B.S. Abdur Rahman Crescent Institute of Science & Technology
Deemed to be University u/s 3 of the UGC Act, 1956

CSD 3102 - ARTIFICIAL INTELLIGENCE TECHNIQUES

III YEAR CSE, CS & IoT

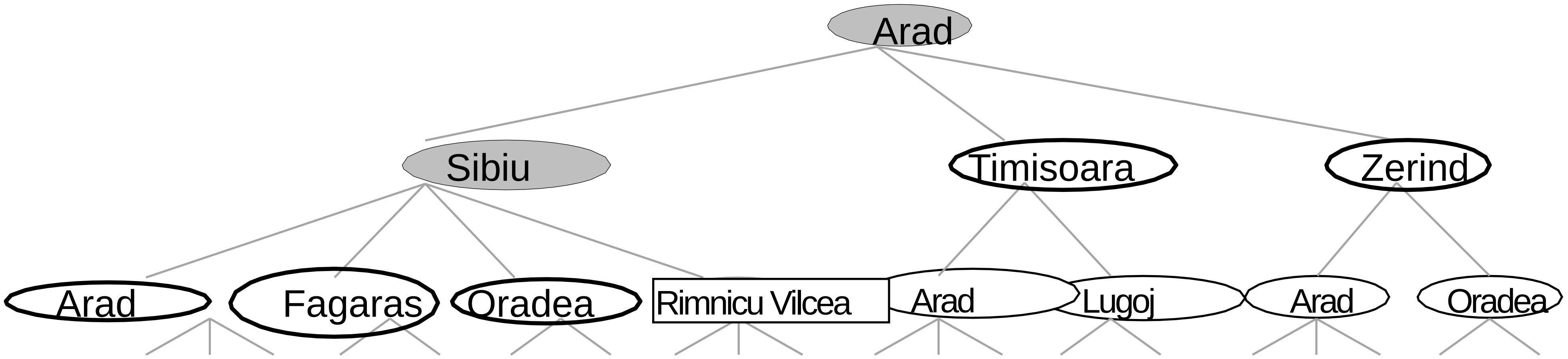
MODULE II	HEURISTIC SEARCH TECHNIQUES	9
General Search algorithm – Uniformed Search Methods – BFS, Uniform Cost Search - Depth First search , Depth Limited search (DLS), Iterative Deepening - Informed Search-Introduction- Generate and Test, BFS, A* Search, Memory Bounded Heuristic Search - Local Search Algorithms and Optimization Problems – Hill climbing and Simulated Annealing.		

Tree Search Algorithms

Basic Idea: offline, simulated exploration of state space by generating successors of already-explored states

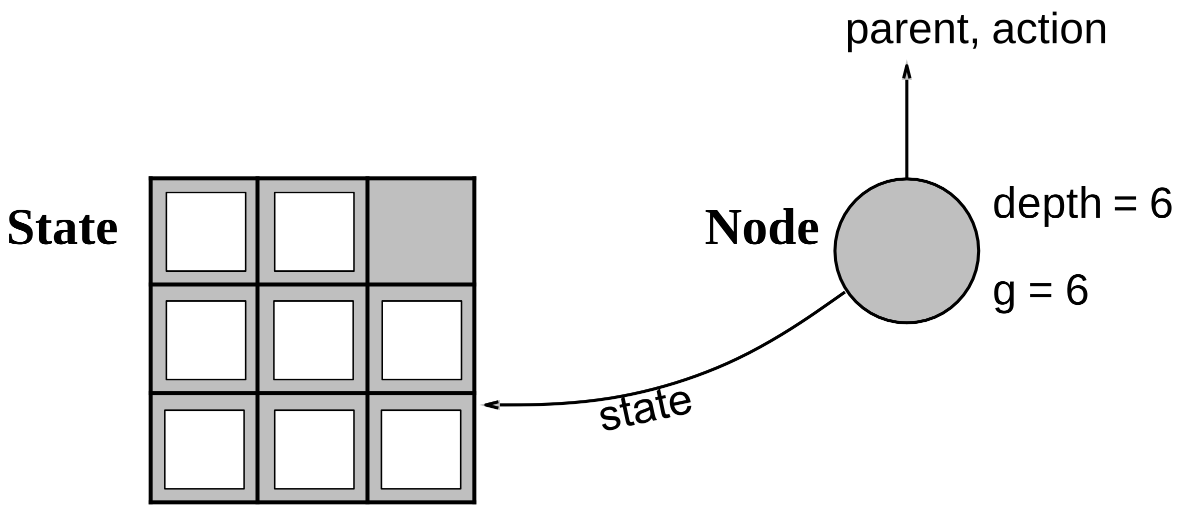
```
function Tree-Search( problem, strategy) returns a solution, or
failure  initialize the search tree using the initial state of problem
loop do
    if there are no candidates for expansion then return failure
else
    choose a leaf node for expansion according to strategy
    if the node contains a goal state then return the
       corresponding solution
else
    expand the node and add its successors to the tree
done
```

Tree Search Example



Implementation: states vs. nodes

- A *state* is a (representation of) a physical configuration
- A *node* is a data structure constituting part of a search tree (and includes such info as parent, children, depth, *path cost* $g(x)$)
- States do not have parents, children, depth, or path cost!



Search Techniques

- Un-informed (Blind) Search Techniques do not take into account the location of the goal. Intuitively, these algorithms ignore where they are going until they find a goal and report success. Uninformed search methods use only information available in the problem definition and past explorations, e.g. cost of the path generated so far. Examples are
  - – **Breadth-first search (BFS)**
  - – **Depth-first search (DFS)**
  - - **Depth Limited Search (DLS)**
  - – **Iterative deepening (IDA)**
  - - **Bi-directional search**

Breadth-first search (BFS)

Breadth-first search (BFS): At each level, we expand all nodes (possible solutions), if there exists a solution then it will be found.

Algorithm BFS

The algorithm uses a queue data structure to store intermediate results as it traverses the graph, as follows:

1. Create a queue with the root node and add its direct children
2. Remove a node in order from queue and examine it
    - If the element sought is found in this node, quit the search and return a result.
    - Otherwise append any successors (the direct child nodes) that have not yet been discovered.
3. If the queue is empty, every node on the graph has been examined – quit the search and return "not found".
4. If the queue is not empty, repeat from Step 2.

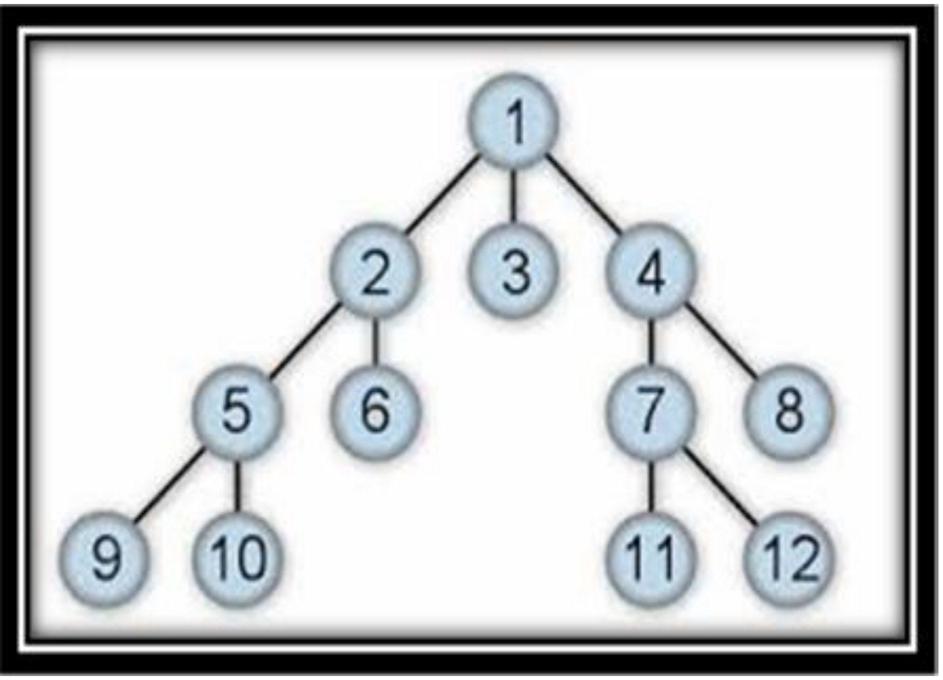
Depth-first search (DFS):

We can start with a node and explore with all possible solutions available with this node.

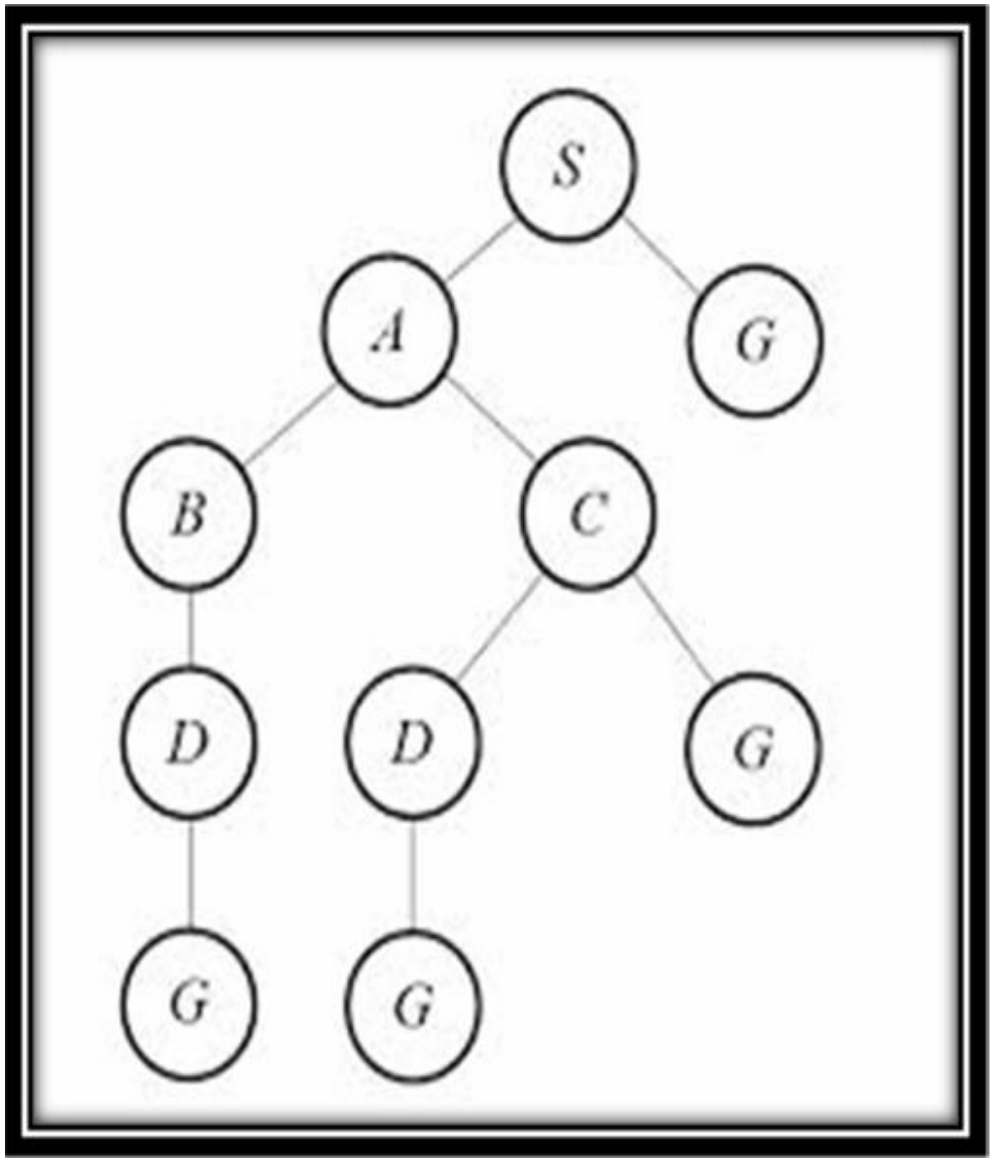
DFS starts at the root node and explores as far as possible along each branch before backtracking

1.Create a stack with the root node and add its direct children
2.Remove a node in order from stack and examine it
If the element sought is found in this node, quit the search and return a result.
Otherwise insert any successors (the direct child nodes)
that have not yet been discovered before existing nodes.
3.If the stack is empty, every node on the graph has been examined – quit the search and return "not found".
4.If the stack is not empty, repeat from Step 2.

