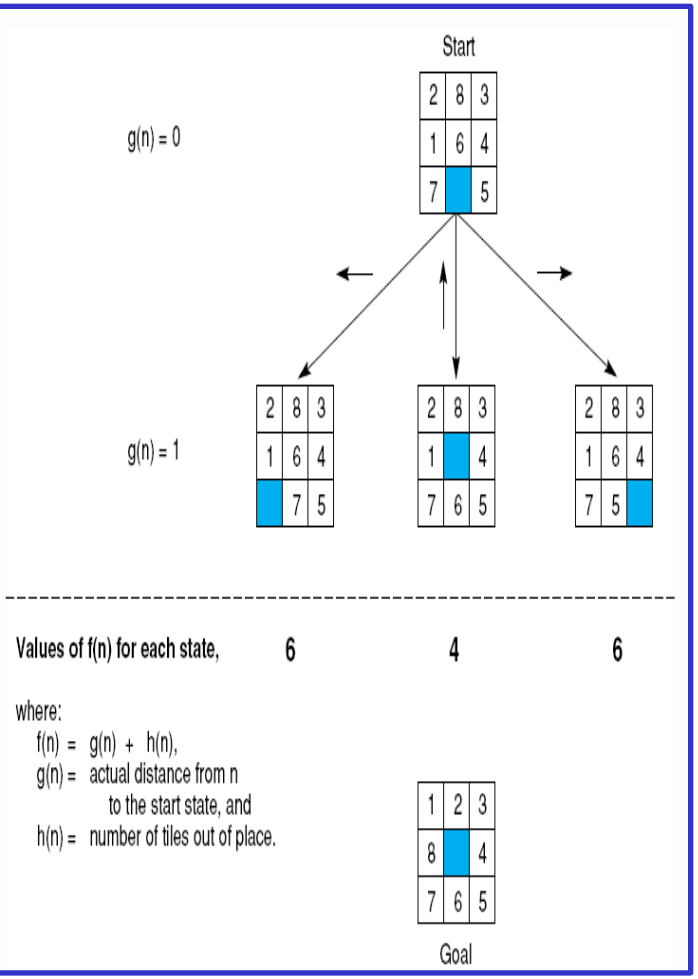
$$f(n) = g(n) + h(n)$$

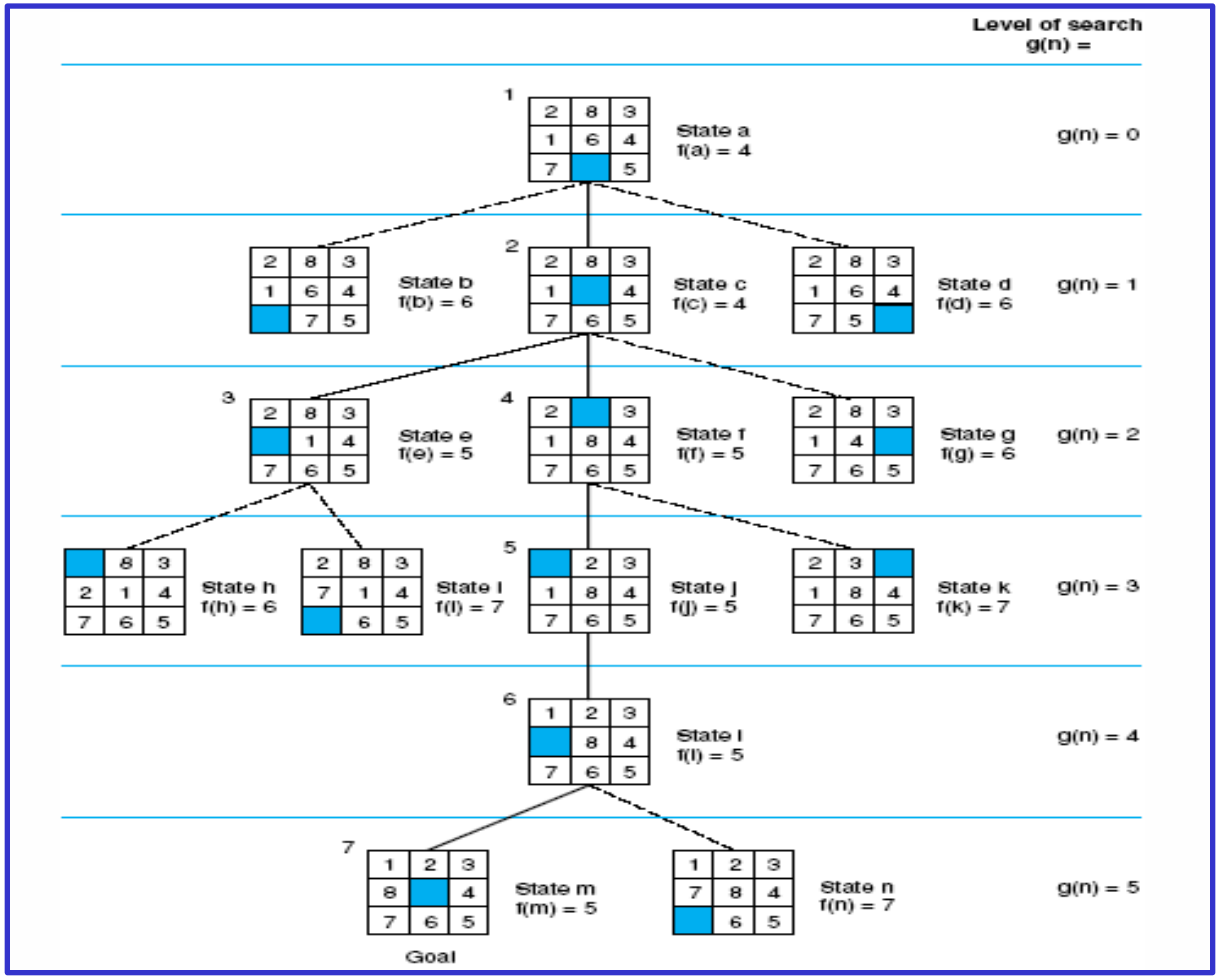
Where

- $g(n)$ measures the actual length of the path from any state n to the start state and,
- $h(n)$ is a heuristic estimate of the distance from state n to a goal.

BEST-FS – Example 3

- The heuristic f applied to states in the 8-puzzle.





