Informed Search Algorithms

- Informed search algorithm contains an array of knowledge such as how far we are from the goal, path cost, how to reach to goal node, etc. This knowledge help agents to explore less to the search space and find more efficiently the goal node.
- Informed search algorithm uses the idea of heuristic, so it is also called Heuristic search.
- **Heuristics function:** Heuristic is a function which is used in Informed Search, and it finds the most promising path. **It takes the current state of the agent as its input and produces the estimation of how close agent is from the goal.**
- The heuristic method, however, might not always give the best solution, but it guaranteed to find a good solution in reasonable time.
- The value of the heuristic function is always **positive**.

- Heuristic function is given as:

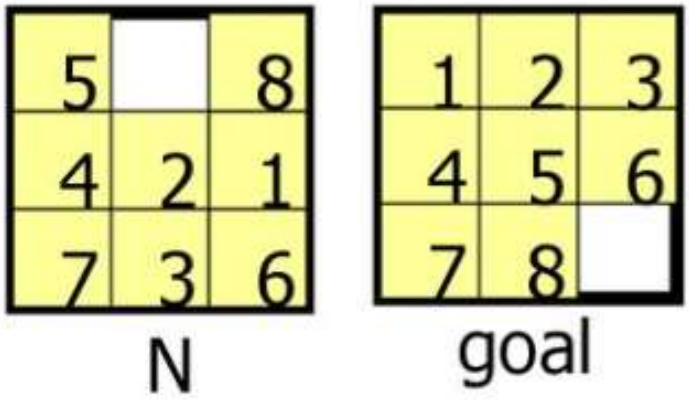
$$h(n) <= h^*(n)$$

- Here h(n) is heuristic cost, and h*(n) is the estimated cost. Hence heuristic cost should be less than or equal to the estimated cost.

Pure Heuristic Search

- Pure heuristic search is the simplest form of heuristic search algorithms. It expands nodes based on their heuristic value h(n). It maintains two lists, **OPEN and CLOSED list**. In the CLOSED list, it places those nodes which have already expanded and in the OPEN list, it places nodes which have yet not been expanded.

- On each iteration, each node n with the **lowest heuristic** value is expanded and generates all its successors and n is placed to the closed list. The algorithm continues unit a goal state is found.