Solve the below examples – using BFS and DFS



Consider the initial state is 1 and Goal state is 12

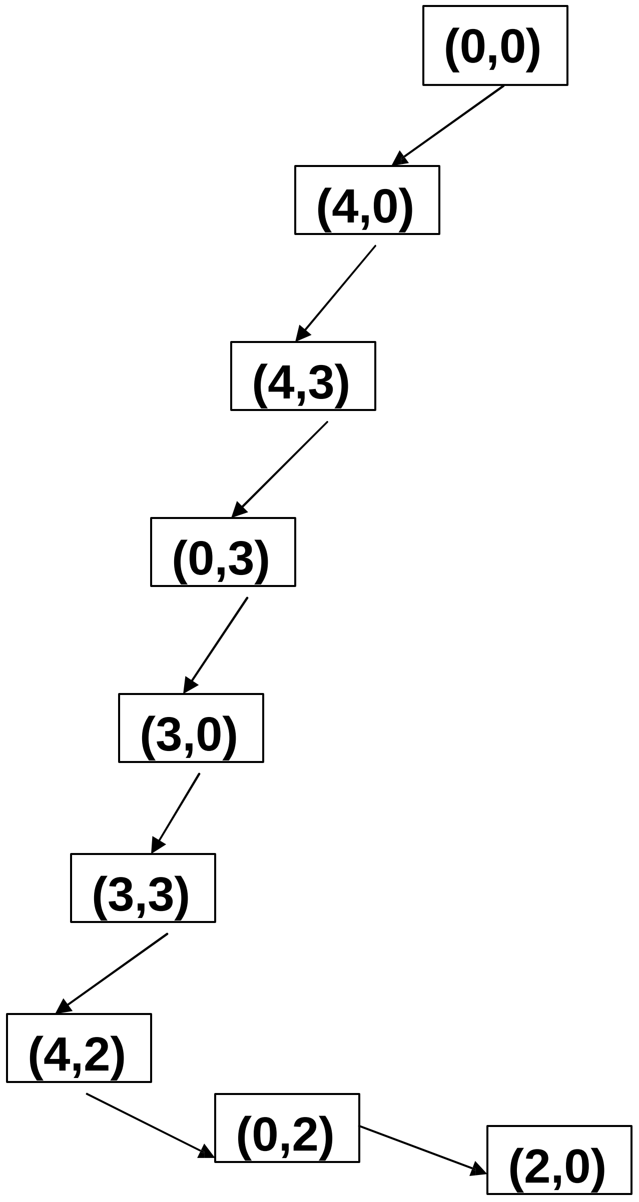


Consider the initial state is S and Goal state is D

BFS for a water jug problem

$$\begin{tikzcd}
 & & (0,0) \arrow[dl] \arrow[drr] & & & \\
 & (4,0) \arrow[dl] \arrow[d] \arrow[dr] & & & (0,3) \arrow[dl] \arrow[d] \arrow[dr] & \\
(4,3) & (0,0) & (1,3) & (4,3) & (0,0) & (3,0)
\end{tikzcd}$$

DFS for a water jug problem



Uniform Cost Search

- Like breadth-first search
- Expand node with lowest path cost.

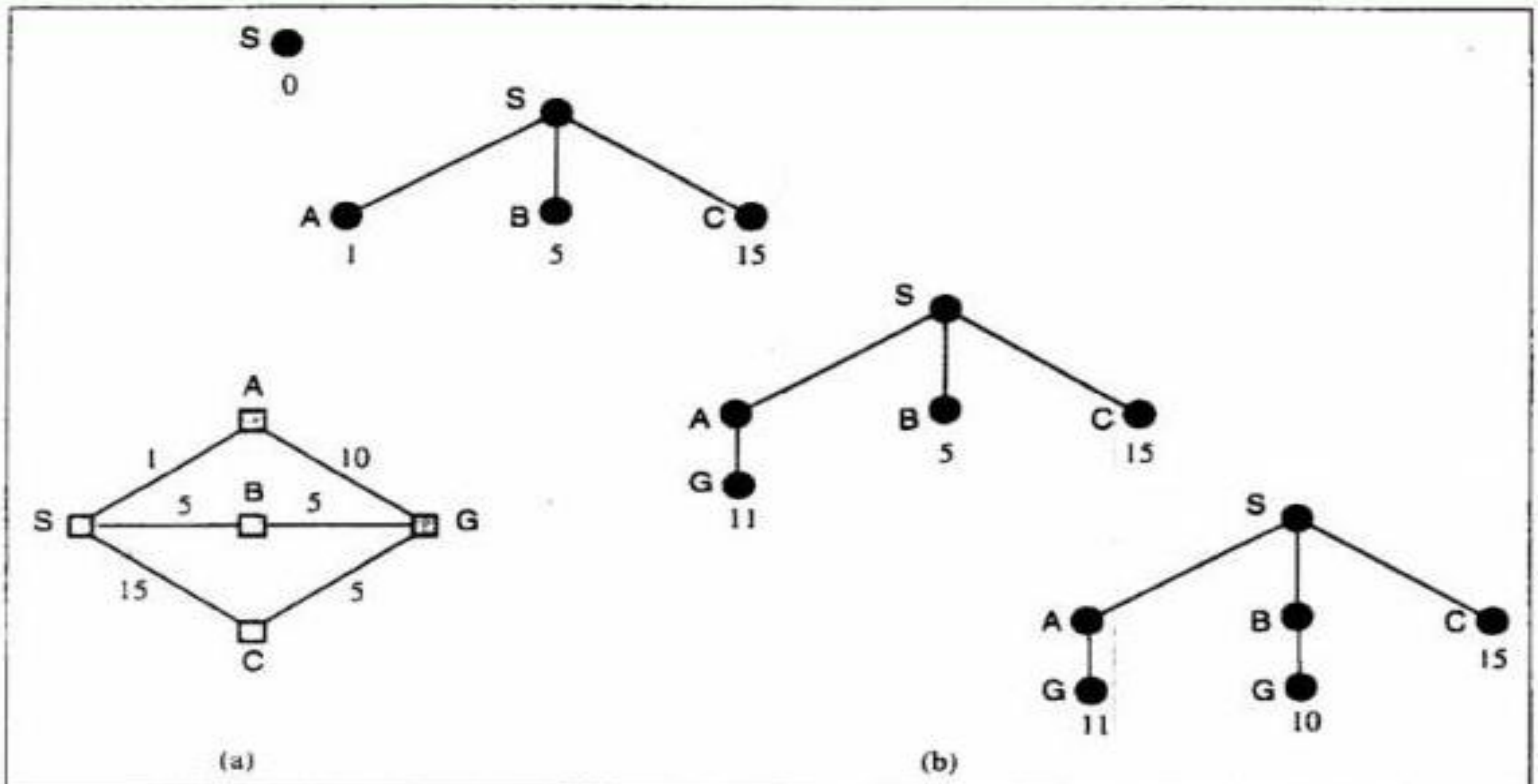
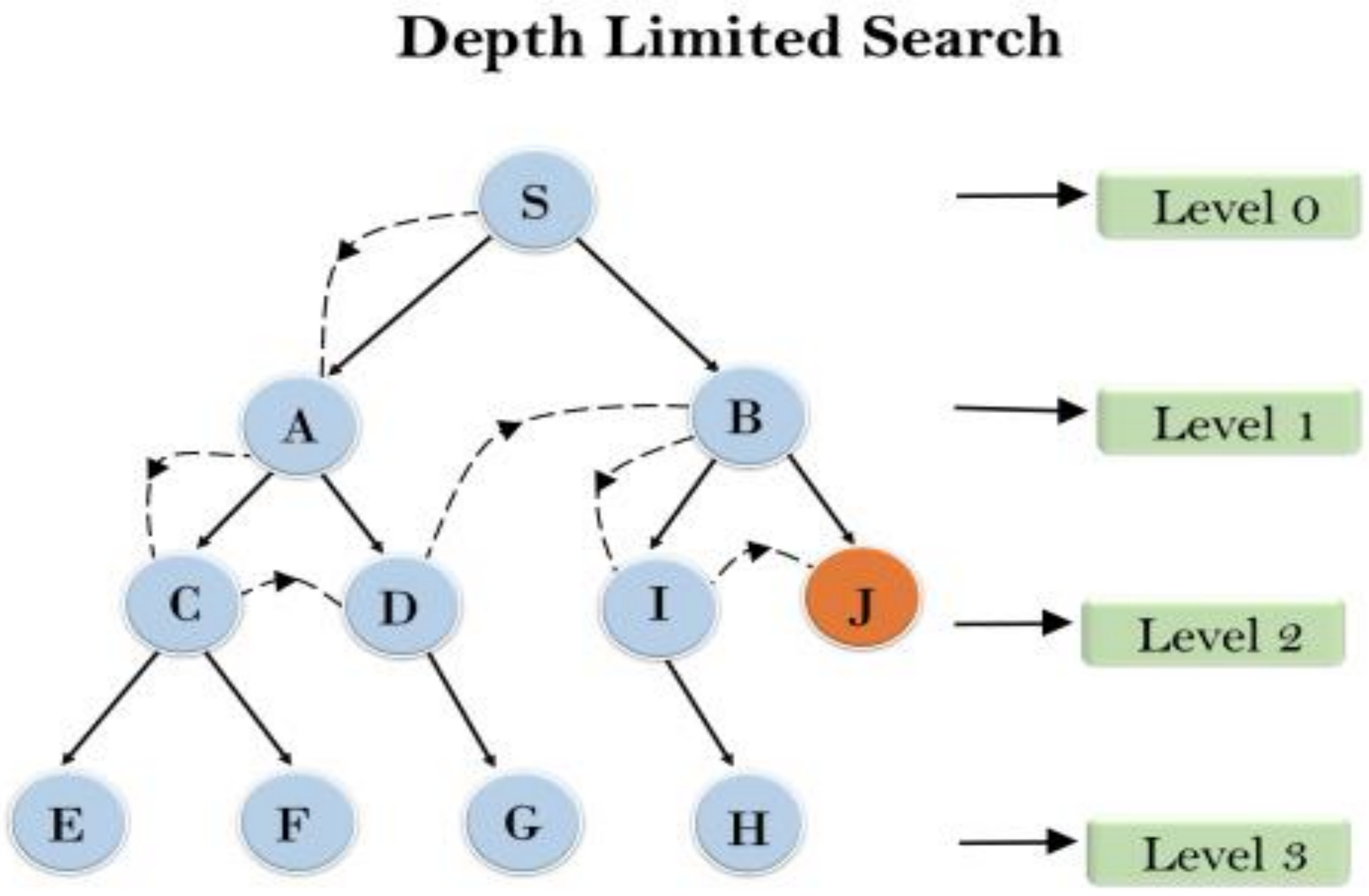


Figure 3.13 A route-finding problem. (a) The state space, showing the cost for each operator. (b) Progression of the search. Each node is labelled with $g(n)$. At the next step, the goal node with $g = 10$ will be selected.

Depth Limited Search

- **Like depth-first search but with depth limit L. Here Depth Limit is Level 2**

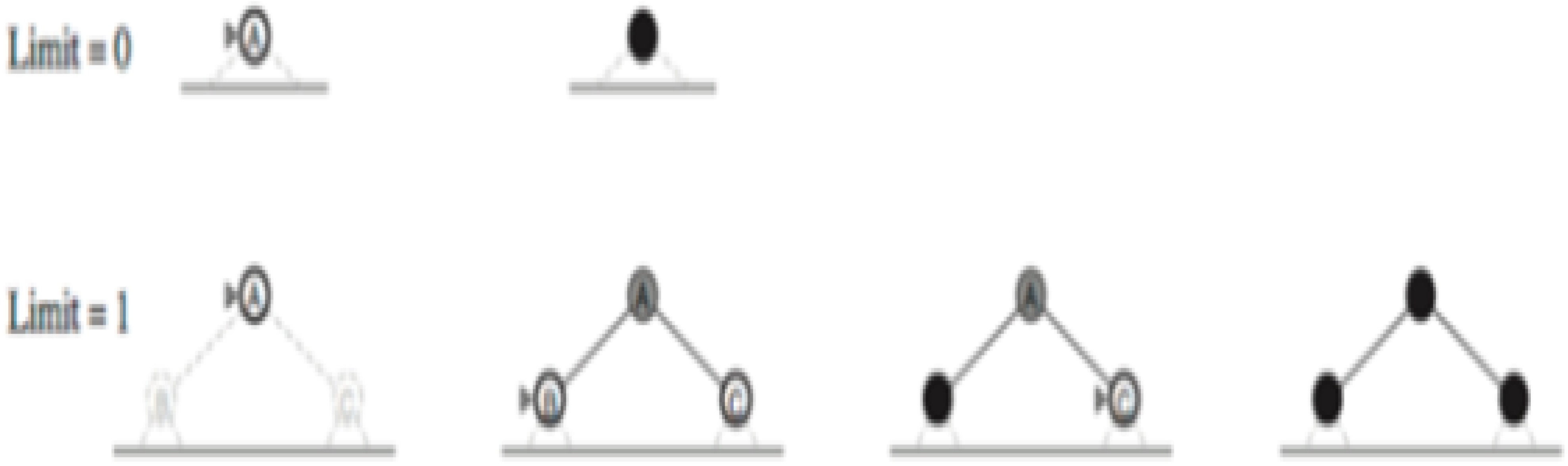


Iterative Deepening Search

```
function Iterative-Deepening-Search
  (problem) return soln
for limit from 0 to MAX-INT
do
result := Depth-Limited-Search(problem,
  limit)
if (result != cutoff) then return result
end for
end function
```