BEST-FS – Example 3

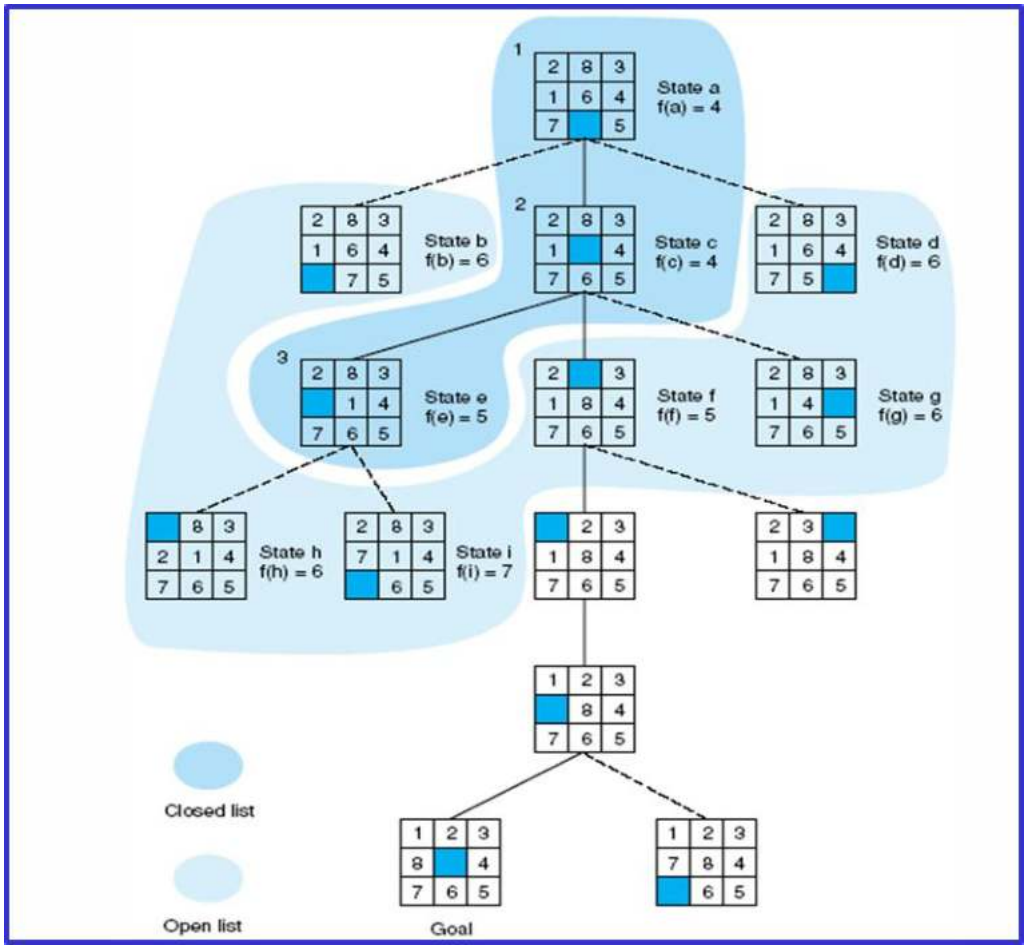
- The full best-first search of the 8-puzzle graph.
- Each state is labeled with a letter and its heuristic weight,

  f(n) = g(n) + h(n)
- The successive stages of open and closed that generate this graph are:

1. open = [a4];
   closed = [ ]
2. open = [c4, b6, d6];
   closed = [a4]
3. open = [e5, f5, b6, d6, g6];
   closed = [a4, c4]
4. open = [f5, h6, b6, d6, g6, i7];
   closed = [a4, c4, e5]
5. open = [j5, h6, b6, d6, g6, k7, i7];
   closed = [a4, c4, e5, f5]
6. open = [l5, h6, b6, d6, g6, k7, i7];
   closed = [a4, c4, e5, f5, j5]
7. open = [m5, h6, b6, d6, g6, n7, k7, i7];
   closed = [a4, c4, e5, f5, j5, l5]
8. success, m = goal!

BEST-FS – Example 3

- Open and closed as they appear after the 3rd iteration of heuristic search



The Best-First Search Algorithm Summarization

To summarize:

- Operations on states generate children of the state currently under examination.
- Each new state is checked to see whether it has occurred before (is on either open or closed), there by preventing loops.
- Each state n is given an f value equal to the sum of its depth in the search space g(n) and a heuristic estimate of its distance to a goal h(n). The h value guides search toward heuristically promising states while the g value can prevent search from persisting indefinitely on a fruitless path.
- States on open are sorted by their f values. By keeping all states on open until they are examined or a goal is found, the algorithm recovers from dead ends.
- As an implementation point, the algorithm's efficiency can be improved through maintenance of the open and closed lists, perhaps as heaps or leftist trees.

BEST-FS Properties

How does best-first search rate according to the four criteria of performance?

- **Complete:**
  - It is complete
    - It always reaches goal.
- **Optimal:**
  - It is not optimal
    - A solution can be found in a longer path (higher *g(n)* with a lower *h(n)* value)
- **Time:**
  - It generate $O(b^m)$ nodes.
    - where b is maximum forward branching factor and m is maximum path length
- **Space:**
  - Space complexity is $O(b^m)$
    - Keeps every node in memory.

Admissibility, Monotonicity , and Informedness

- We may evaluate the behavior of heuristics along a number of dimensions.
- We may desire a solution and also require the algorithm to find the shortest path to the goal.
- For instance, this could be important when an application might have an excessive cost for extra solution steps, such as planning a path for an autonomous robot through a dangerous environment.
- Heuristics that find the shortest path to a goal whenever it exists are said to be admissible.

Admissibility Measures

- A search algorithm is *Admissible* if it is guaranteed to find a minimal path to a solution whenever such a path exists.