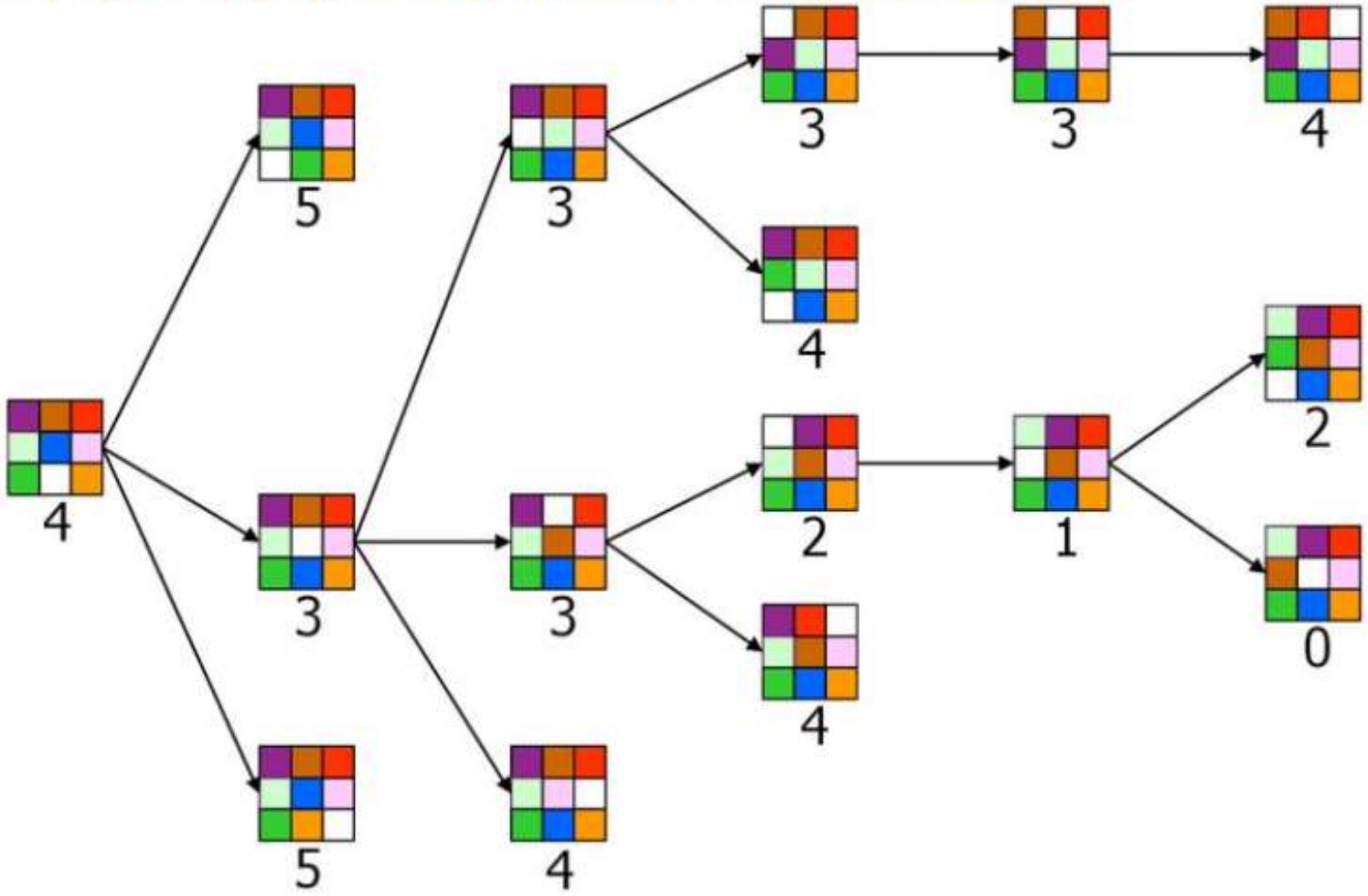


h(N) = number of misplaced tiles
= 6

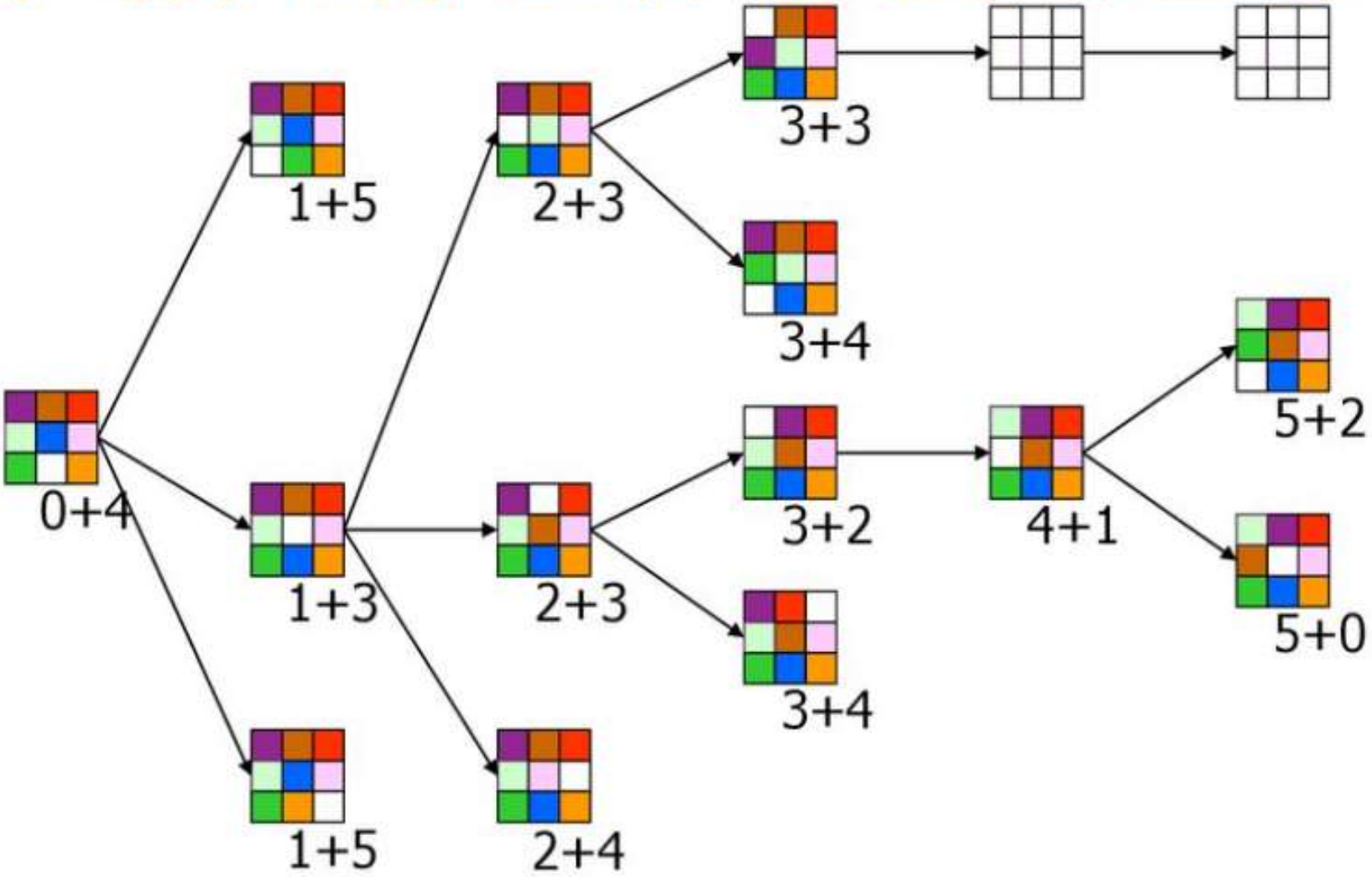
h(N) = sum of the distances of every tile to its goal position
= 2 + 3 + 0 + 1 + 3 + 0 + 3 + 1
= 13

8-PUZZLE

f(N) = h(N) = number of misplaced tiles



$f(N) = g(N) + h(N)$ with h(N) = number of misplaced tiles

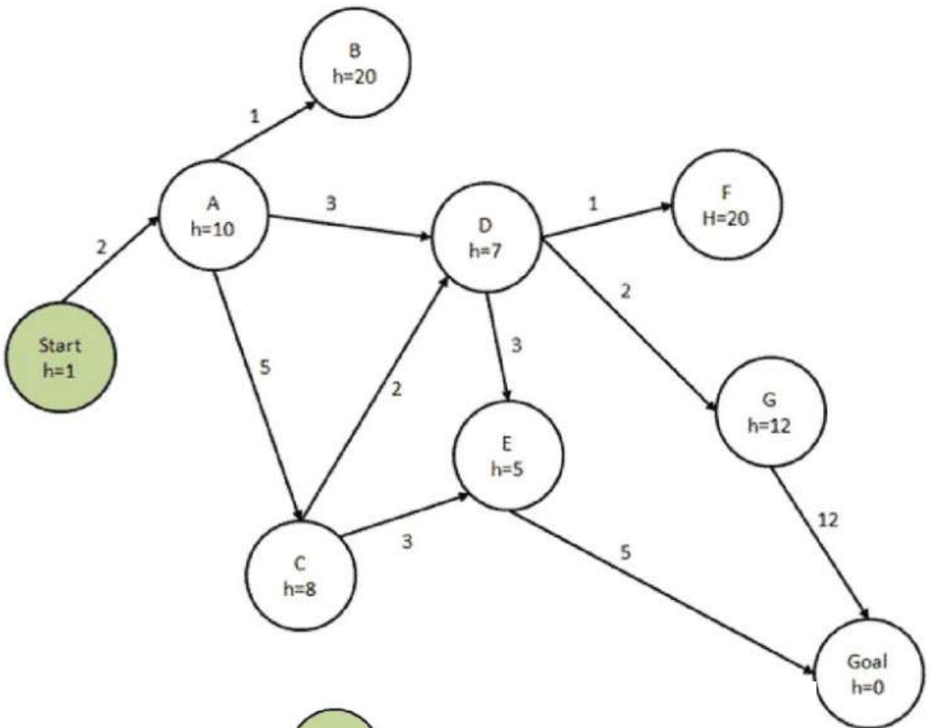


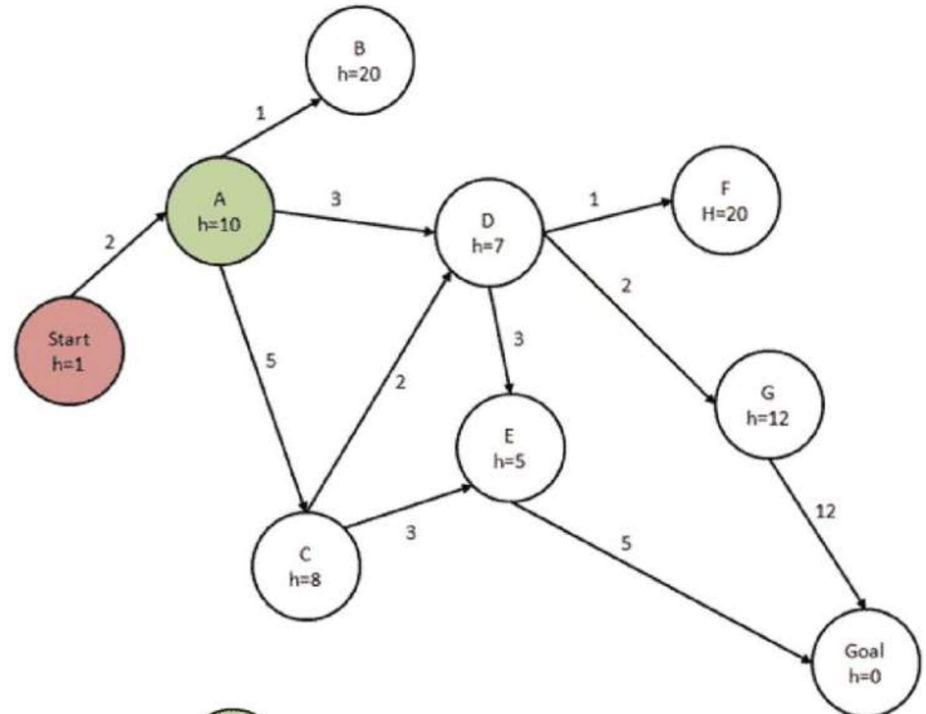
- In the informed search we will discuss two main algorithms