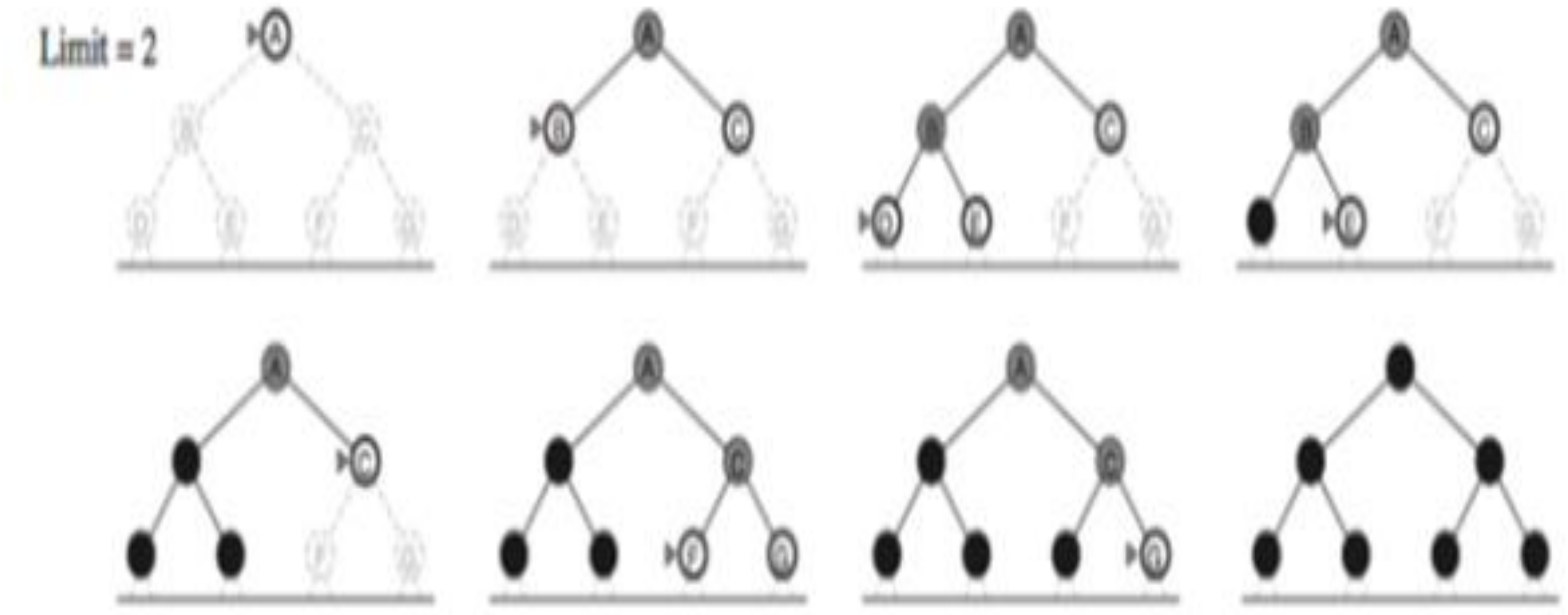
Iterative Deepening Search

Starting Limit=0, then Expanding Limit=1



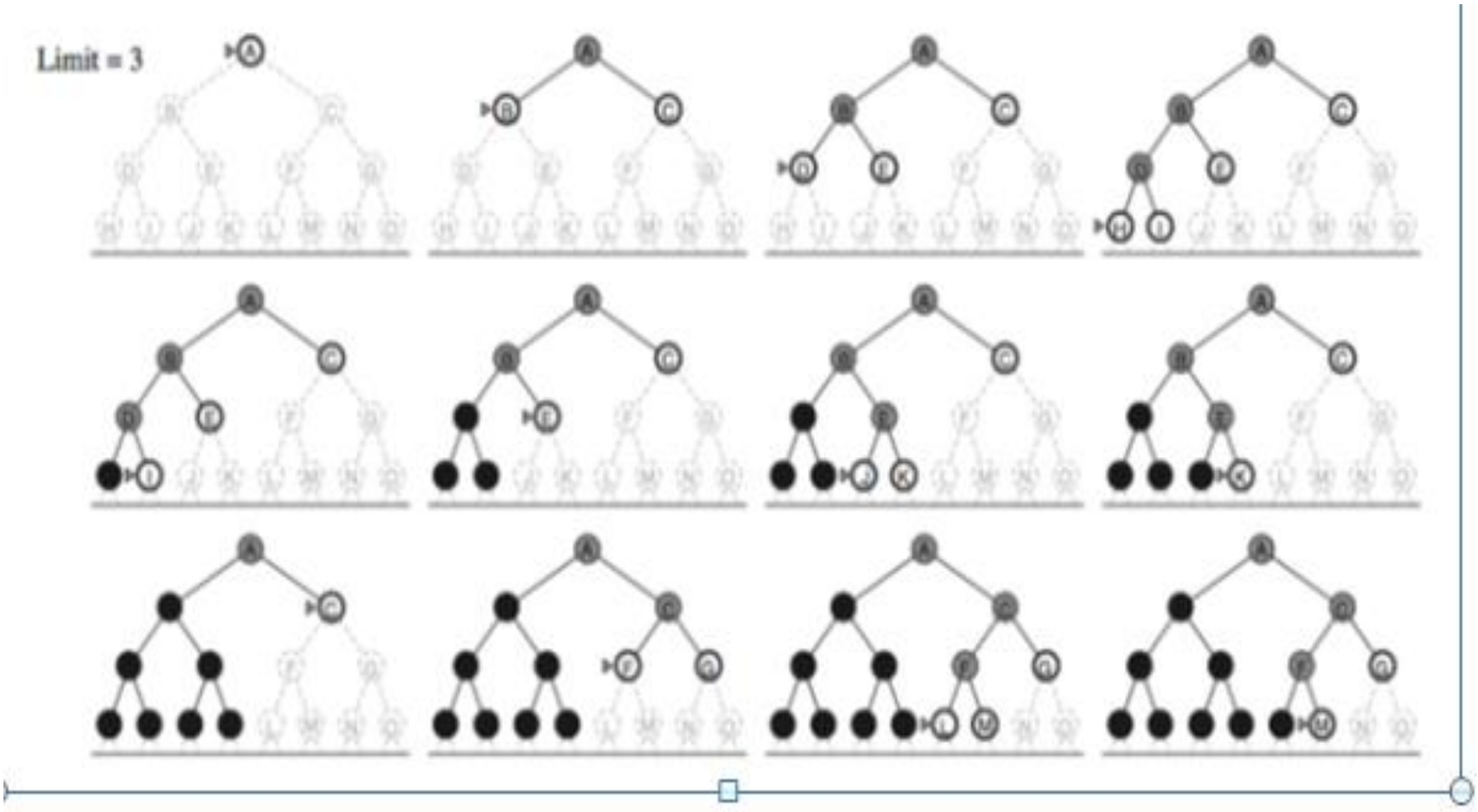
Iterative Deepening Search

Expanding Limit=2

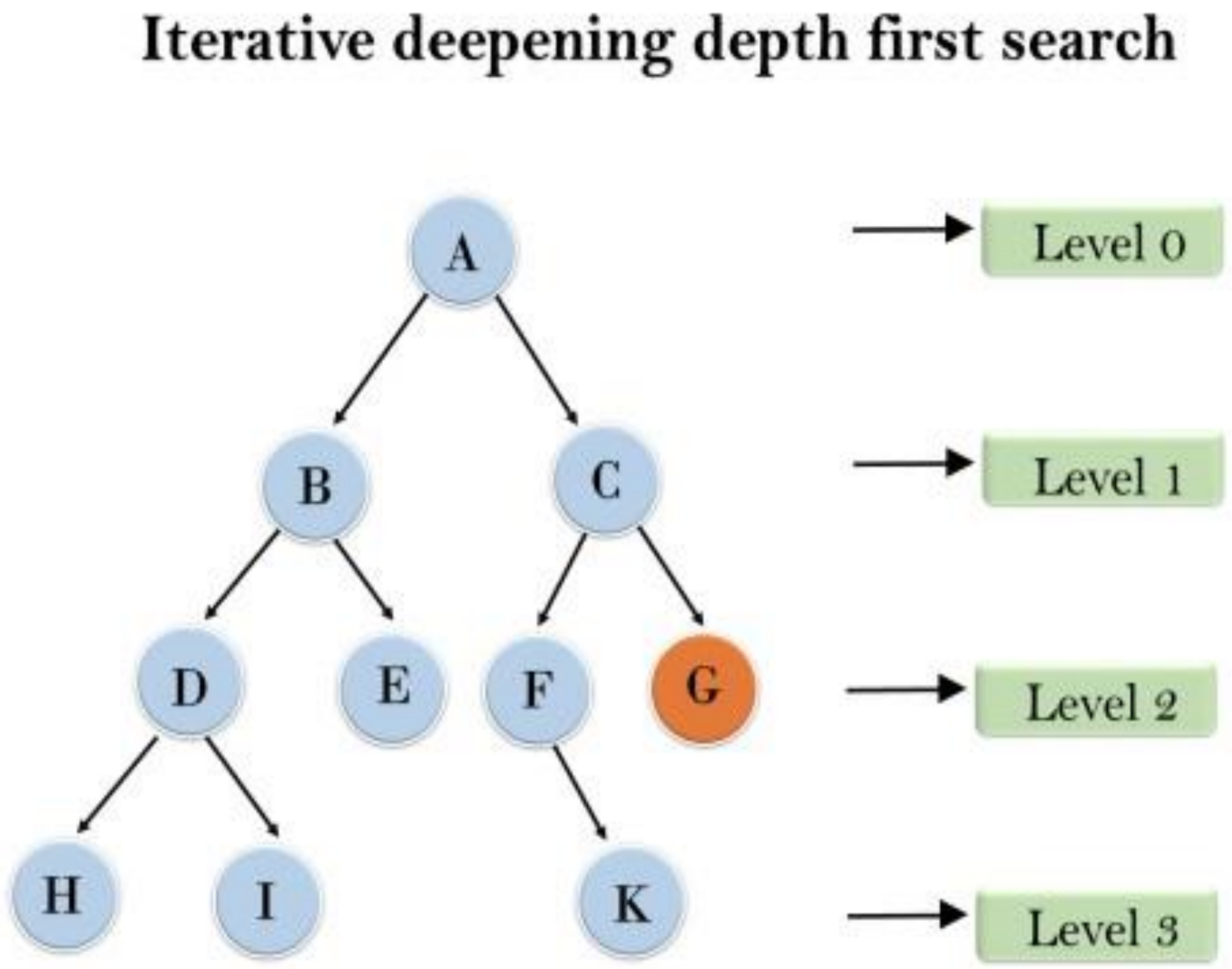


Iterative Deepening Search

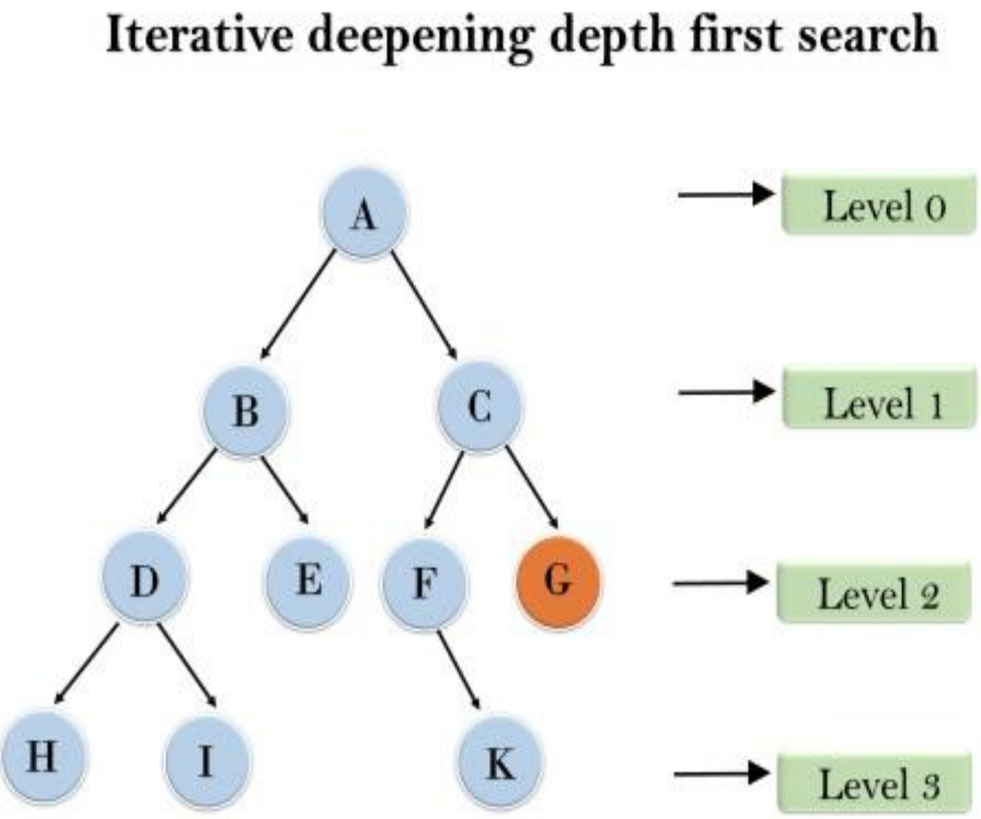
Expanding Limit=3



Iterative deepening depth- first Search\ Example-2:



Iterative deepening depth- first Search



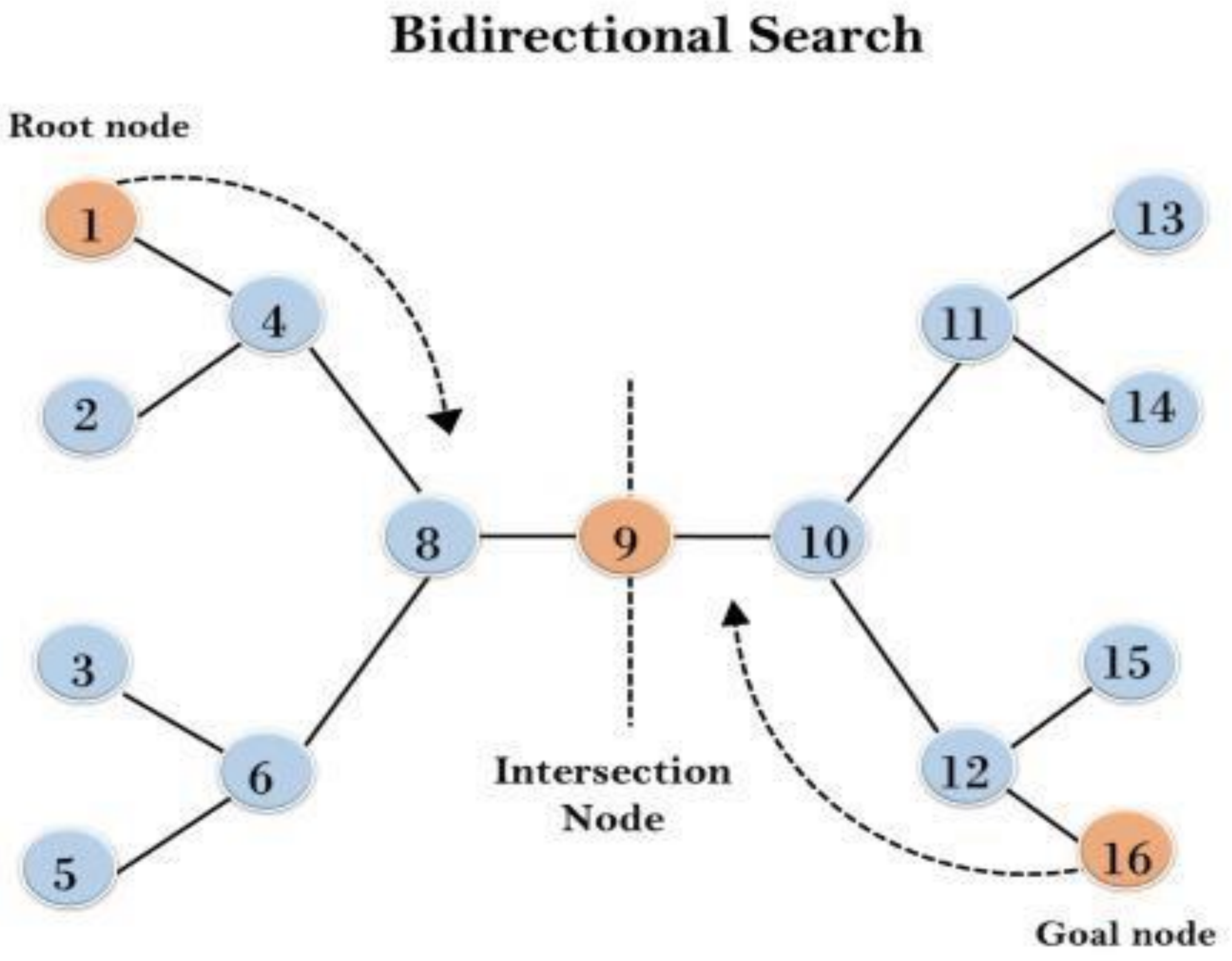
1'st Iteration-----> A
2'nd Iteration----> A, B, C
3'rd Iteration------>A,B,D,E,C,F, G
4'th Iteration------ >A,B,D,H,I,E,C,F,K,G

In the fourth iteration, the algorithm will find the goal node.

Bidirectional Search Algorithm

- Bidirectional search algorithm runs two simultaneous searches, one form initial state called as **forward-search** and other from goal node called as **backward-search**, to find the goal node.
- Bidirectional search replaces one single search graph with two small subgraphs in which one starts the search from an initial vertex and other starts from goal vertex.
- The search stops when these two graphs intersect each
  - other

Bidirectional Search Algorithm



Advantages and Disadvantages

- **Advantages:**
  - Bidirectional search is fast.
  - Bidirectional search requires less memory
- **Disadvantages:**
  - Implementation of the bidirectional search tree is difficult.
  - **In bidirectional search, one should know the goal**
    - **state in advance.**

Search Techniques

- Informed Search Techniques A search strategy which is better than another at identifying the most promising branches of a search-space is said to be more *informed*. It incorporates additional measure of a potential of a specific state to reach the goal. The potential of a state (node) to reach a goal is measured through a **heuristic function.** These are also called intelligent search
- Best first search
- Greedy Search
- A^* search

- In every informed search (Best First or A^* Search), there is a heuristic function and or a local function g(n). The heuristic function at every state decides the direction where next search is to be made.

Heuristics

- “Heuristics are criteria, methods or principles for deciding which among several alternative courses of action promises to be the most effective in order to achieve some goal”.
- In heuristic search or informed search, heuristics are used to identify the most promising search path.
- Heuristics can estimate the **“goodness”** of a particular node (or state) n:

How close is n to a goal node?

What might be the minimal cost path from n to a goal node?

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

Example of Heuristic Function (8-puzzle)

(i) **Misplaced Tiles Heuristics:** - number of tiles out of place.

2	8	3
1	6	4
	7	5

Initial State

1	2	3
8		4
7	6	5

Goal state

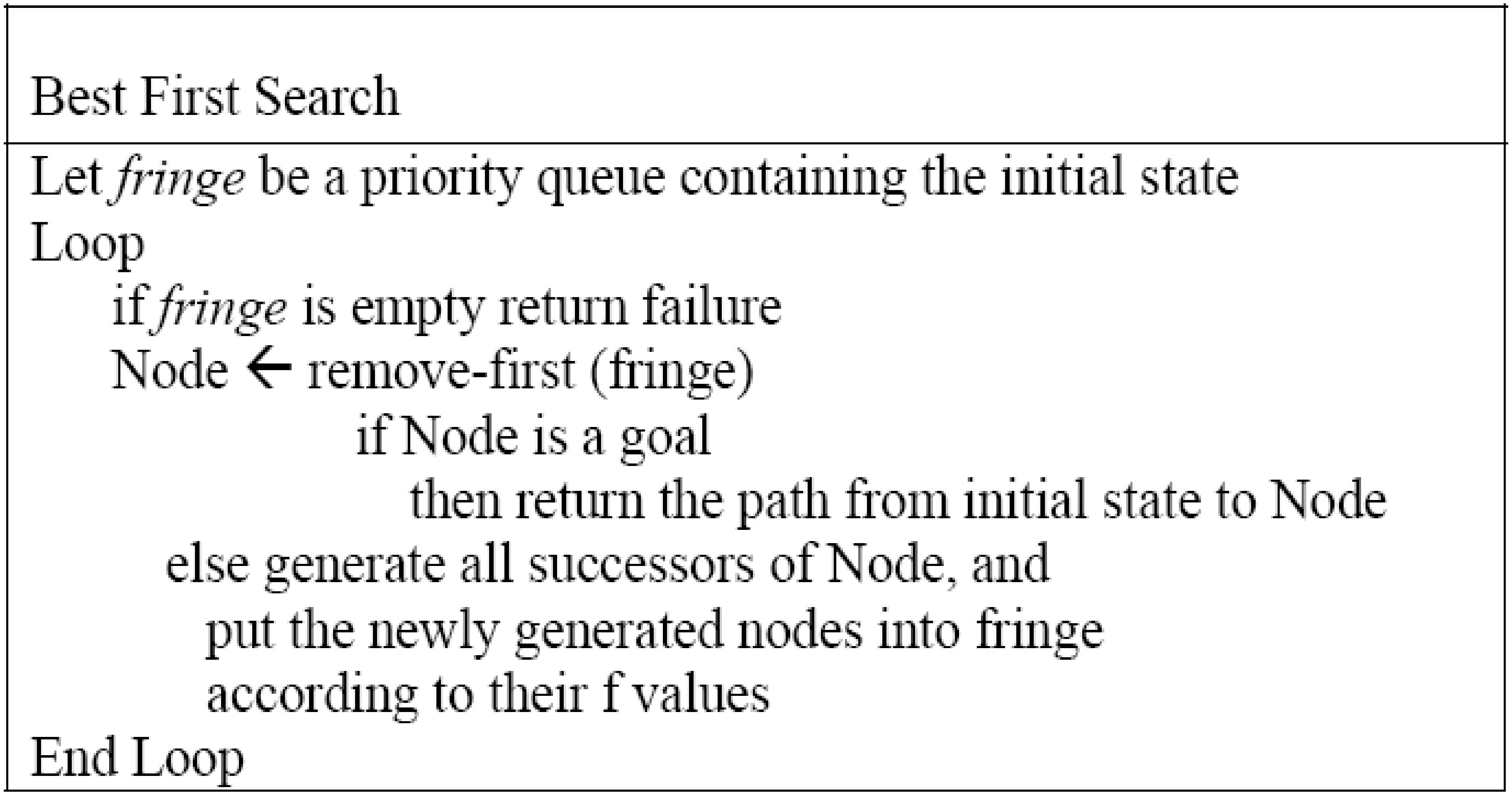
The first picture shows the current state n, and the second picture the goal state.

h(n) = 5 : because the tiles 2, 8, 1, 6 and 7 are out of place.

(ii) Manhattan Distance Heuristic: This heuristic sums the distance that the tiles are out of place. The distance of a tile is measured by the sum of the differences in the x-positions and the y-positions.

h(n) = 1 + 1 + 0 + 0 + 0 + 1 + 1 + 2 = 6

Best First Search Algorithm



Best First Search

```
Let *fringe* be a priority queue containing the initial state
Loop
    if *fringe* is empty return failure
    Node ← remove-first (fringe)
        if Node is a goal
            then return the path from initial state to Node
      else generate all successors of Node, and
        put the newly generated nodes into fringe
        according to their f values
End Loop
```

We will now consider different ways of defining the function f. This leads to different search algorithms.

Algorithm for Greedy Best First Search

We use a heuristic function

$f(n) = h(n)$

- h(n) estimates the distance remaining to a goal.

- It expands nodes in the order of their heuristic values.

Let h(n) be the heuristic function in a graph. In simple case, let it be the straight line distance SLD from a node to destination.

1. Start from source node S, determine all nodes outward from S and queue them.
2. Examine a node from queue (as generated in 1) .
   - ❖ If this node is desired destination node, stop and return success.
   - ❖ Evaluate h(n) of this node. The node with optimal h(n) gives the next successor, term this node as S.
3. Repeat steps 1 and 2.

Algorithm for A* Search

Let h(n) be the heuristic function in a graph. In simple case, let it be the straight line distance SLD from a node to destination. Let g(n) be the function depending on the distance from source to current node. Thus $f(n) = g(n) + h(n)$

1.Start from source node S, determine all nodes outward from S and queue them.