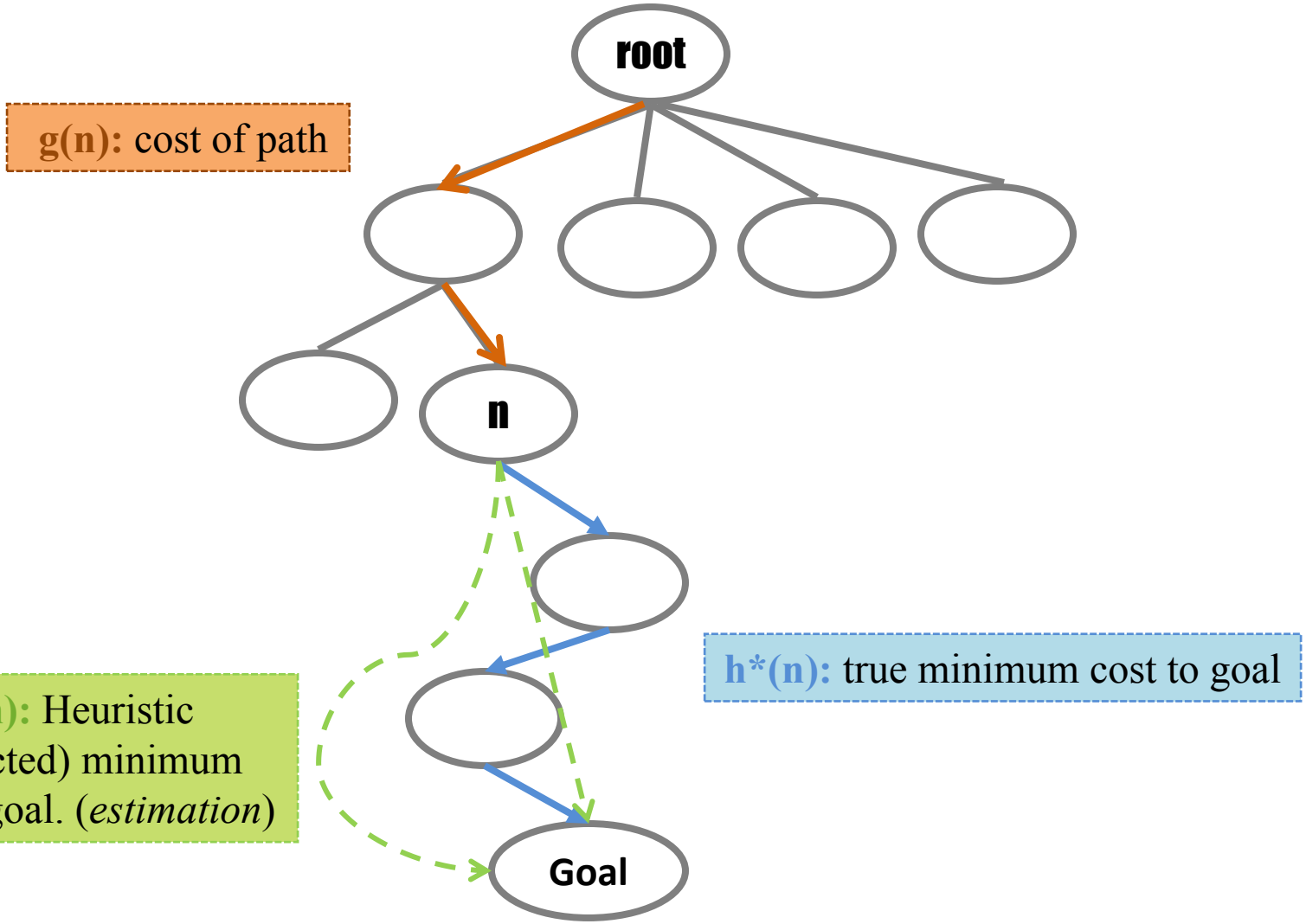
- Breadth-first search is an *admissible* search strategy. Because it looks at every state at level *n* of the graph before considering any state at the level n+1, any goal nodes are found along the shortest possible path.

- Unfortunately, breadth-first search is often too inefficient for practical use.

A* algorithm

- Best-First Search Algorithm A is called as A* algorithm
- Using the evaluation function f(n) = g(n) + h(n)
- If n is a node in the state space graph,
  - g(n) measures the depth at which that state has been found in the graph, and
  - h(n) is the heuristic estimate of the distance from n to a goal.
- In this sense f(n) estimates the total cost of the path from the start state through n to the goal state.

A* search


root
g(n): cost of path
n
h*(n): true minimum cost to goal
h(n): Heuristic (expected) minimum cost to goal. (*estimation*)
Goal

A* algorithm

- **A* algorithm:**

  1- Start with OPEN holding the initial state

  2- Pick the best node on OPEN

  3- Generate its successors

  4- For each successor Do

  If it has not been generated before evaluate it add it to OPEN and record its parent

  If it has been generated before change the parent if this new path is better and in that case update the cost of getting to any successor nodes

  5- If a goal is found or no more nodes left in OPEN, quit, else return to 2.

```
function best_first_search;

begin
   open := [Start];                                          % initialize
   closed := [ ];
   while open ≠ [ ] do                                       % states remain
      begin
         remove the leftmost state from open, call it X;
         if X = goal then return the path from Start to X
         else begin
                  generate children of X;
                  for each child of X do
                  case
                       the child is not on open or closed:
                         begin
                              assign the child a heuristic value;
                              add the child to open
                         end;
                       the child is already on open:
                         if the child was reached by a shorter path
                         then give the state on open the shorter path
                       the child is already on closed:
                         if the child was reached by a shorter path then
                           begin
                              remove the state from closed;
                              add the child to open
                           end;
                  end;                                       % case
                  put X on closed;
                  re-order states on open by heuristic merit (best leftmost)
               end;
return FAIL                                                  % open is empty
end.
```

A* algorithm

- We may characterize a class of *Admissible Heuristic* search strategies. we define an evaluation function f*:

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

Where

- g*(n) is the cost of the *shortest* path from the start to node n and,
- h*(n) returns the *actual* cost of the shortest path from n to the goal.

It follows that

f*(n) is the actual cost of the optimal path from a start node to a goal node that passes through node n.

A* algorithm

- In algorithm A, g(n), the cost of the current path to state n , is a reasonable estimate of g*, but they may not be equal: g(n)≥g*(n).
- These are equal only if the graph search has discovered the optimal path to state n.
- Similarly, we replace h*(n) with h(n), a heuristic estimate of the minimal cost to a goal state.
- Although we usually may not compute h*, it is often possible to determine whether or not the heuristic estimate, h(n), is bounded from above by h*(n), i.e., is always less than or equal to the actual cost of a minimal path.
- If algorithm A uses an evaluation function *f* in which h(n) ≤ h*(n), it is called *algorithm A**.

ALGORITHM A, ADMISSIBILITY, ALGORITHM A*

Consider the evaluation function f(n) = g(n) + h(n), where

n is any state encountered in the search.

g(n) is the cost of n from the start state.

h(n) is the heuristic estimate of the cost of going from n to a goal.

If this evaluation function is used with the best_first_search algorithm of Section 4.1, the result is called *algorithm A*.

A search algorithm is *admissible* if, for any graph, it always terminates in the optimal solution path whenever a path from the start to a goal state exists.