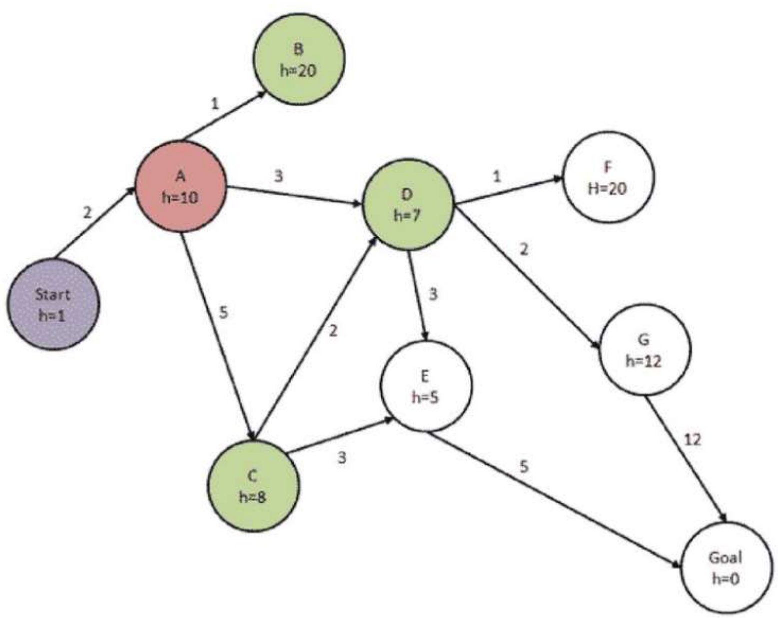
1) Best First Search Algorithm(Greedy search)
2) A* Search Algorithm

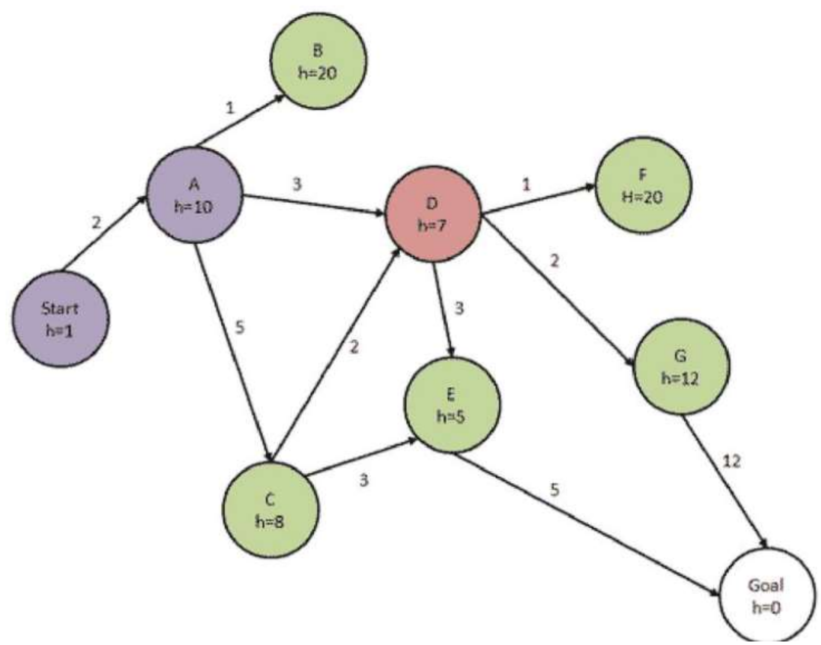
Best-first Search Algorithm (Greedy Search):

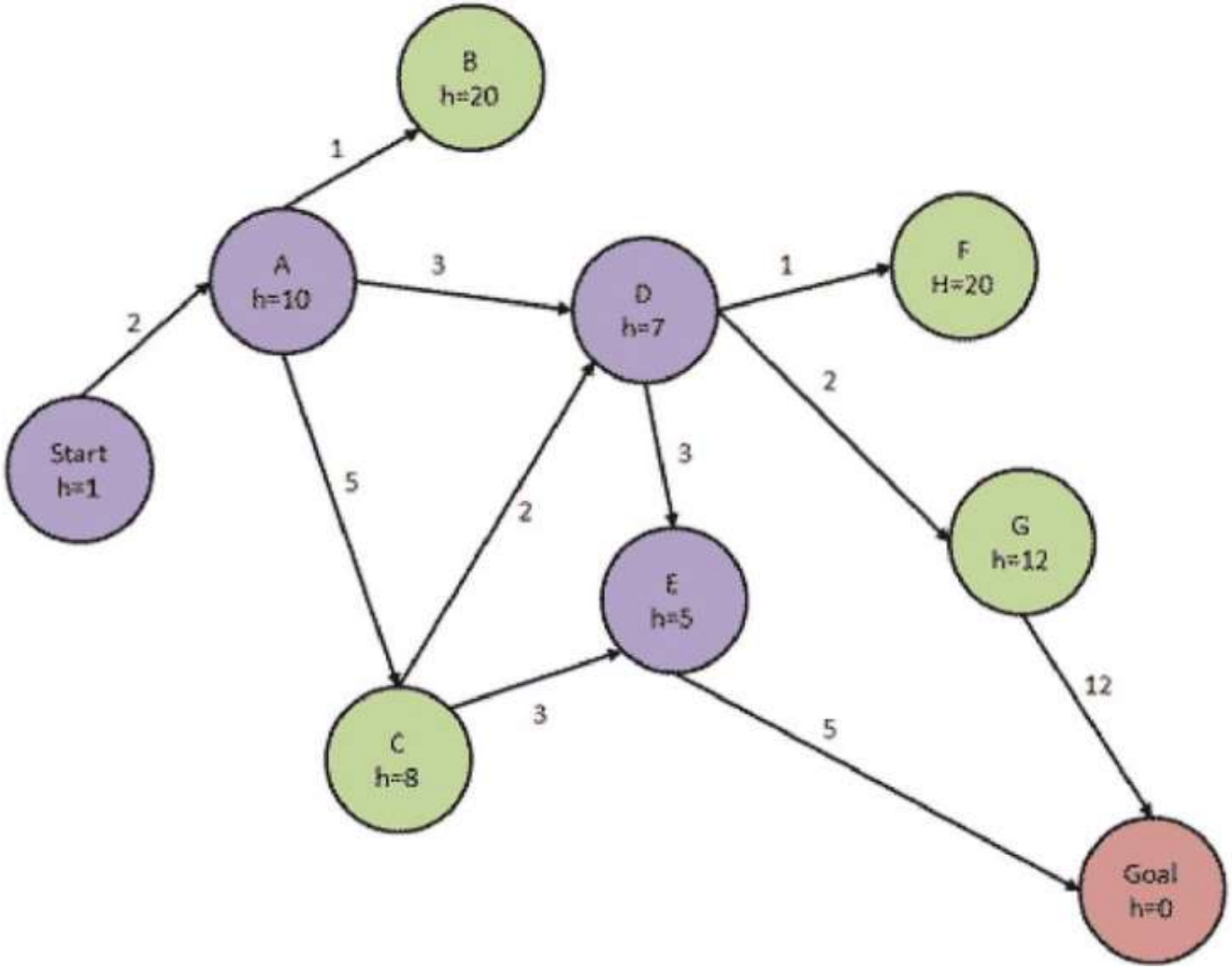
- Greedy best-first search algorithm always selects the path which appears best at that moment.
- It is the combination of **depth-first search and breadth-first** search algorithms. It uses the heuristic function and search.
- In the best first search algorithm, we expand the node **which is closest to the goal node**