2.Examine a node from queue (as generated in 1) .

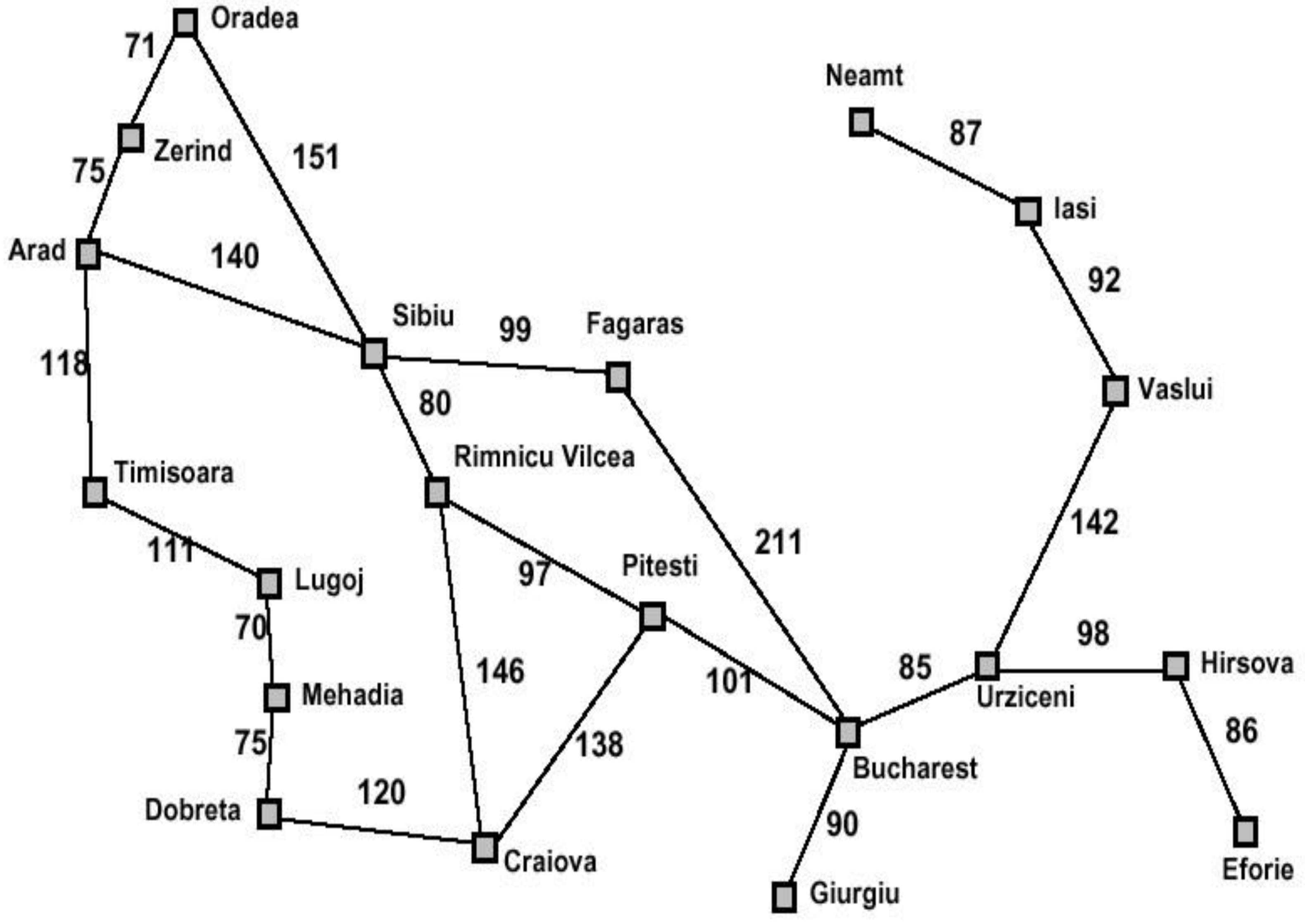
- * If this node is desired destination node, stop and return success.
- * Evaluate f(n) at this node. The node with optimal f(n) gives the next successor, term this node as S.

3.Repeat steps 1 and 2.

Time = O(log f(n)) where h(n) is the actual distance travelled from n to goal

Best First Search An Example

There are cities in a country (Romania). The task is to reach from Arad) to Bucharest

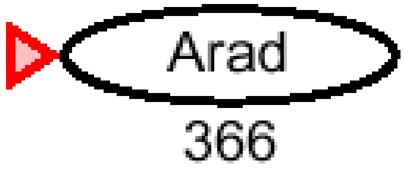


Straight−line distance to Bucharest

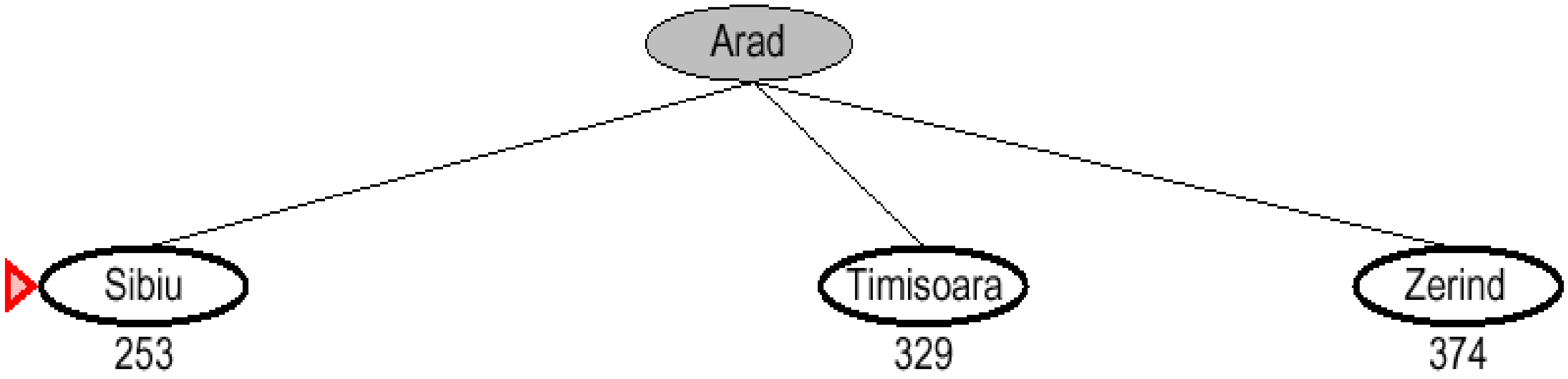
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Method: Greedy Best First Search: Start from Source (Arad). At each possible outward node n from S, write the heuristic function h(n). Proceed further in the direction in which h(n) is minimum. Repeat the exercise till goal (destination-Bucharest) is achieved

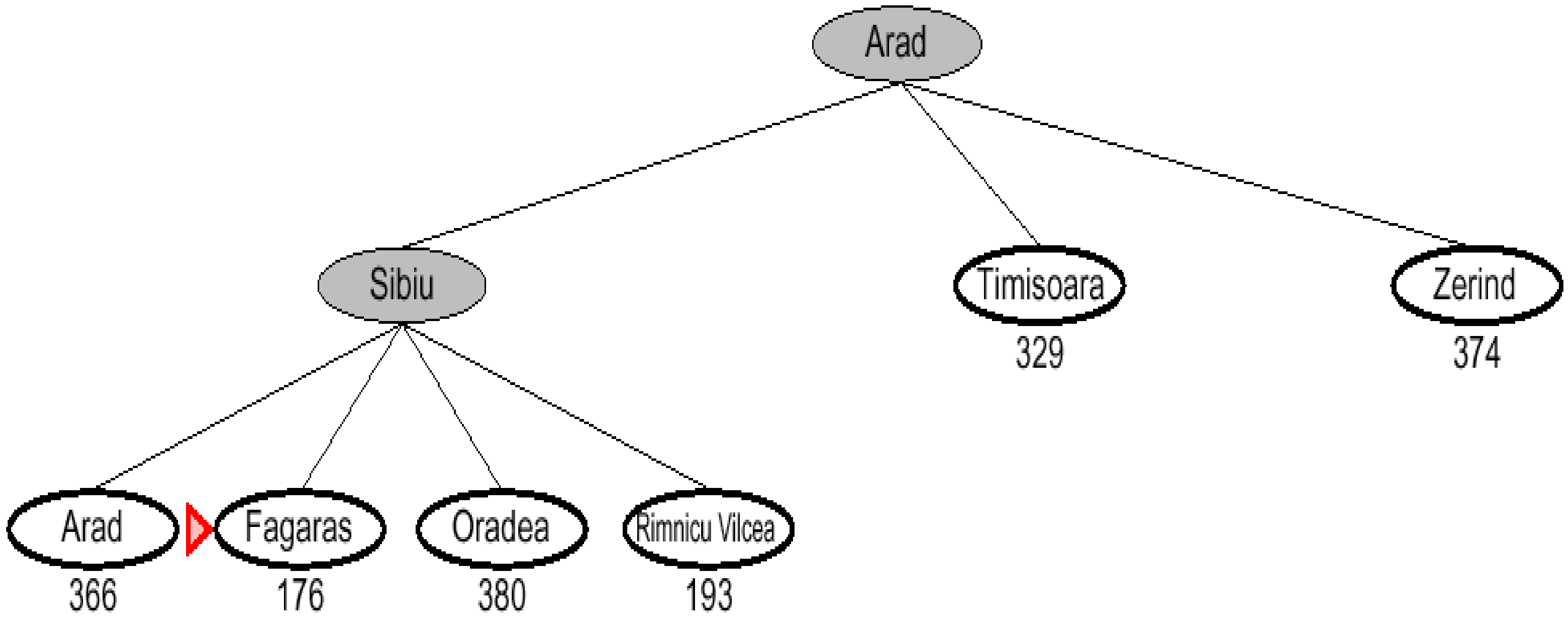
Step 1:



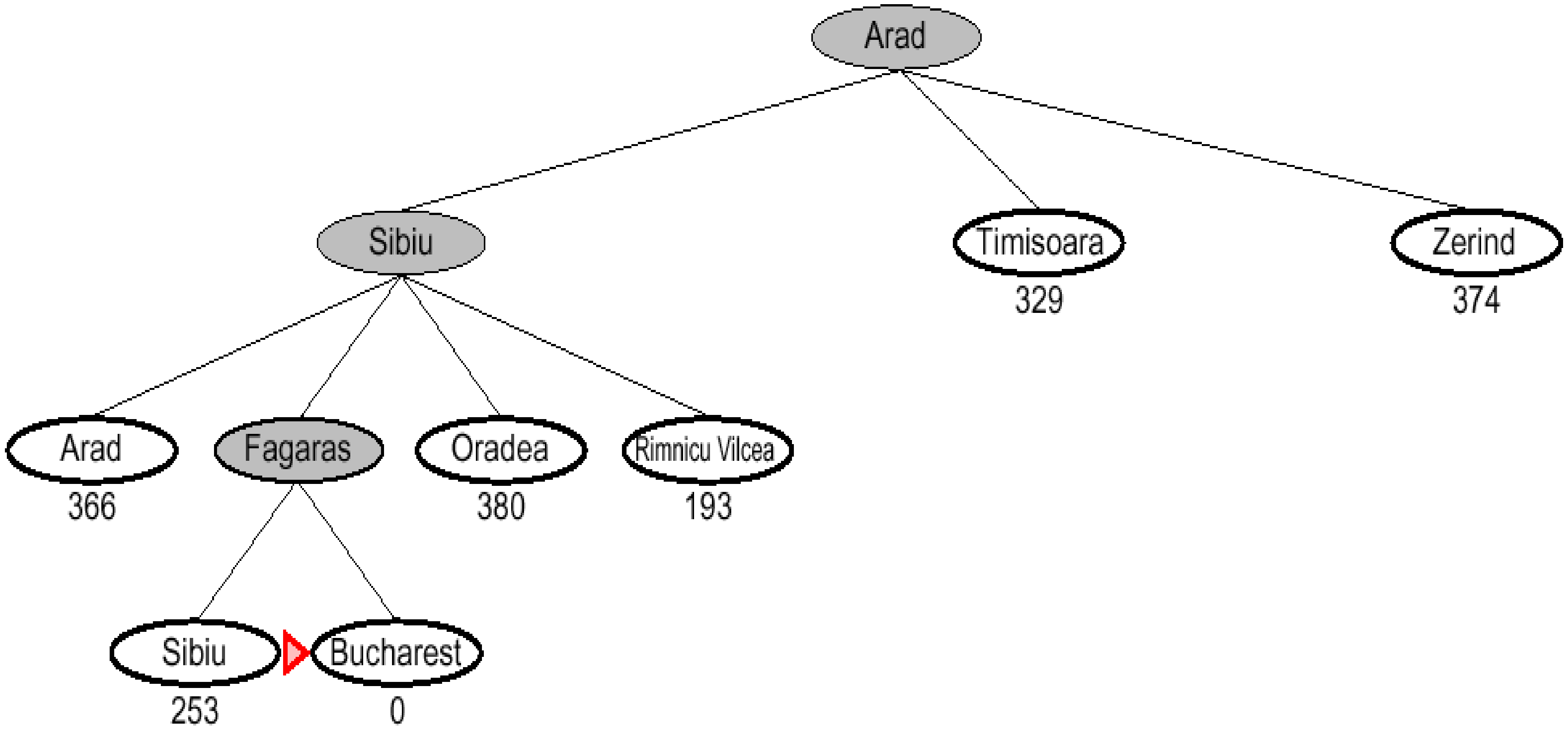
Step 2:



Step 3:

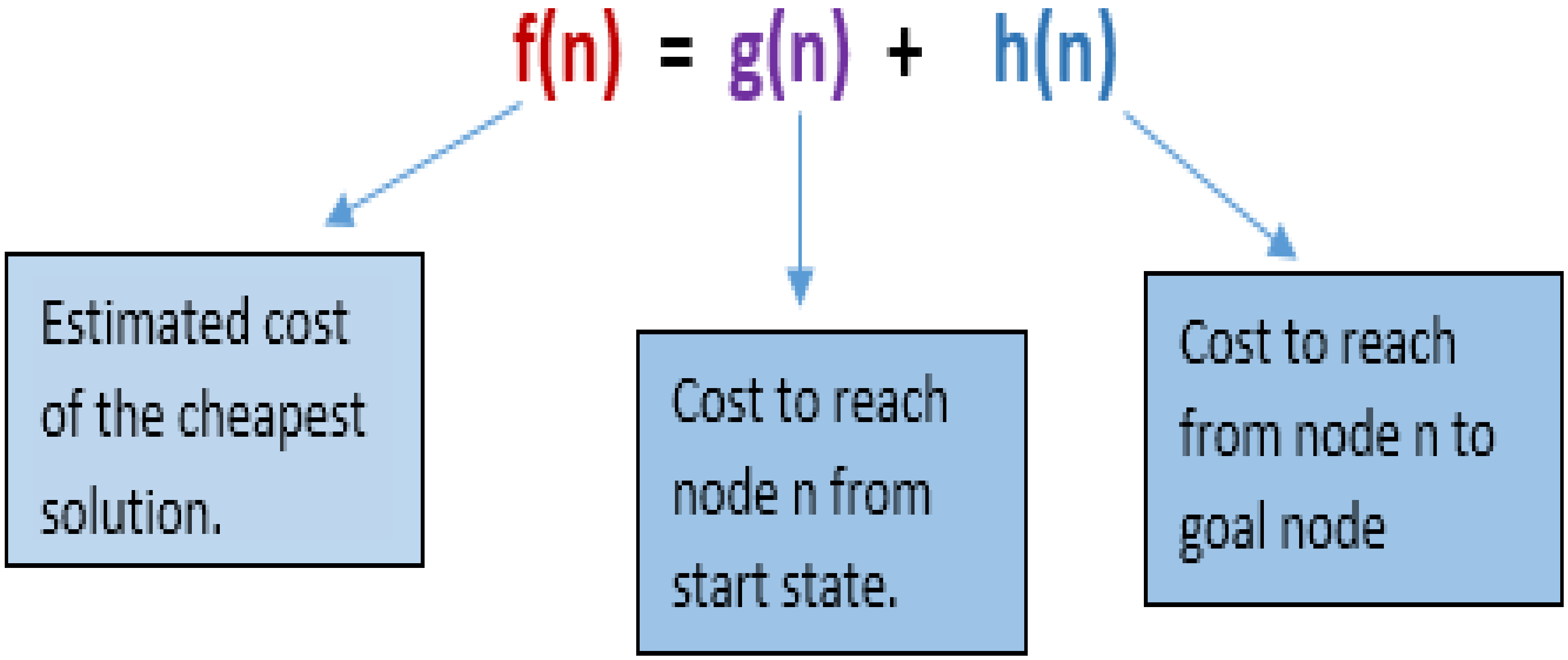

Arad
Sibiu
Timisoara
329
Zerind
374
Arad
366
Fagaras
176
Oradea
380
Rimnicu Vilcea
193

Step 4:

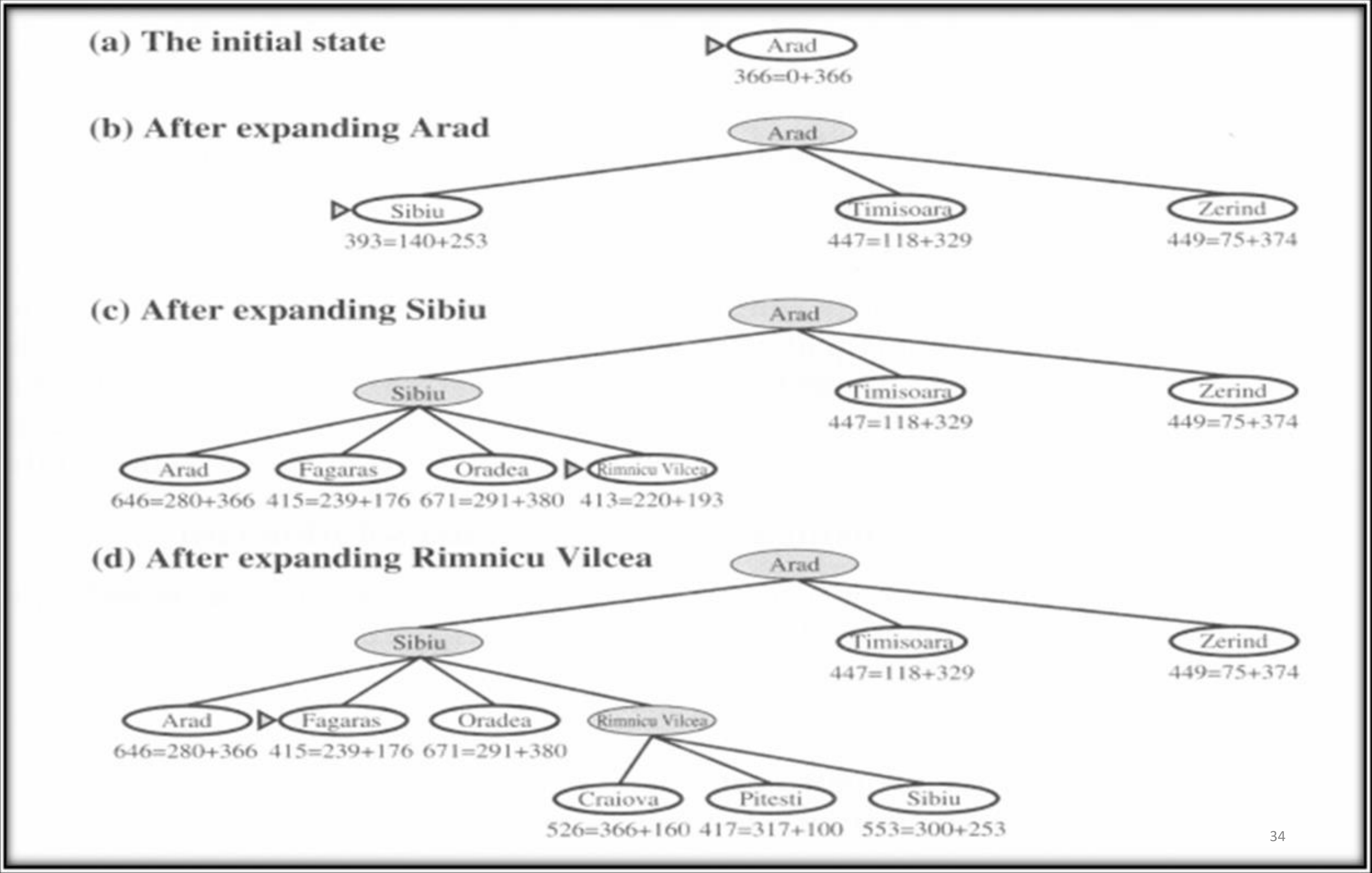

Arad
Sibiu
Timisoara
329
Zerind
374
Arad
366
Fagaras
Oradea
380
Rimnicu Vilcea
193
Sibiu
253
Bucharest
0

Method: **A^* Search:** Start from Source (Arad). At each possible outward n, node from S, calculate **f(n)=g(n)+h(n)**, where the heuristic function is h(n) and the total distance travelled so far is g(n). Proceed further in the direction in which h(n)(is minimum. Repeat the exercise till goal (destination- Bucharest) is achieved

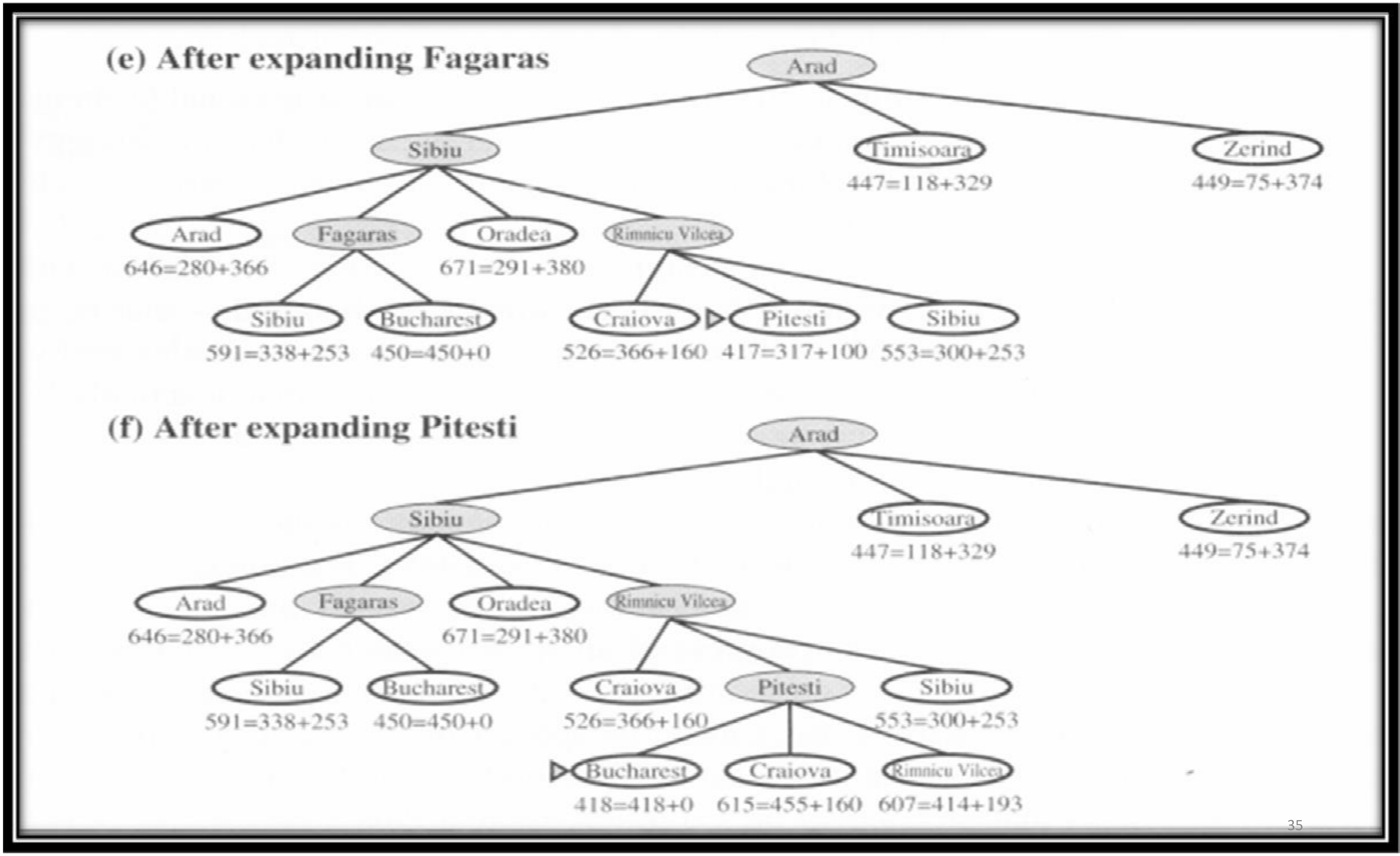
- **fitness number/ evaluation function**


f(n) = g(n) + h(n)
Estimated cost of the cheapest solution.
Cost to reach node n from start state.
Cost to reach from node n to goal node