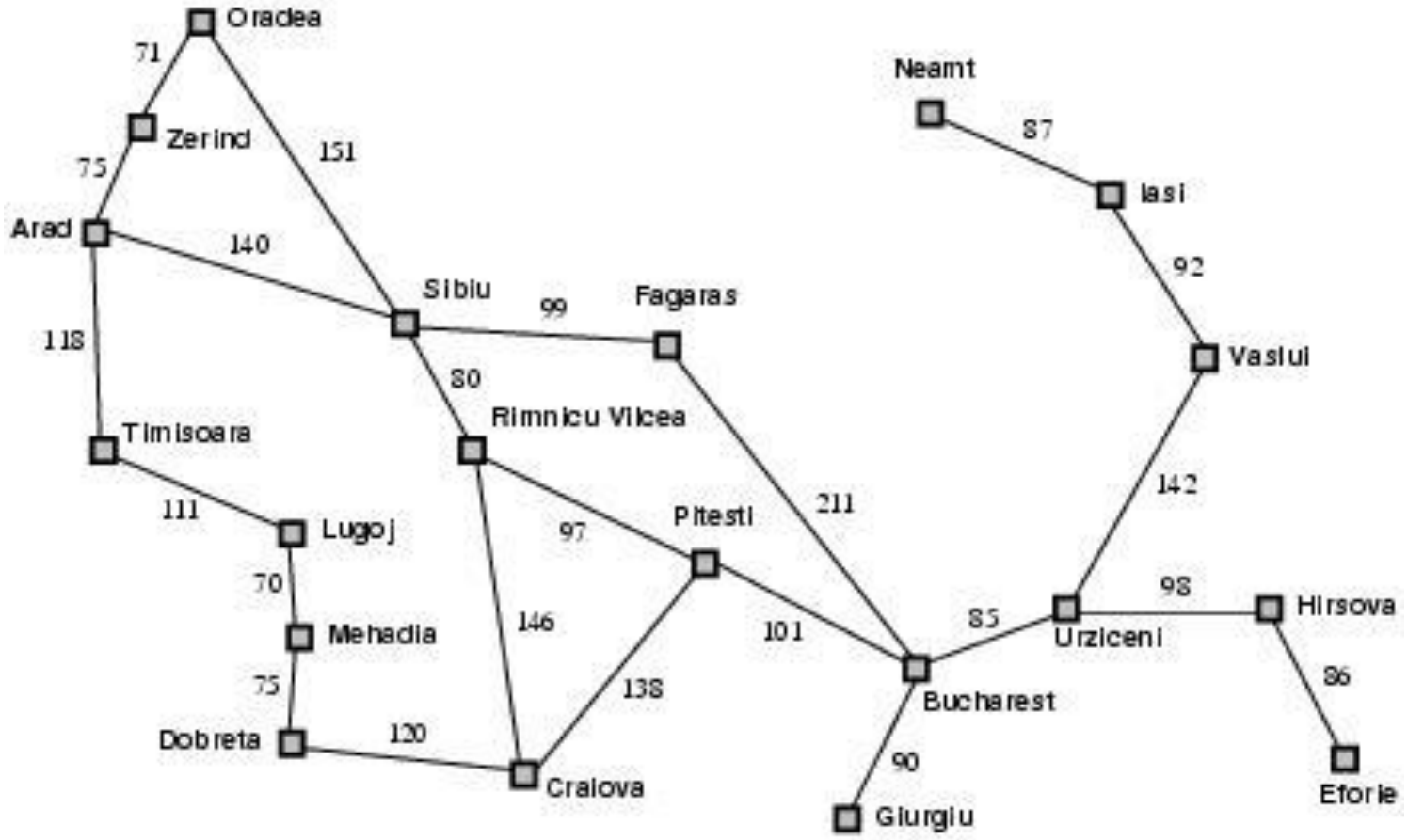
If algorithm A is used with an evaluation function in which h(n) is less than or equal to the cost of the minimal path from n to the goal, the resulting search algorithm is called *algorithm* A* (pronounced "A STAR").

It is now possible to state a property of A* algorithms:

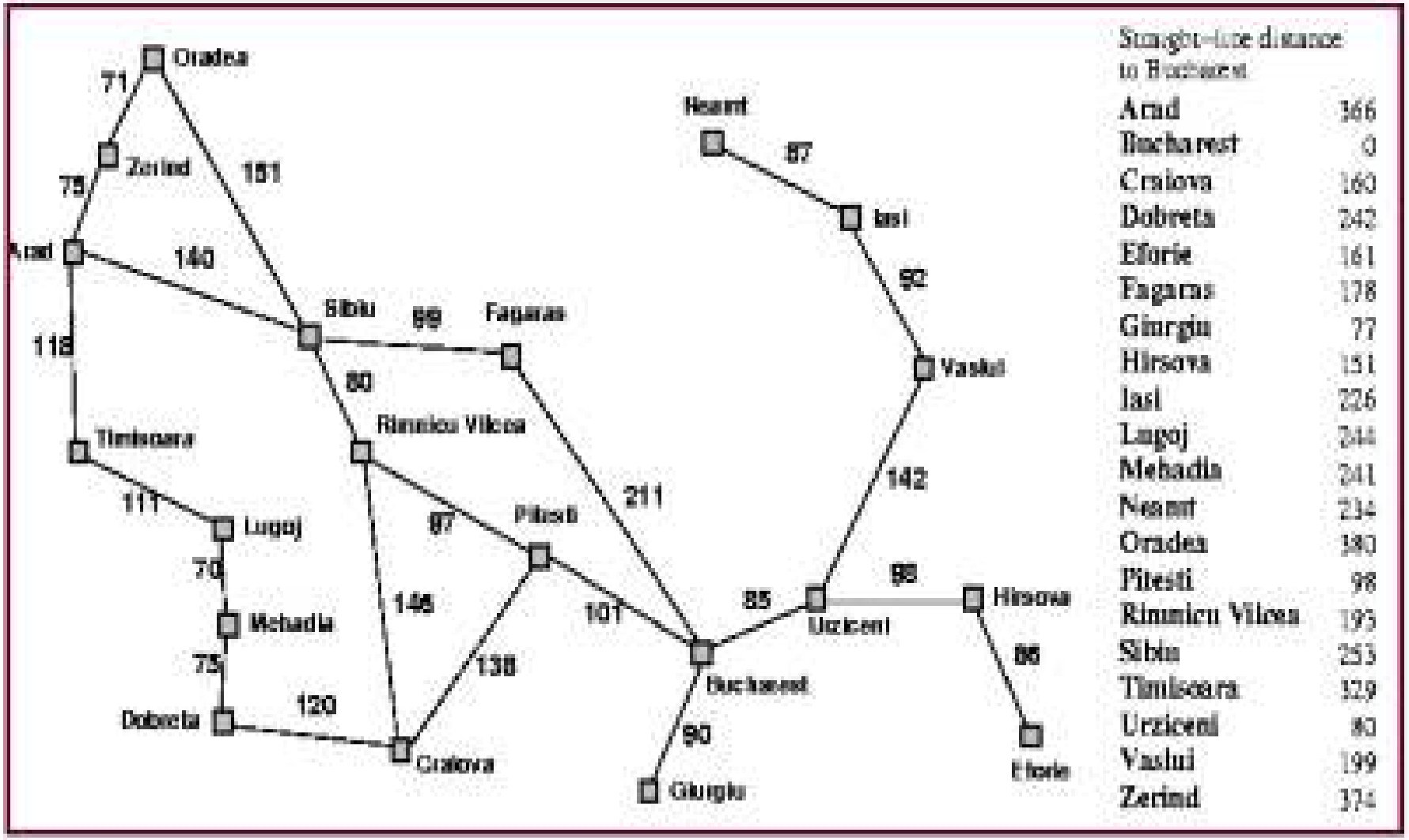
All A* algorithms are admissible.