B
h=20
A
h=10
D
h=7
F
H=20
Start
h=1
G
h=12
E
h=5
C
h=8
Goal
h=0

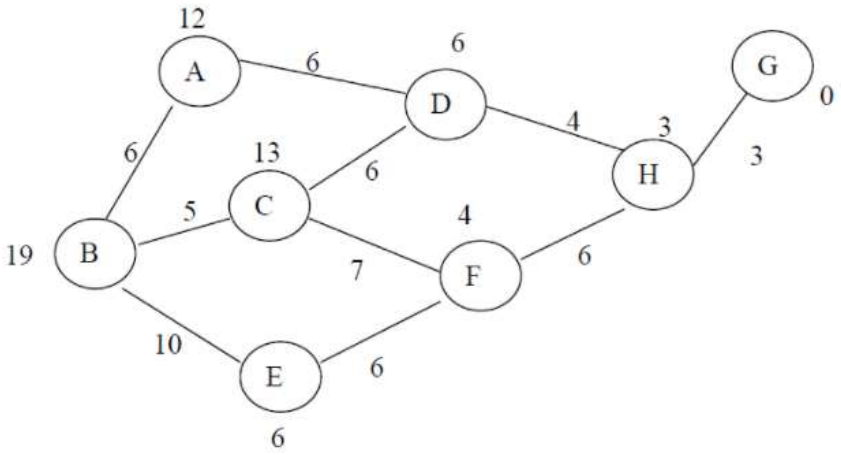

B
h=20
A
h=10
D
h=7
F
H=20
Start
h=1
G
h=12
E
h=5
C
h=8
Goal
h=0

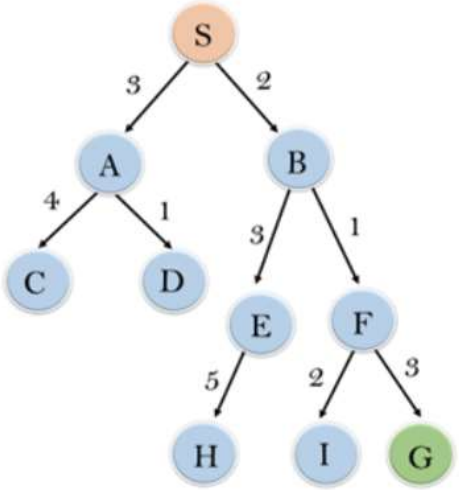

B
h=20
A
h=10
D
h=7
F
H=20
Start
h=1
G
h=12
E
h=5
C
h=8
Goal
h=0


B
h=20
A
h=10
D
h=7
F
H=20
Start
h=1
G
h=12
E
h=5
C
h=8
Goal
h=0



Start → A → D → E → Goal


12
A
6
6
D
G
0
4
3
3
6
13
6
H
5
C
4
19
B
7
F
6
10
E
6
6



node	H (n)
A	12
B	4
C	7
D	3
E	8
F	2
H	4
I	9
S	13
G	0

Best First Search Algorithm:

Step 1: Place the starting node into the OPEN list.

Step 2: If the OPEN list is empty, Stop and return failure.

Step 3: Remove the node n, from the OPEN list which has the lowest value of h(n), and places it in the CLOSED list.

Step 4: Expand the node n, and generate the successors of node n.

Step 5: Check each successor of node n, and find whether any node is a goal node or not. If any successor node is goal node, then return success and terminate the search, else proceed to Step 6.

Step 6: For each successor node, algorithm checks for evaluation function f(n), and then check if the node has been in either OPEN or CLOSED list. If the node has not been in both list, then add it to the OPEN list.

Step 7: Return to Step 2.