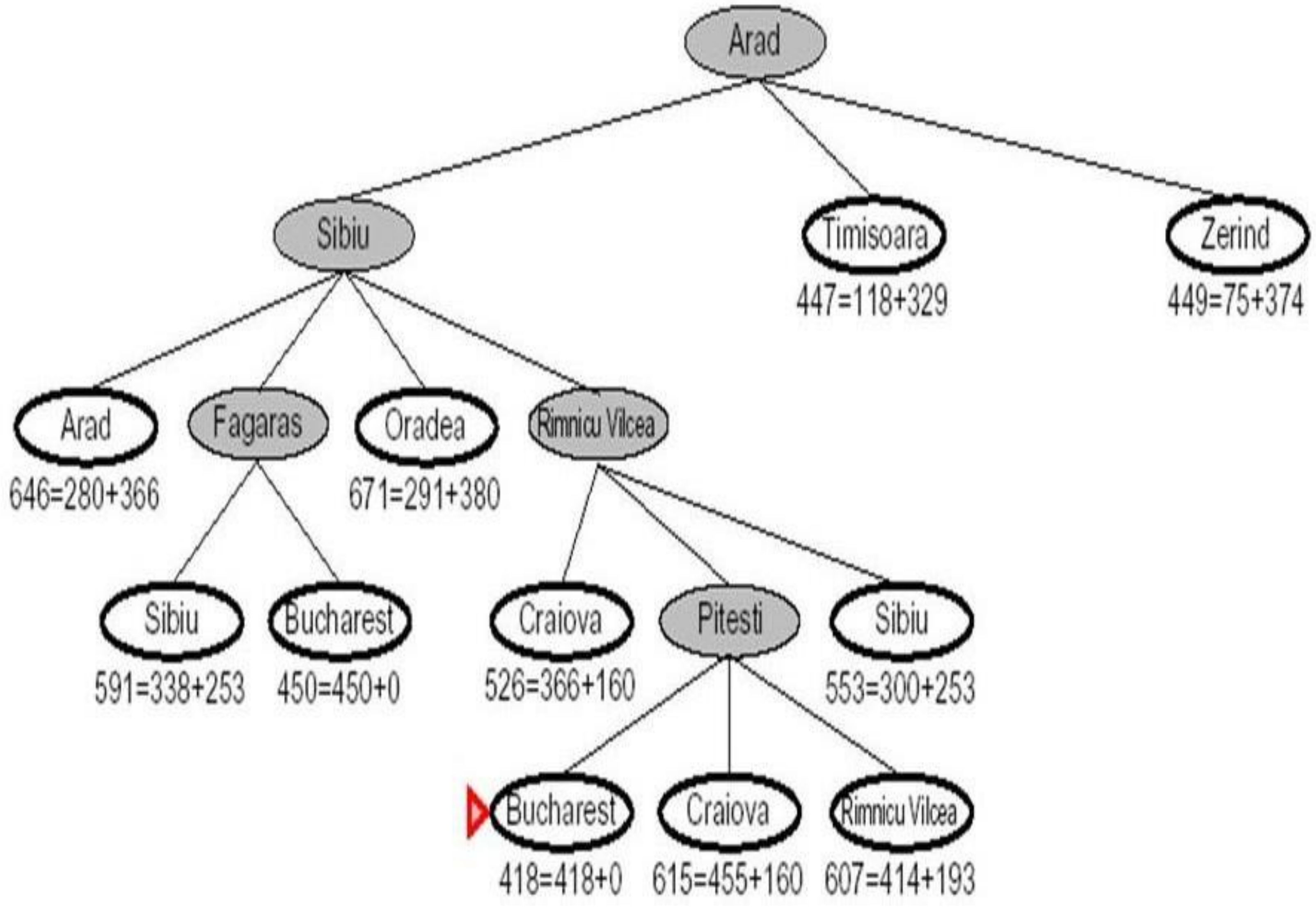
Method: A* Search: Start from Source (Arad). At each possible outward n, node from S, calculate **f(n)=g(n)+h(n)**, where the heuristic function is h(n) and the total distance travelled so far is g(n). Proceed further in the direction in which h(n)(is minimum. Repeat the exercise till goal (destination- Bucharest) is achieved



Method: A* Search: Start from Source (Arad). At each possible outward node from S, write the heuristic function h(n). Add the total distance travelled so far g(n). Proceed further in the direction in which h(n)(is minimum. Repeat the exercise till goal (destination- Bucharest) is achieved



Heuristic Search : A* Algorithm



Heuristics

- Where the exhaustive search is impractical, heuristic methods are used to speed up the process of finding a satisfactory solution via mental shortcuts to ease the cognitive load of making a decision. Examples of this method include using a rule of thumb, an educated guess, an intuitive judgment, stereotyping, or common sense.
- In more precise terms, heuristics are strategies using readily accessible, though loosely applicable, information to control problem solving in human beings and machines. Error and trial is simplest form of heuristics. We can fit some variables in an algebraic equation to solve it.

A* SEARCH-PROPERTIES

- **Admissible:** the algorithm A* is admissible . This means that provided a solution exists, the first solution found by A* is an optimal solution. A* is admissible under following conditions:
  - In the state space graph
    - Every node has a finite number of successors
    - Every arc in the graph has a cost greater than some $\varepsilon > 0$
  - Heuristic function: for every node n, $h(n) \leq h^*(n)$
- **Complete**: A* is also complete under the aboveconditions.
- **Optimal:** A* is optimally efficient for a given heuristic-of the optimal search algorithm that expand search paths from the root node, it can be shown that no other optimal algorithm will expand fewer nodes and find a solution

Local Search Algorithms - Generate and Test Heuristic Search

Algorithm: Generate-and-Test

1. Generate a possible solution. For some problems. this means generating a particular point in the problem space. For others, it means generating a path from a start state.
2. Test to see if this is actually a solution by comparing the chosen point or the endpoint of the chosen path to the set of acceptable goal states.
3. If a solution has been found, quit. Otherwise, return to step 1.

Local Search Algorithms

Features:

1. Acceptable for simple problems.
2. Inefficient for problems with large space.

Local Search Algorithms

- Just operate on a single current state rather than multiple paths
- Generally move only to neighbors of that state
- The paths followed by the search are not retained hence the method is not systematic

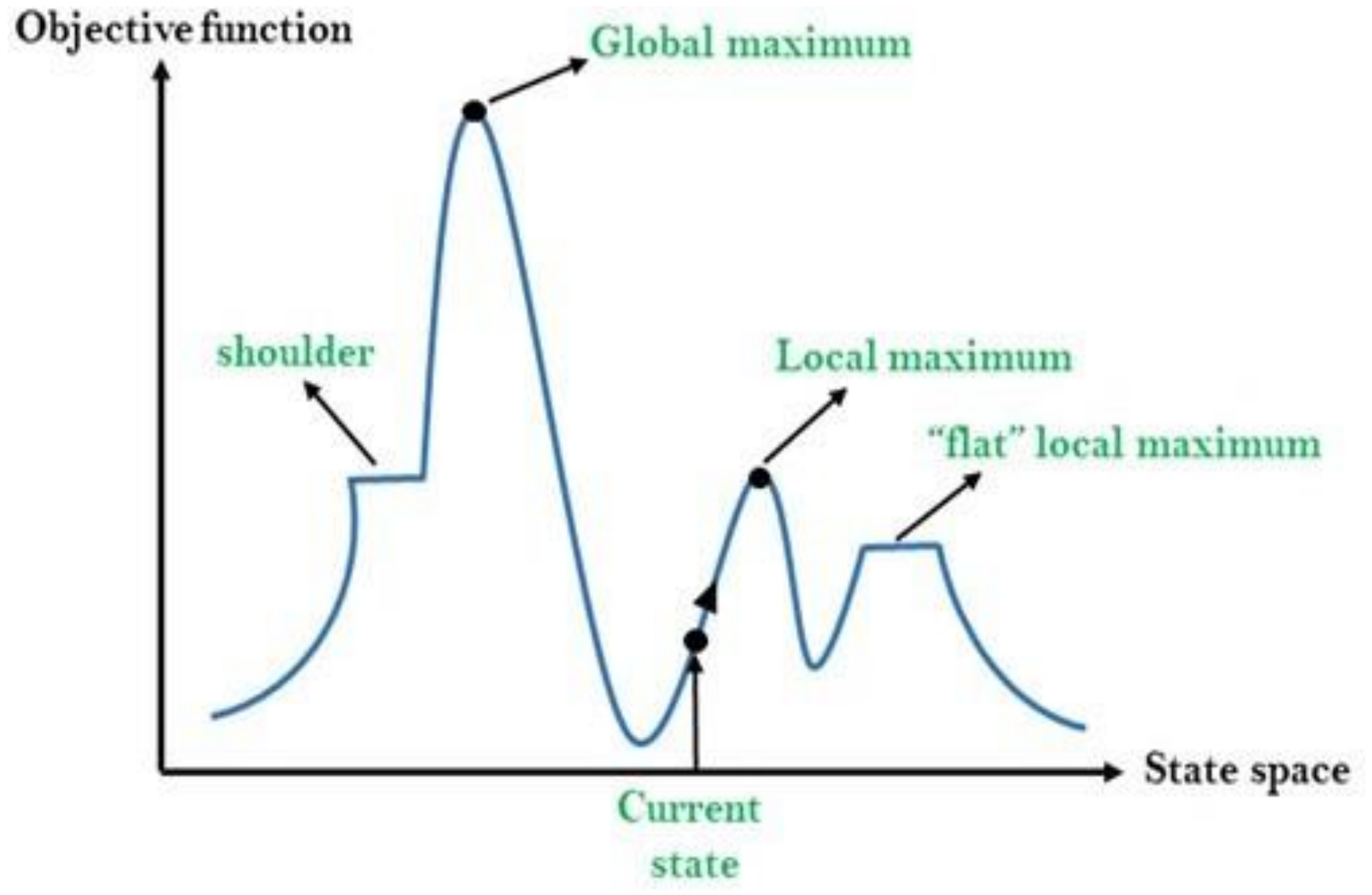
Benefits:

1. uses little memory – a constant amount for current state and some information or infinite
2. can find reasonable solutions in large (continuous) state spaces
   - where systematic algorithms are unsuitable

Local Search

- **State space landscape has two axes**
  - location (defined by states)
  - Elevation or height (defined by objective function or by the value of heuristic cost function)
  - In this figure, the cost refers to global minima and the objective function refers to global maxima(profit e.g.)

Hill Climbing – State space Diagram



Different regions in the state space

- **Local Maximum:** Local maximum is a state which is better than its neighbor states, but there is also another state which is higher than it.
- **Global Maximum:** Global maximum is the best possible state of state space landscape. It has the highest value of objective function.
- **Current state:** It is a state in a landscape diagram where an agent is currently present.
- **Flat local maximum:** It is a flat space in the landscape where all the neighbor states of current states have the same value.
- **Shoulder:** It is a plateau region which has an uphill edge.

Local Search: Hill Climbing

Hill Climbing is an iterative algorithm that starts with an arbitrary solution to a problem, then attempts to find a better solution by incrementally changing a single element of the solution. If the change produces a better solution, an incremental change is made to the new solution, repeating until no further improvements can be found.

- In **simple hill climbing**, the first closer node is chosen, whereas in steepest ascent hill climbing all successors are compared and the closest to the solution is chosen. Both forms fail if there is no closer node, which may happen if there are local maxima in the search space which are not solutions.
- **Steepest ascent hill climbing** is similar to best- first search, which tries all possible extensions of the current path instead of only one.
- **Stochastic hill climbing** does not examine all neighbors before deciding how to move. Rather, it selects a neighbor at random, and decides (based on the amount of improvement in that neighbor) whether to move to that neighbor or to examine another.

Simple Hill Climbing

•It is the simplest form of the Hill Climbing Algorithm. It only takes into account the neighboring node for its operation.

•If the neighboring node is better than the current node then it sets the neighbor node as the current node.

•The algorithm checks only one neighbor at a time. Following are a few of the key feature of the Simple Hill Climbing Algorithm.

•Since it needs low computation power, it consumes lesser time

•The algorithm results in sub-optimal solutions and at times the solution is not guaranteed.

Algorithm

1. Examine the current state, Return success if it is a goal state
2. Continue the Loop until a new solution is found or no operators are left to apply
3. Apply the operator to the node in the current state
4. Check for the new state
   - If Current State = Goal State, Return success and exit
   - Else if New state is better than current state then Goto New state
   - Else return to step 2
5. Exit

Steepest-Ascent Hill Climbing

- Steepest-Ascent hill climbing is an advanced form of simple Hill Climbing Algorithm.
- It runs through all the nearest neighbor nodes and selects the node which is nearest to the goal state.
- The algorithm requires more computation power than Simple Hill Climbing Algorithm as it searches through multiple neighbors at once.