A* algorithm - Romania Example

- Find Bucharest starting at Arad



A* algorithm - Romania Example



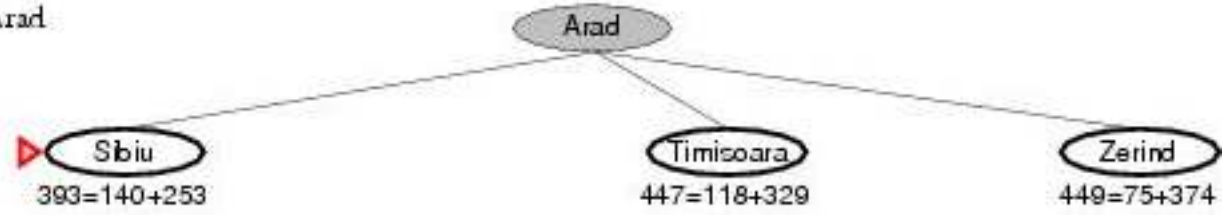
Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* algorithm - Romania Example

(a) The initial state

f(Arad) = c(??,Arad)+h(Arad)=0+366=366

After expanding Arad

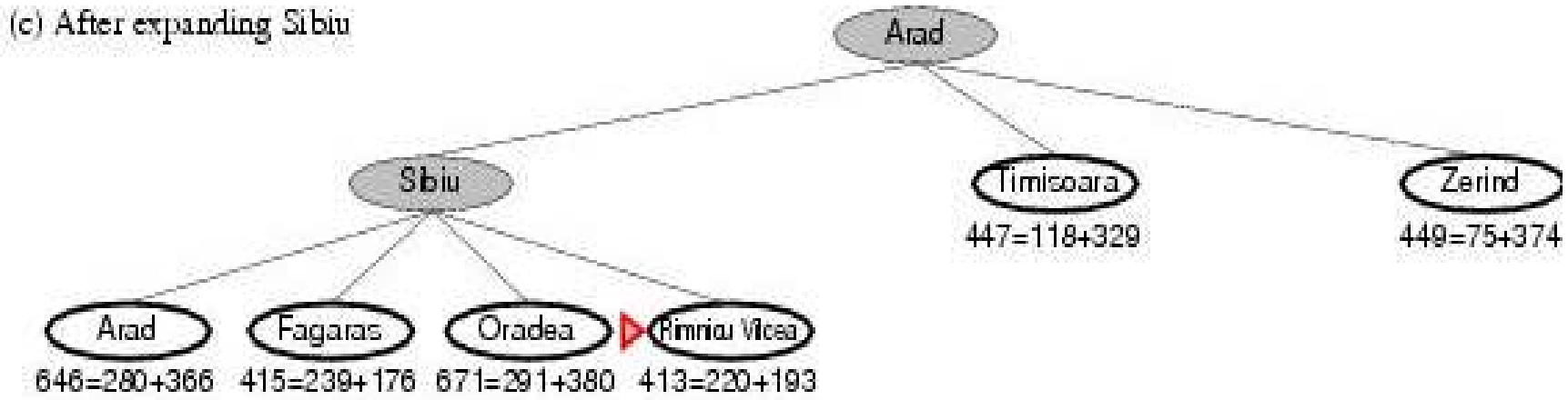


Expand Arad and determine f(n) for each node

- **f(Sibiu)=c(Arad,Sibiu)+h(Sibiu)=140+253=393**
- **f(Timisoara)=c(Arad,Timisoara)+h(Timisoara)=118+329=447**
- **f(Zerind)=c(Arad,Zerind)+h(Zerind)=75+374=449**

Best choice is Sibiu

A* algorithm - Romania Example

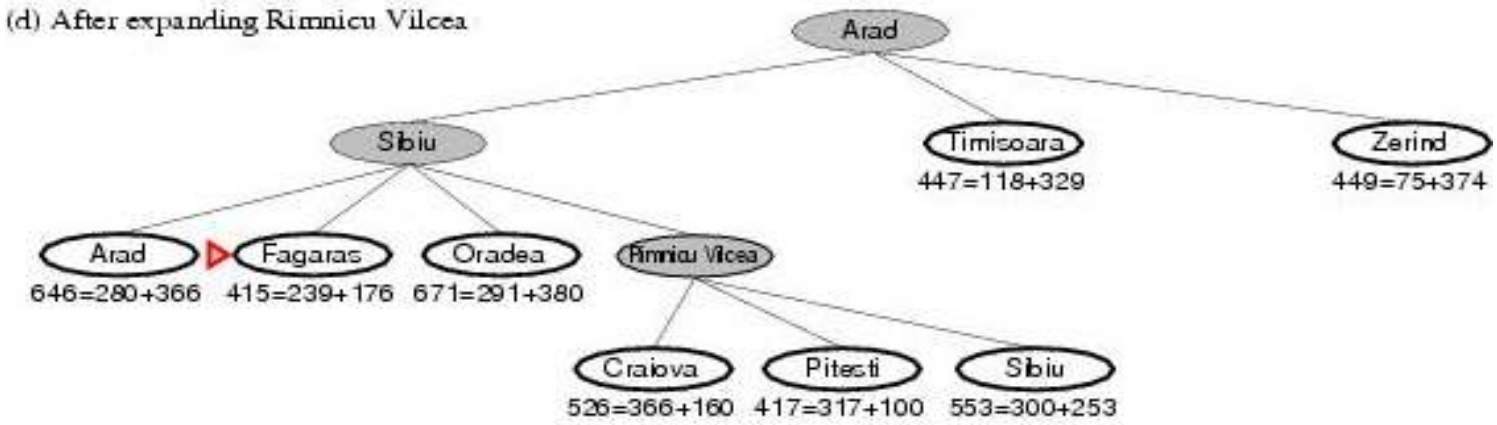


Expand Sibiu and determine f(n) for each node

- **f(Arad)=c(Sibiu,Arad)+h(Arad)=280+366=646**
- **f(Fagaras)=c(Sibiu,Fagaras)+h(Fagaras)=239+179=415**
- **f(Oradea)=c(Sibiu,Oradea)+h(Oradea)=291+380=671**
- **f(Rimnicu Vilcea)=c(Sibiu,Rimnicu Vilcea)+h(Rimnicu Vilcea)=220+192=413**

Best choice is Rimnicu Vilcea

A* algorithm - Romania Example

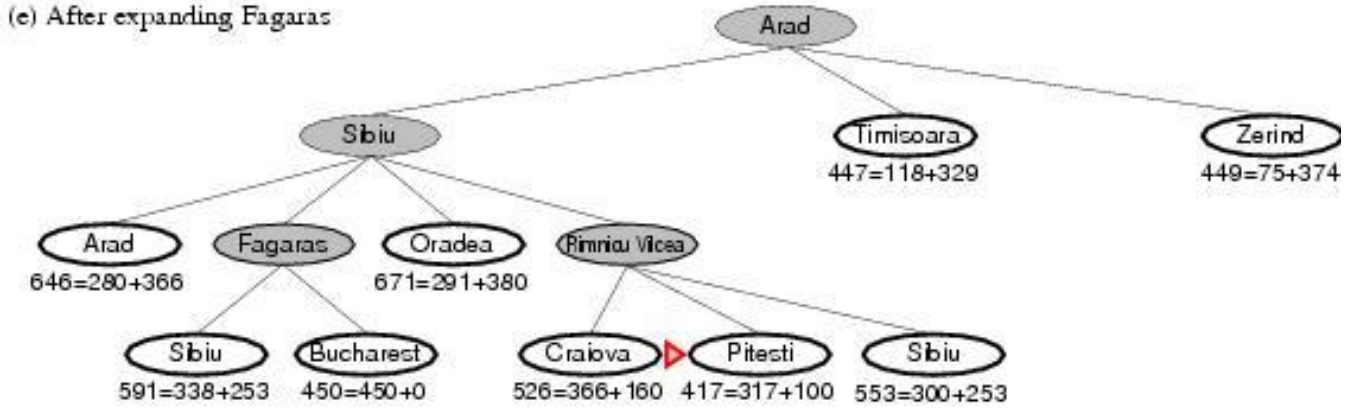


Expand Rimnicu Vilcea and determine f(n) for each node

- **f(Craiova)=c(Rimnicu Vilcea, Craiova)+h(Craiova)=360+160=526**
- **f(Pitesti)=c(Rimnicu Vilcea, Pitesti)+h(Pitesti)=317+100=417**
- **f(Sibiu)=c(Rimnicu Vilcea,Sibiu)+h(Sibiu)=300+253=553**