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
```

Time Complexity: The worst case time complexity of Greedy best first search is $O(b^m)$.

Space Complexity: The worst case space complexity of Greedy best first search is $O(b^m)$. Where, m is the maximum depth of the search space.

Complete: Greedy best-first search is also incomplete, even if the given state space is finite.

Optimal: Greedy best first search algorithm is not optimal.

Time Complexity

Space Complexity

Complete

Optimal

Advantages and Disadvantages of Best First Search

Advantages:

- Best first search can switch between BFS and DFS by gaining the advantages of both the algorithms.
- This algorithm is more efficient than BFS and DFS algorithms.

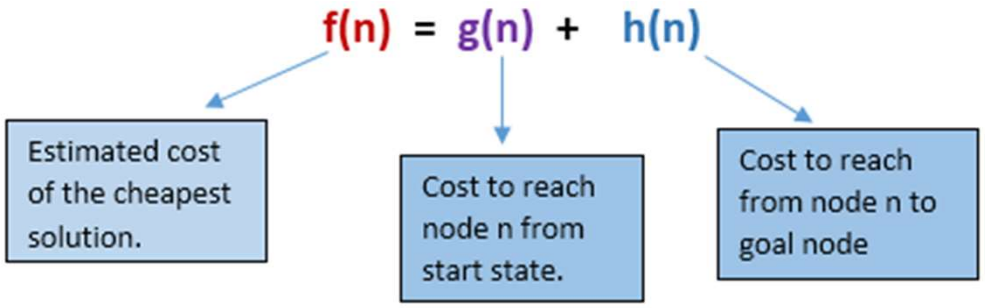
Disadvantages:

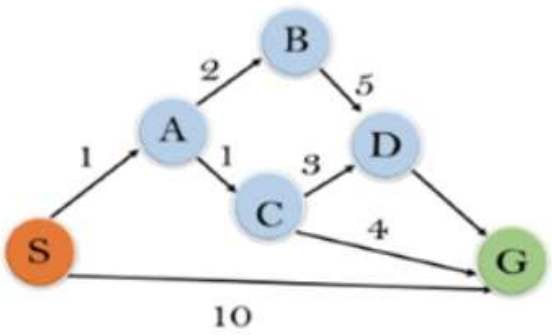