Algorithm:

1. Examine the current state, Return success if it is a goal state
2. Continue the Loop until a new solution is found or no operators are left to apply

Let ‘Temp’ be a state such that any successor of the current state will have a higher value for the objective function. For all operators that can be applied to the current state

- Apply the operator to create a new state
- Examine new state
- If Current State = Goal State, Return success and exit
- Else if New state is better than Temp then set this state as Temp
- If Temp is better than Current State set Current state to Target

Stochastic Hill Climbing

- Stochastic Hill Climbing doesn’t look at all its neighboring nodes to check if it is better than the current node instead, it randomly selects one neighboring node, and based on the pre-defined criteria it decides whether to go to the neighboring node or select an alternate node.

Algorithm:

- Evaluate the initial state. If it is a goal state then stop and return success. Otherwise, make the initial state the current state.
- Repeat these steps until a solution is found or the current state does not change.
  - Select a state that has not been yet applied to the current state.
  - Apply the successor function to the current state and generate all the neighbor states.
  - Among the generated neighbor states which are better than the current state choose a state randomly (or based on some probability function).
  - If the chosen state is the goal state, then return success, else make it the current state and repeat step 2 of the second point.
- Exit from the function.

Simulated Annealing

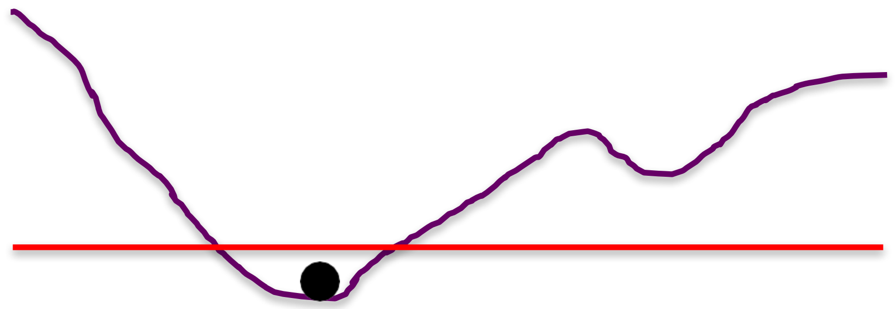
written to find minimum value solutions

```
function SIMULATED-ANNEALING( problem, schedule) return a solution state
    input: problem, a problem
           schedule, a mapping from time to temperature
    local variables: current, a node.
                     next, a node.
                     T, a "temperature" controlling the prob. of downward steps

    current ← MAKE-NODE(INITIAL-STATE[problem])
    for t ← 1 to ∞ do
        T ← schedule[t]
        if T = 0 then return current
        next ← a randomly selected successor of current
        ΔE ← VALUE[next] - VALUE[current]
        if ΔE<0   then current ← next
        else      current ← next only with probability $e^{\Delta E/T}$
```

Physical Interpretation of Simulated Annealing

- A Physical Analogy:
  - Imagine letting a ball roll downhill on the function surface
  - Now shake the surface, while the ball rolls,
  - Gradually reducing the amount of shaking

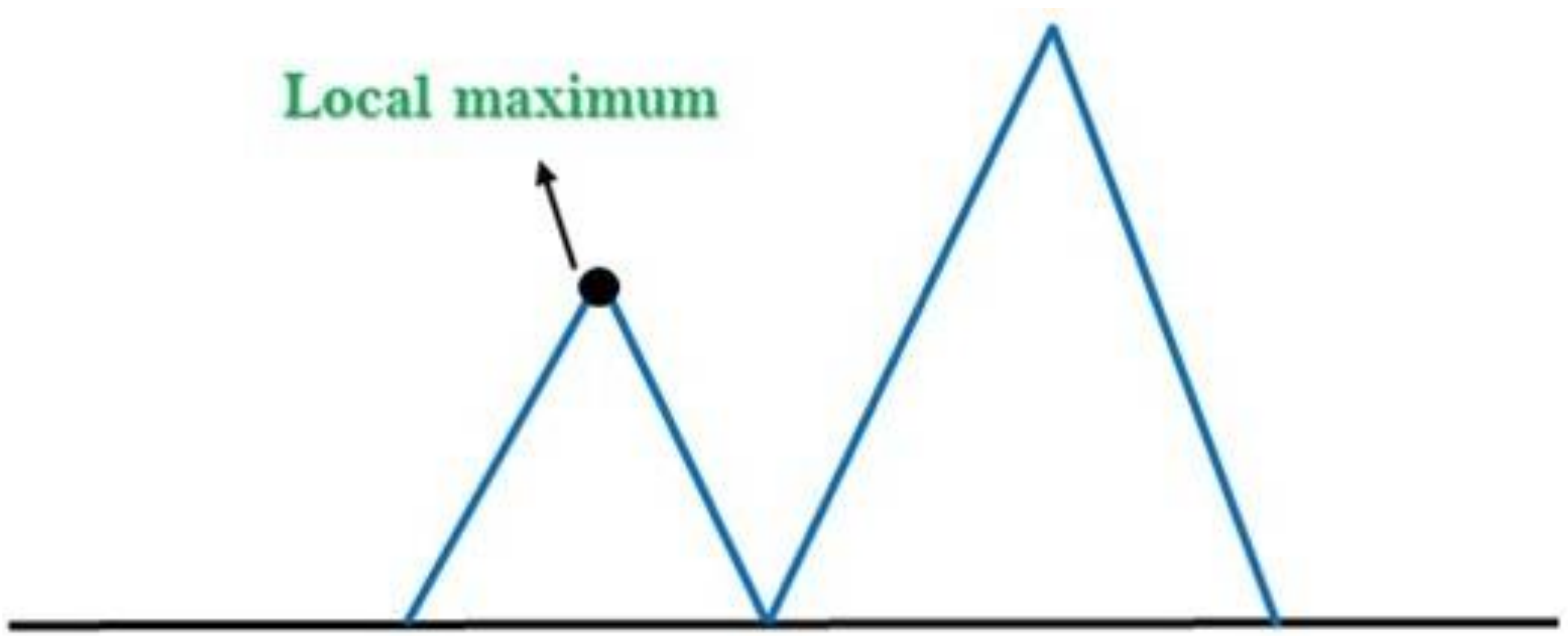


- Annealing = physical process of cooling a liquid → frozen
  - simulated annealing:
    - free variables are like particles
    - seek "low energy" (high quality) configuration
    - slowly reducing temp. T with particles moving around randomly

Problems in hill Climbing Algorithm

- 1. Local Maximum:
  - A local maximum is a peak state in the landscape which is better than each of its neighboring states, but there is another state also present which is higher than the local maximum.
  - Solution: Backtracking technique can be a solution of the local maximum in state space landscape. Create a list of the promising path so that the algorithm can backtrack the search space and explore other paths as well.

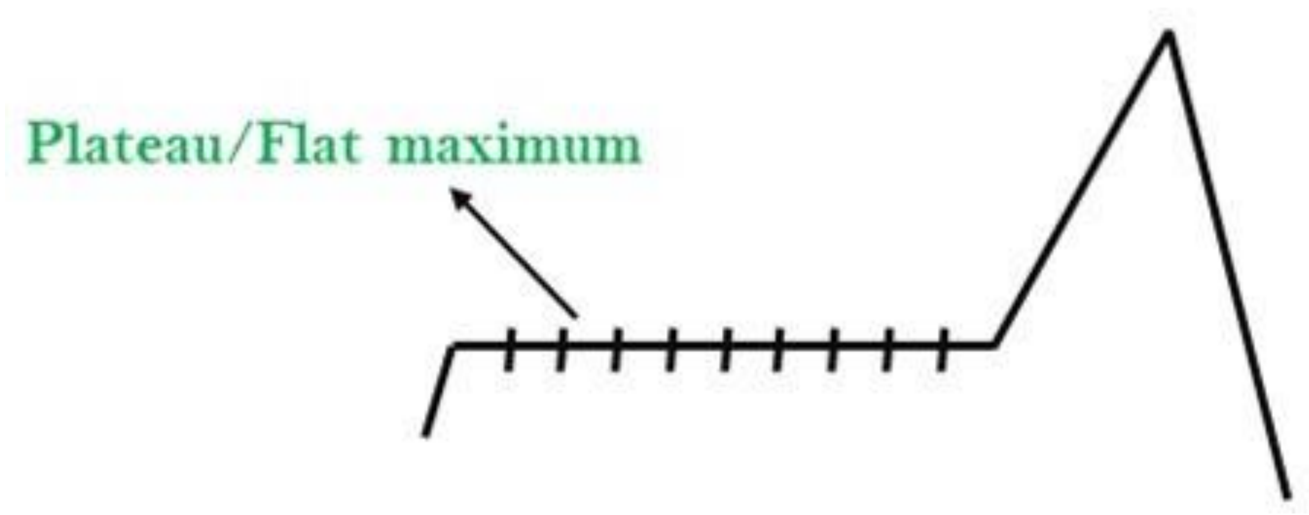
Problems in hill Climbing Algorithm



Problems in hill Climbing Algorithm

- 2. Plateau: A plateau is the flat area of the search space in which all the neighbor states of the current state contains the same value, because of this algorithm does not find any best direction to move. A hill-climbing search might be lost in the plateau area.
- Solution: The solution for the plateau is to take big steps or very little steps while searching, to solve the problem. Randomly select a state which is far away from the current state so it is possible that the algorithm could find non-plateau region.

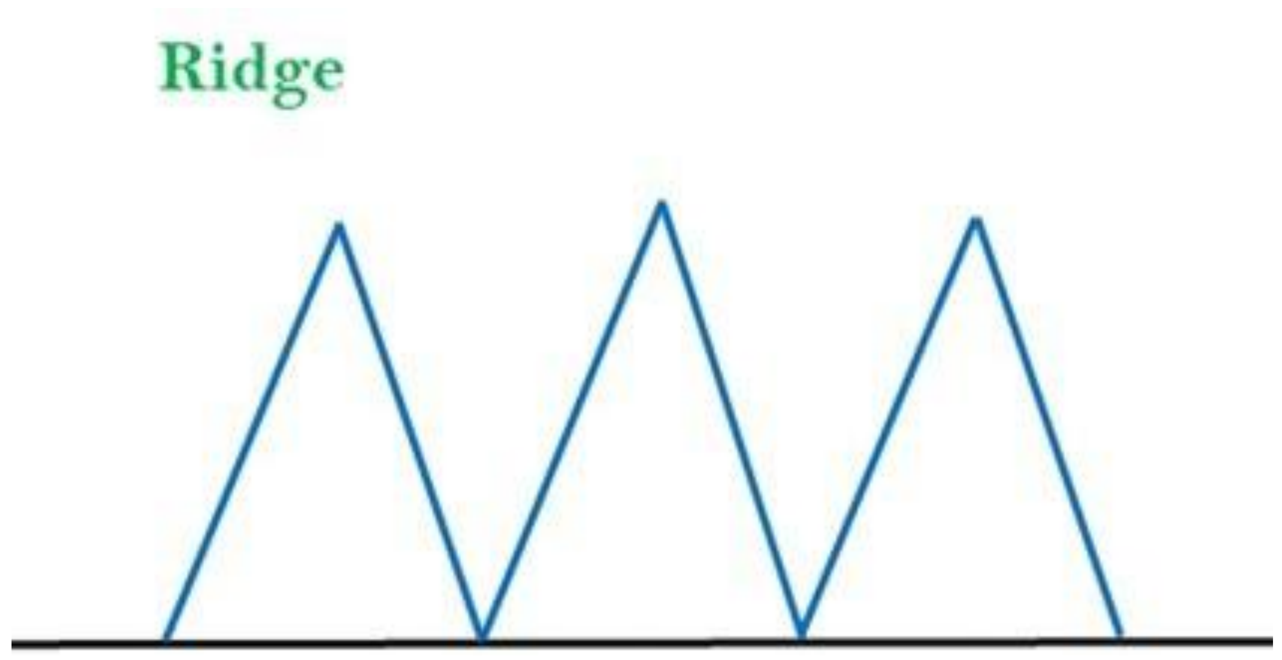
Problems in hill Climbing Algorithm



Problems in hill Climbing Algorithm

- 3. Ridges: A ridge is a special form of the local maximum. It has an area which is higher than its surrounding areas, but itself has a slope, and cannot be reached in a single move.
- Solution: With the use of bidirectional search, or by moving in different directions, we can improve this problem.

Problems in hill Climbing Algorithm



Advantage of Hill Climbing Algorithm

- Hill Climbing is very useful in routing-related problems like Travelling Salesmen Problem, Job Scheduling, Chip Designing, and Portfolio Management
- It is good in solving the optimization problem while using only limited computation power