Best choice is Fagaras

A* algorithm - Romania Example

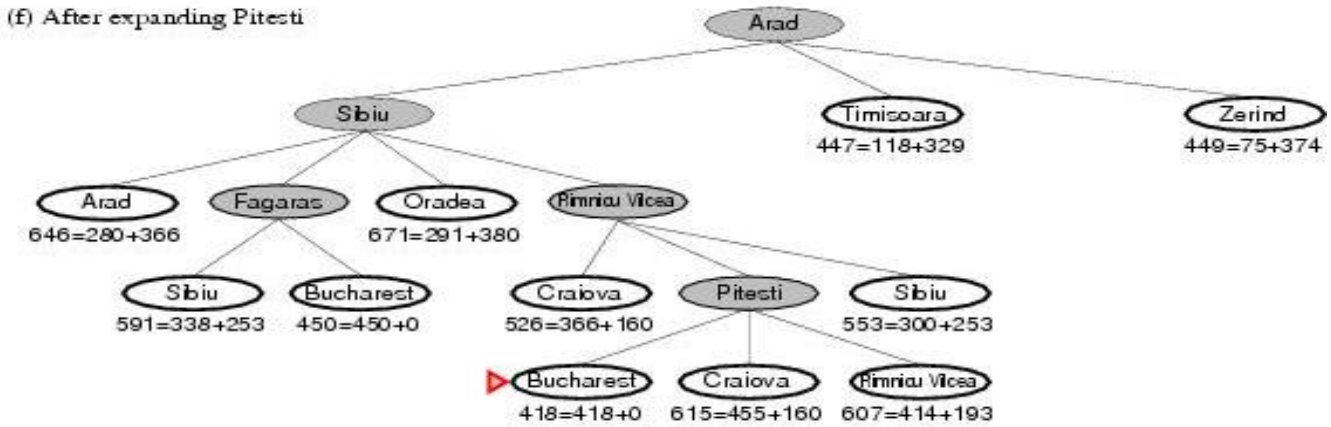


Expand Fagaras and determine f(n) for each node

- **f(Sibiu)=c(Fagaras, Sibiu)+h(Sibiu)=338+253=591**
- **f(Bucharest)=c(Fagaras,Bucharest)+h(Bucharest)=450+0=450**

Best choice is Pitesti !!!

A* algorithm - Romania Example



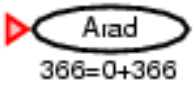
Expand Pitesti and determine f(n) for each node

- **f(Bucharest)=c(Pitesti,Bucharest)+h(Bucharest)=418+0=418**

Best choice is Bucharest !!!

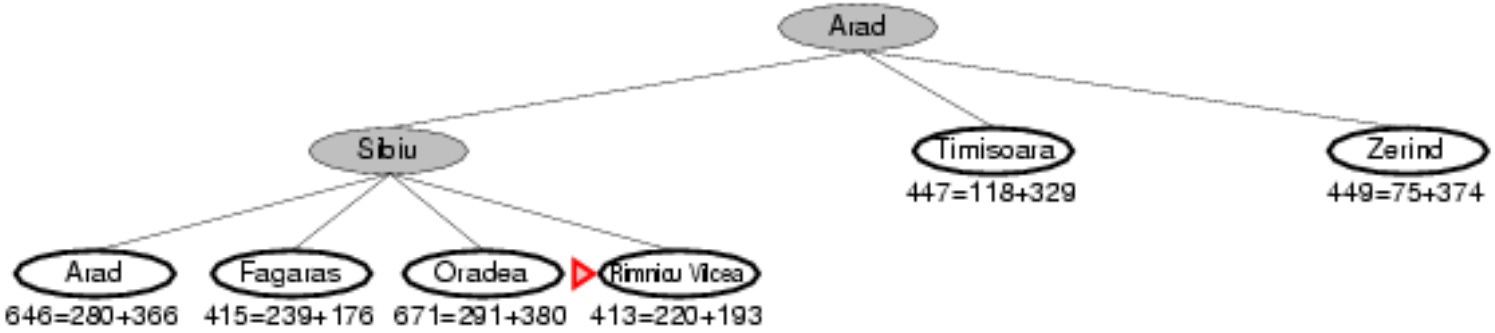
- **Optimal solution (only if h(n) is admissable)**
- **Note values along optimal path !!**