- It can behave as an unguided depth-first search in the worst case scenario.
- It can get stuck in a loop as DFS.
- This algorithm is not optimal.

A Search Algorithm*

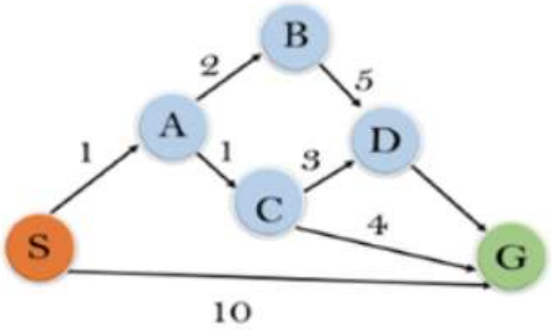
- A* search is the most commonly known form of best-first search.
- It uses **heuristic function h(n),** and **cost to reach the node n from the start state g(n)**.
- A* algorithm is similar to UCS except that it uses g(n)+h(n) instead of g(n).
- It has combined features of UCS and greedy best-first search, by which it solve the problem efficiently.
- A* search algorithm finds the shortest path through the search space using the heuristic function.
- This search algorithm expands less search tree and provides **optimal result** faster.

- In A* search algorithm, we use search **heuristic** as well as the **cost to reach** the node. Hence we can combine both costs as following, and this sum is called as a **fitness number**.

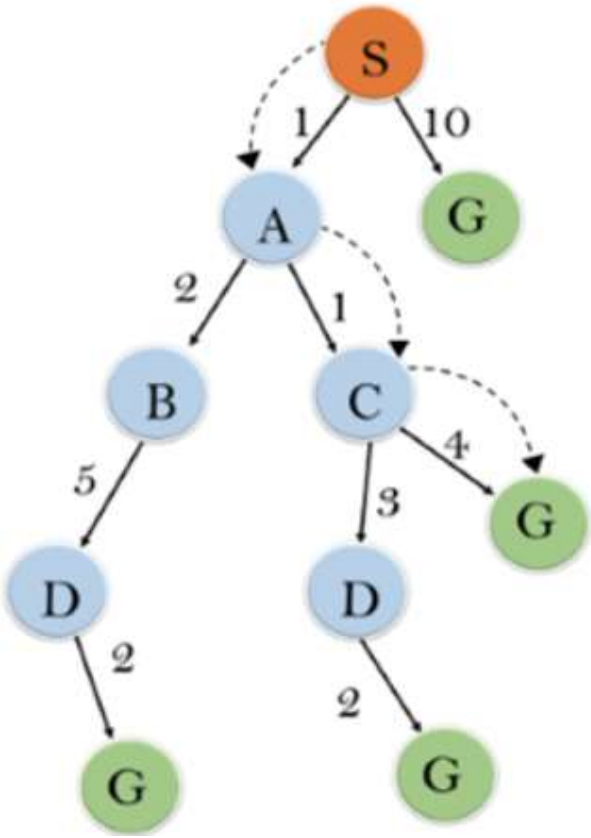




State	h(n)
S	5
A	3
B	4
C	2
D	6
G	0



State	h(n)
S	5
A	3
B	4
C	2
D	6
G	0



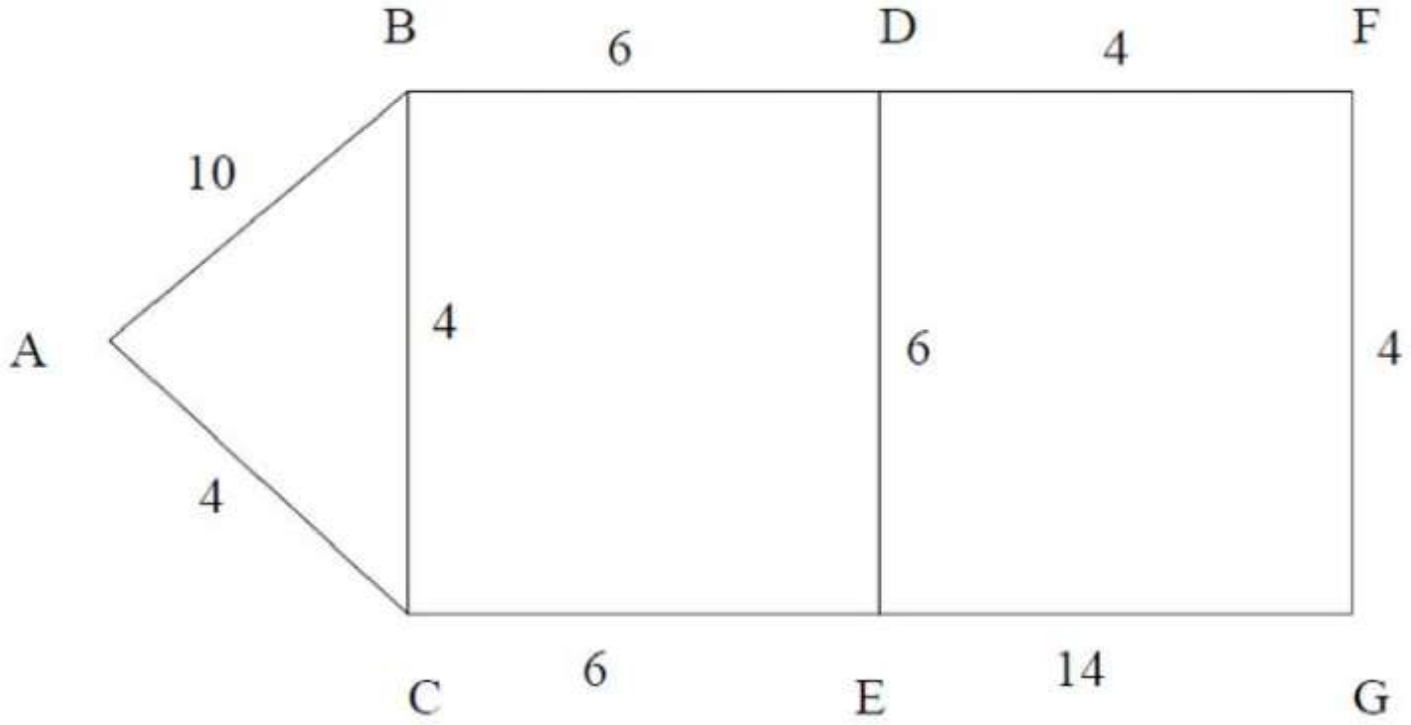
Initialization: {(S, 5)}

Iteration1: {(S--> A, 4), (S-->G, 10)}

Iteration2: {(S--> A-->C, 4), (S--> A-->B, 7), (S-->G, 10)}

Iteration3: {(S--> A-->C--->G, 6), (S--> A-->C--->D, 11), (S--> A-->B, 7), (S-->G, 10)}

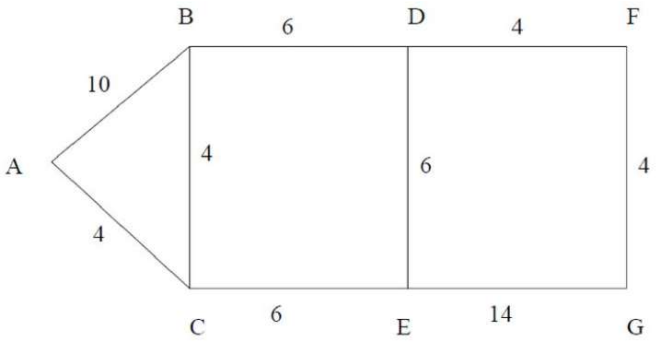
Iteration 4 will give the final result, as **S--->A--->C--->G** it provides the optimal path with cost 6.



Assume h(n)=

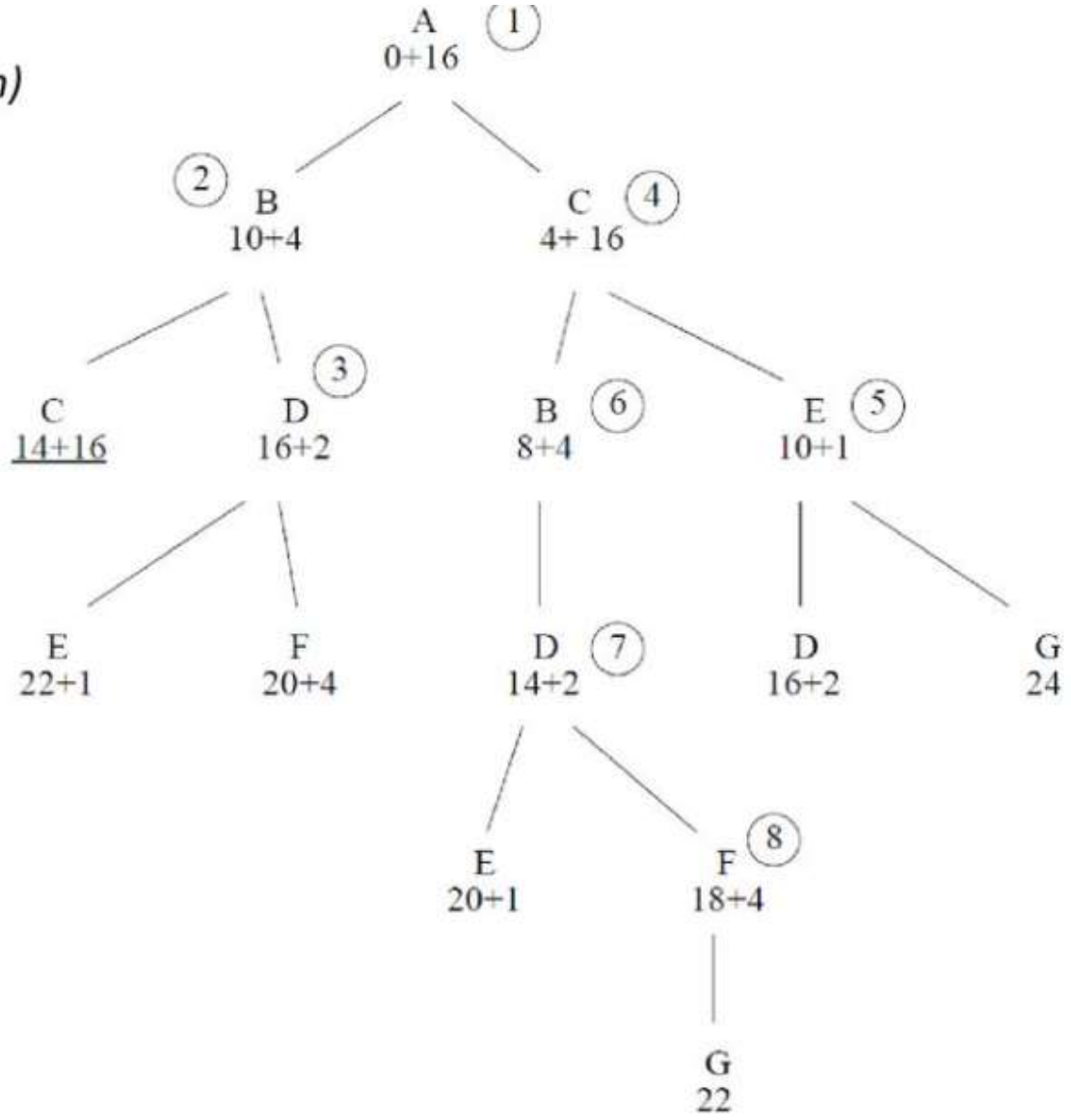
From A	16
From B	4
From C	16
From D	2
From E	1
From F	4

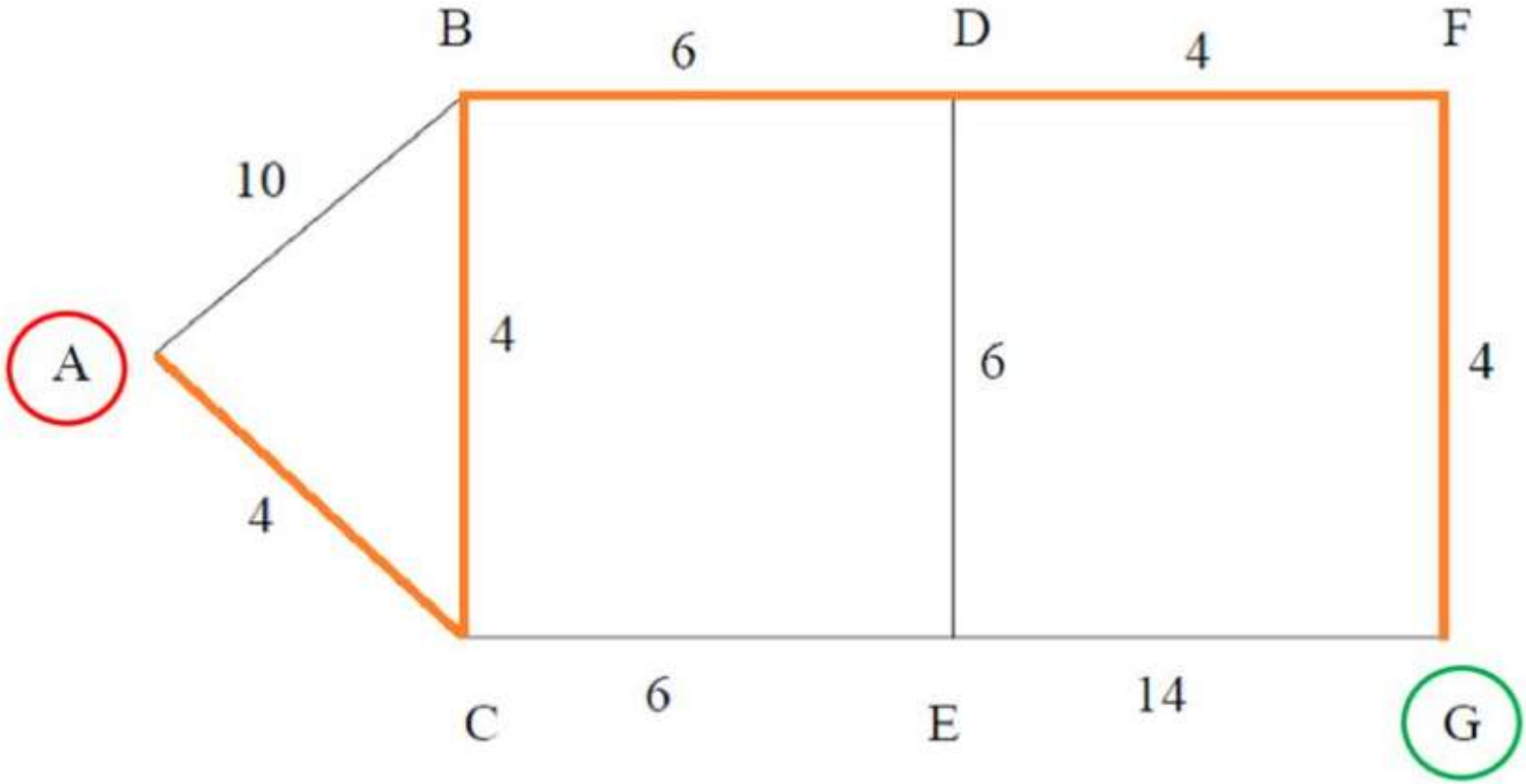
$$f(n) = g(n) + h(n)$$