A* search example

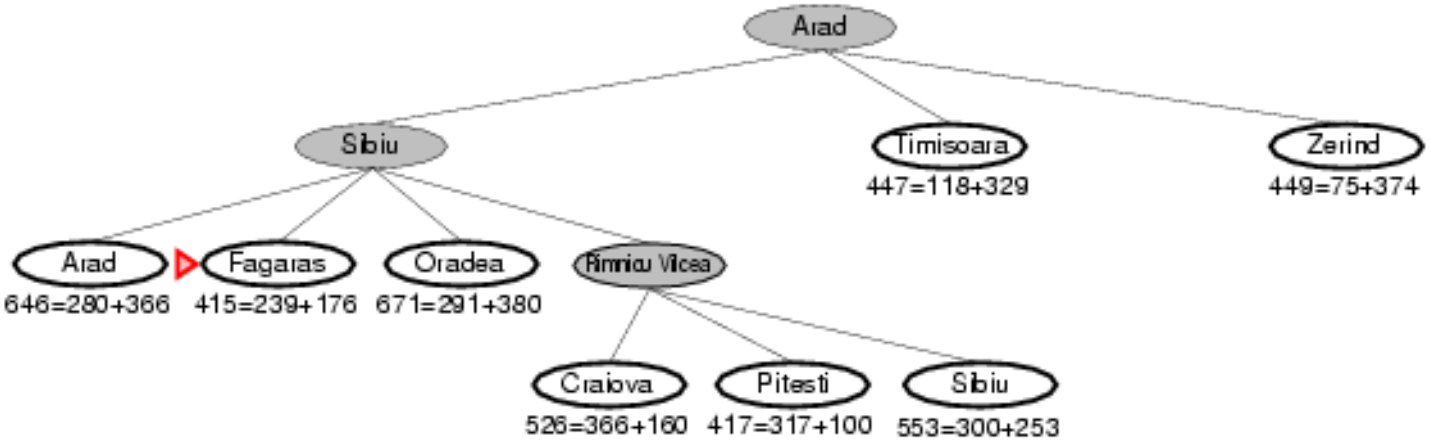


A* search example

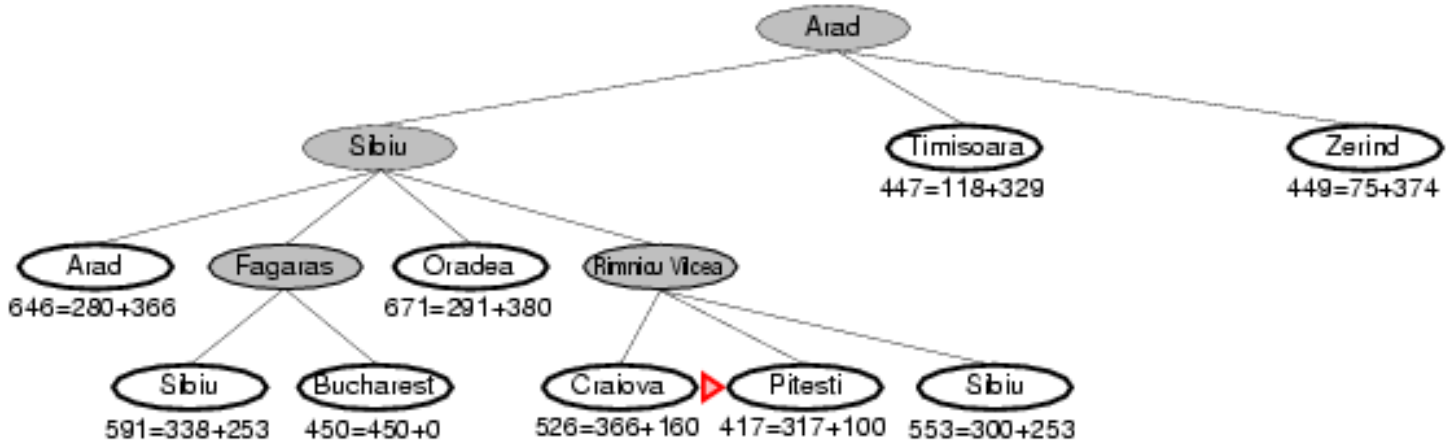
A* search example



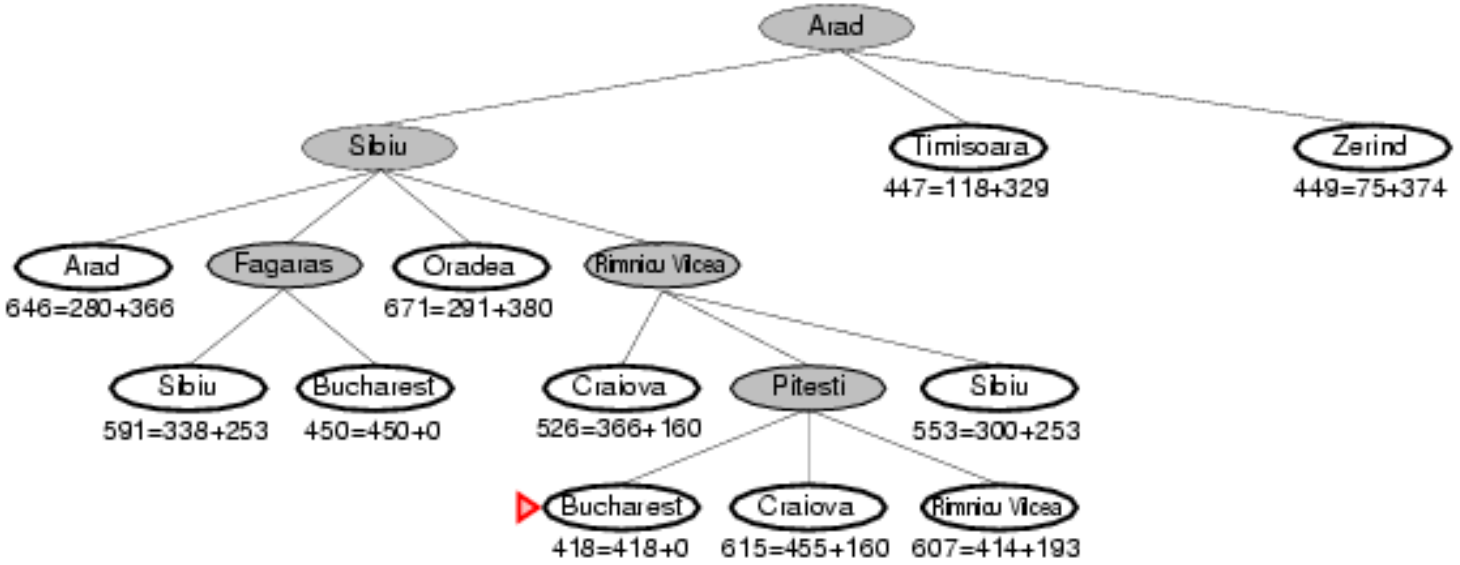
A* search example



A* search example

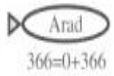


A* search example

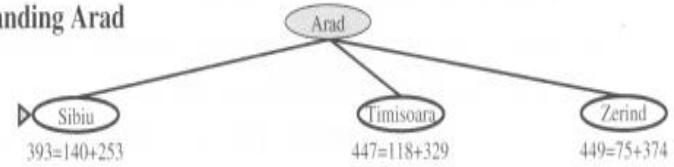


A* algorithm - Romania Example

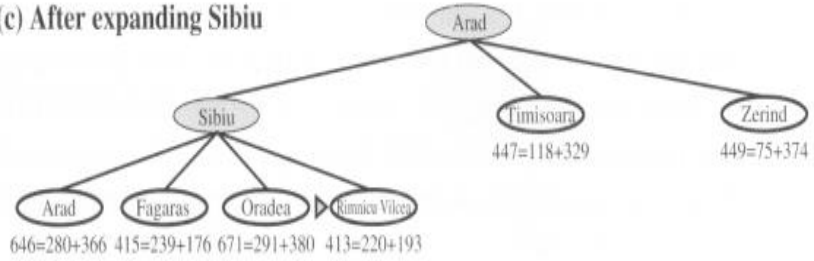
(a) The initial state



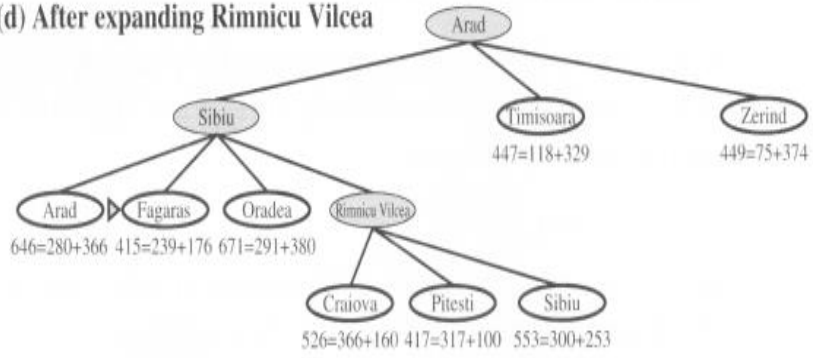
(b) After expanding Arad



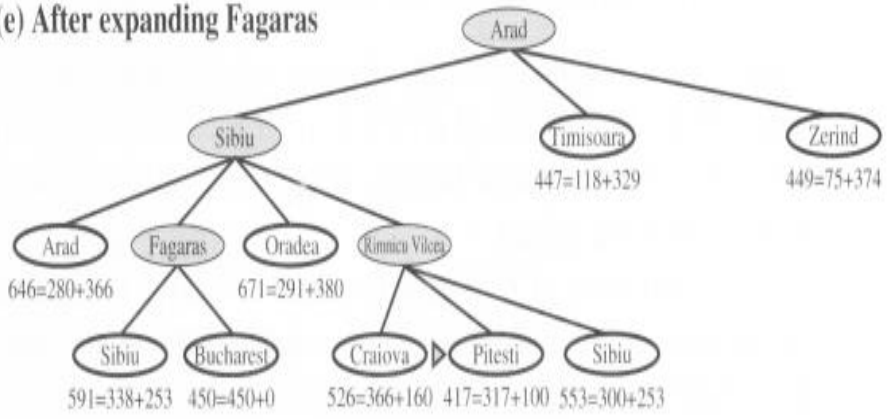
(c) After expanding Sibiu



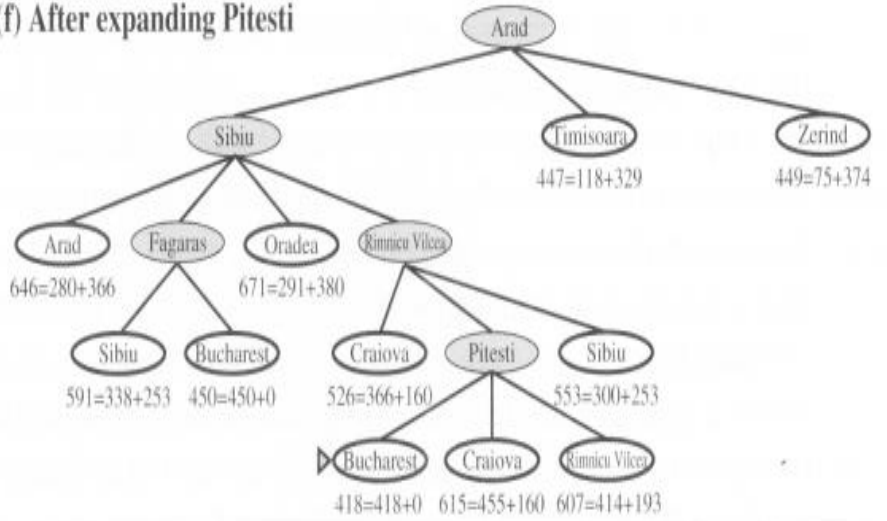
(d) After expanding Rimnicu Vilcea



(e) After expanding Fagaras



(f) After expanding Pitesti



A* algorithm Properties

How does A* search rate according to the four criteria of performance?

- **Complete:**
  - **It is complete**
    - **As long as the memory supports the depth and branching factor of the tree.**
- **Optimal:**
  - **It is optimal**
    - **Depends on the use of an admissible heuristic**
      - No algorithm with the same heuristic is guaranteed to expand fewer nodes
- **Time:**
  - **$O(b^d)$**
    - **Must keep track of the nodes evaluated so far**
- **Space:**
  - ***$O(b^d)$***
    - **Must keep track of the discovered nodes to be evaluated**

MONOTONICITY

- In the study of path-finding problems in artificial intelligence, a heuristic function is said to be **consistent**, or **monotone**, if its estimate is always less than or equal to the estimated distance from any neighboring vertex to the goal, plus the step cost of reaching that neighbor.

DEFINITION

MONOTONICITY

A heuristic function h is monotone if

1. For all states n_i and n_j, where n_j is a descendant of n_i,

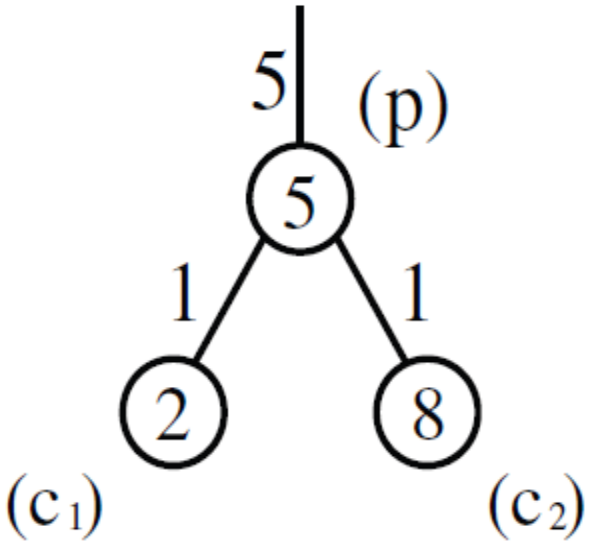
   $h(n_i) - h(n_j) \leq cost(n_i,n_j)$,

   where $cost(n_i,n_j)$ is the actual cost (in number of moves) of going from state n_i to n_j.

2. The heuristic evaluation of the goal state is zero, or h(Goal) = 0.

Inconsistent heuristics

- *h* decreases from parent to child
  - $f(p)=5+5=10$
  - $f(c1) = 6+2 = 8$
  - lower bound on the total cost of reaching the goal through c_1
  - information provided by evaluating $c1$ is "inconsistent"



Monotonicity Example

A simple argument can show that any monotonic heuristic is admissible. This argument considers any path in the space as a sequence of states $s_1, s_2, \ldots, s_g$, where s_1 is the start state and s_g is the goal. For the sequence of moves in this arbitrarily selected path:

s_1 to s_2	$h(s_1) - h(s_2) \leq cost(s_1,s_2)$	by monotone property
s_2 to s_3	$h(s_2) - h(s_3) \leq cost(s_2,s_3)$	by monotone property
s_3 to s_4	$h(s_3) - h(s_4) \leq cost(s_3,s_4)$	by monotone property
.	. . .	by monotone property
.	. . .	by monotone property
s_{g-1} to s_g	$h(s_{g-1}) - h(s_g) \leq cost(s_{g-1},s_g)$	by monotone property

Summing each column and using the monotone property of $h(s_g) = 0$:

path s_1 to s_g $\quad h(s_1) \leq cost(s_1,s_g)$

This means that monotone heuristic h is A* and admissible. It is left as an exercise whether or not the admissibility property of a heuristic implies monotonicity.

INFORMEDNESS

- When One Heuristic Is Better ? : More Informed Heuristics
- The more informed the search, the less the space that must be searched to get the minimal path solution

DEFINITION

INFORMEDNESS

For two A* heuristics h_1 and h_2, if $h_1(n) \leq h_2(n)$, for all states n in the search space, heuristic h_2 is said to be *more informed* than h_1.