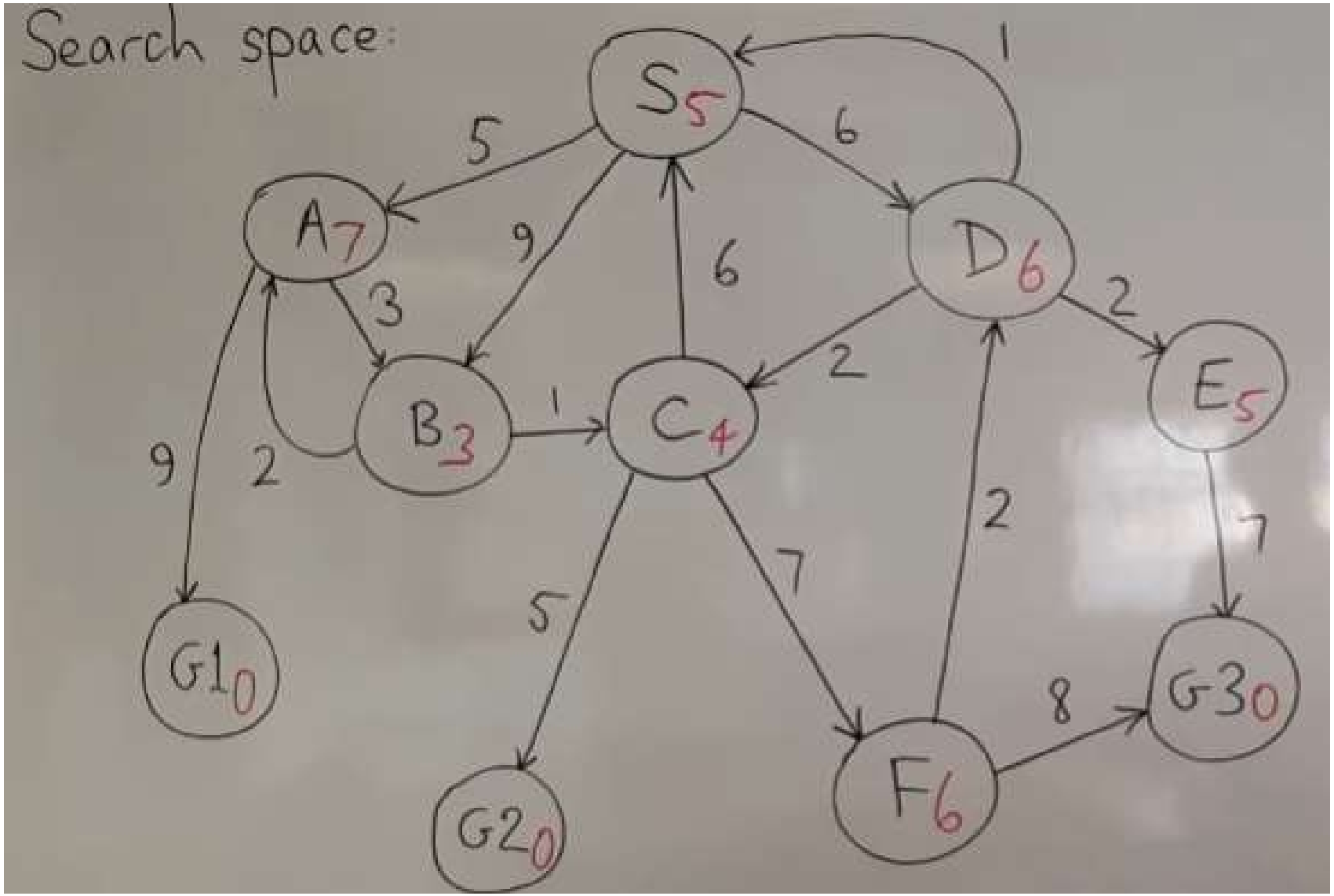


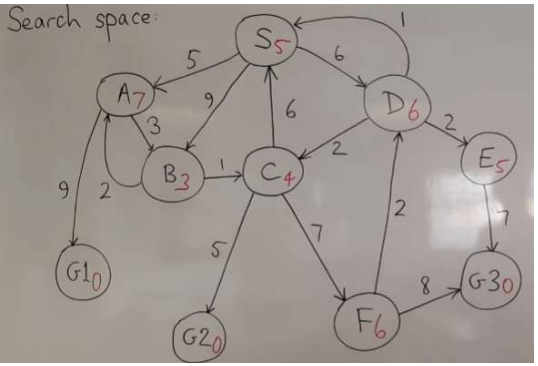
Assume h(n)=

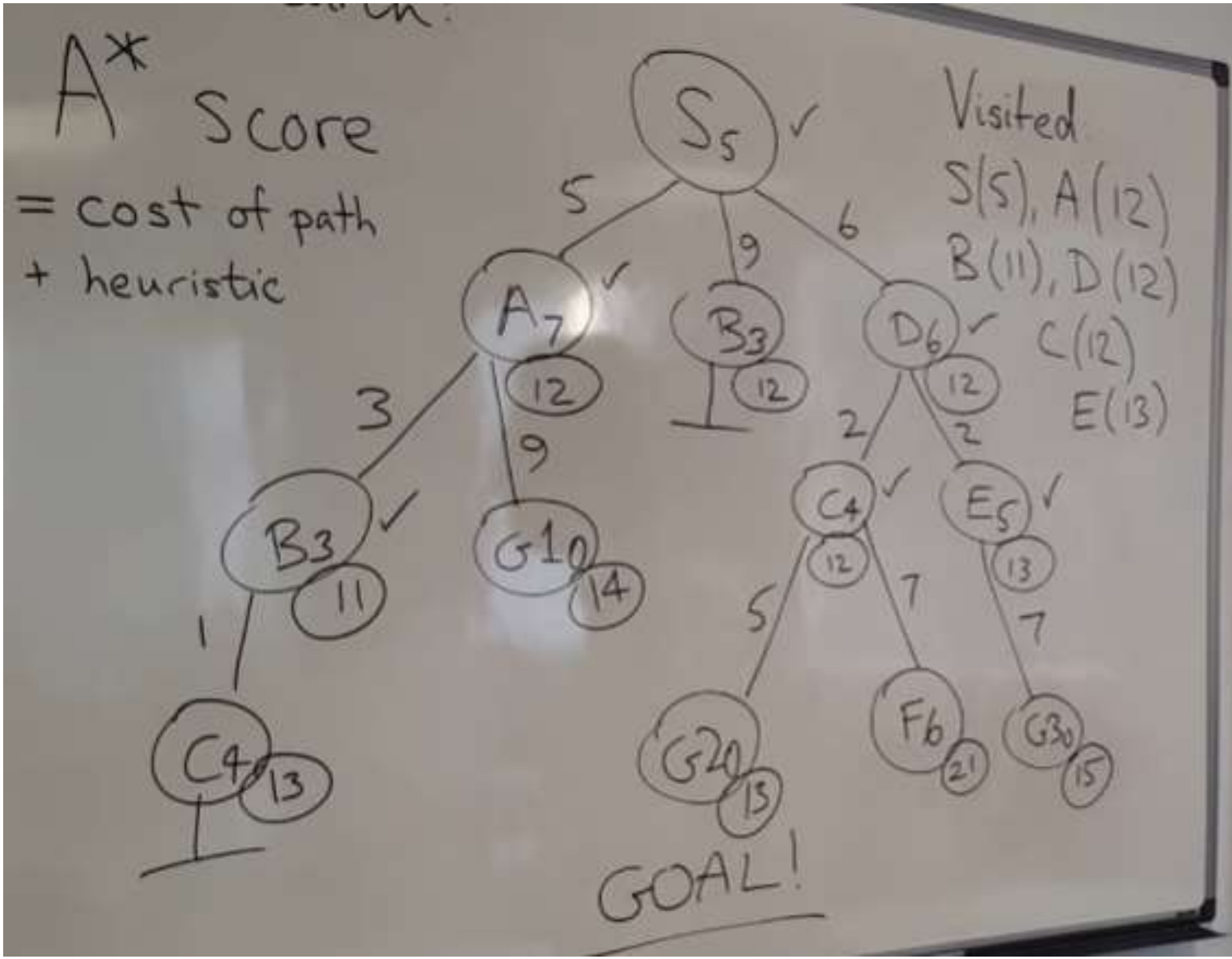
From A	16
From B	4
From C	16
From D	2
From E	1
From F	4

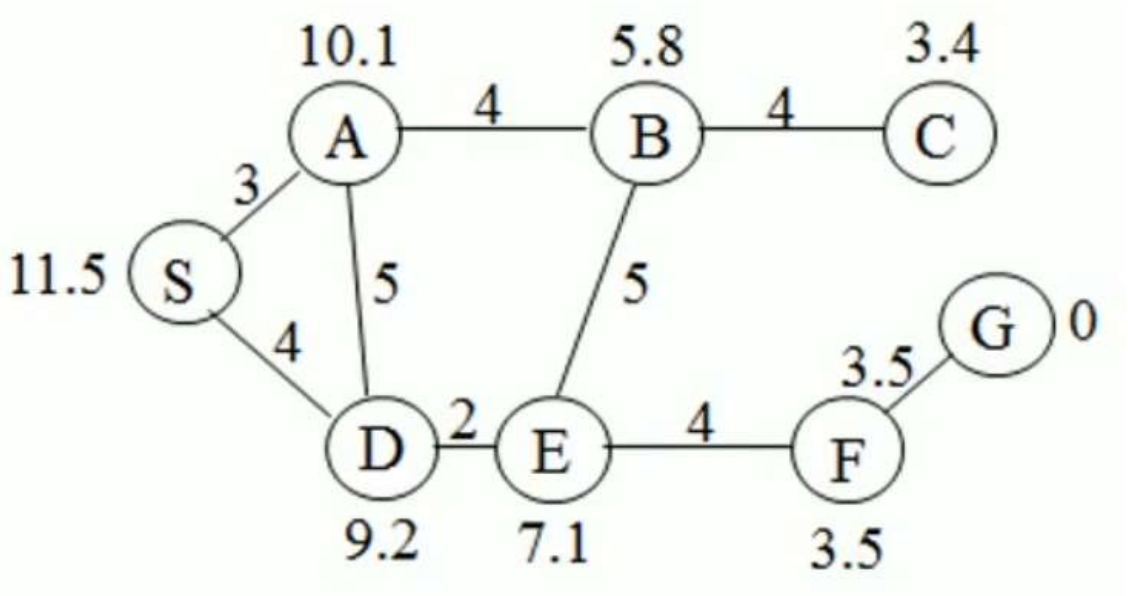


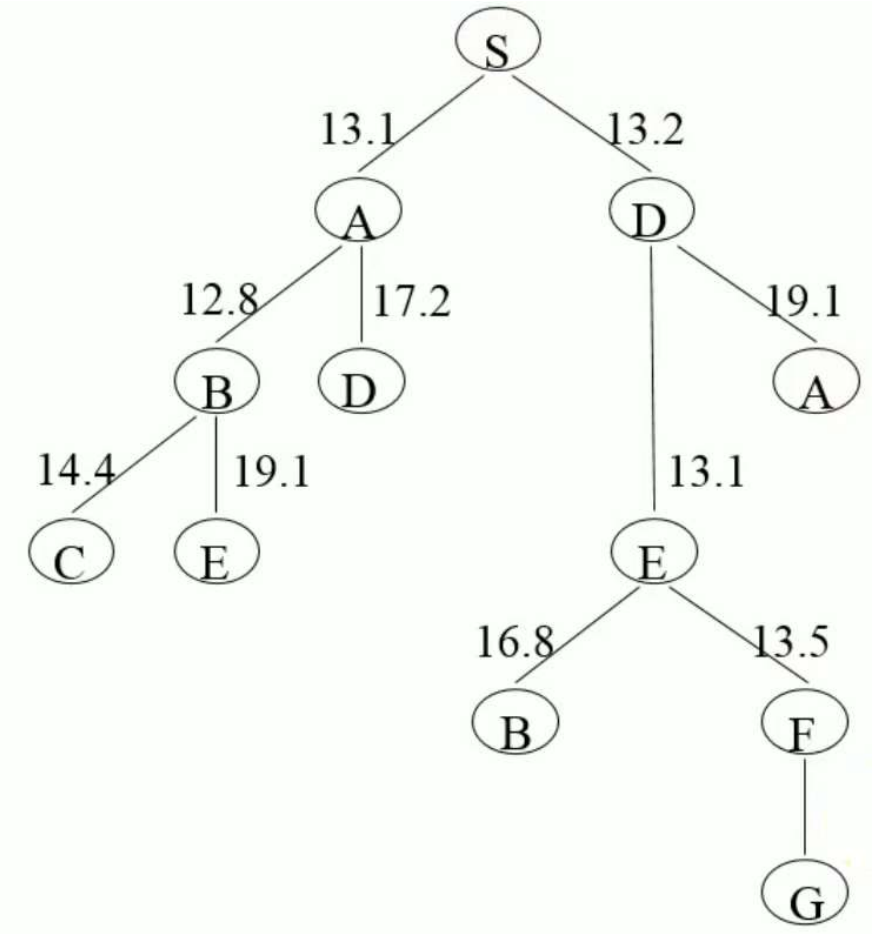

B
6
D
4
F
10
A
4
6
4
4
6
14
C
E
G

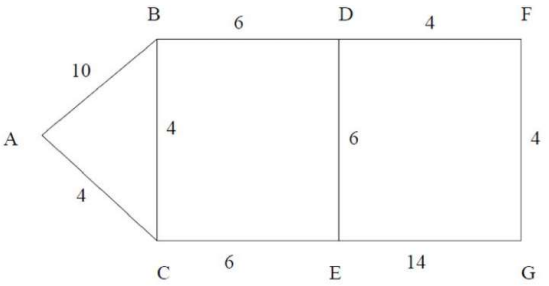

Search space:
S5
A7
D6
B3
C4
E5
G10
G20
F6
G30
1
5
6
9
6
3
2
1
9
2
2
7
5
7
8


Search space:
S5
A7
B3
C4
D6
E5
G10
G20
F6
G30


A* Score
= cost of path
+ heuristic
Visited
S(5), A(12)
B(11), D(12)
C(12)
E(13)
S5
A7
B3
D6
C4
E5
G10
G20
F6
G30
GOAL!


10.1
5.8
3.4
A
4
B
4
C
3
11.5
S
5
5
G
0
4
3.5
D
2
E
4
F
9.2
7.1
3.5


S
13.1
13.2
A
D
12.8
17.2
19.1
B
D
A
14.4
19.1
13.1
C
E
E
16.8
13.5
B
F
G



Algorithm A*

OPEN = nodes on frontier. CLOSED = expanded nodes.
OPEN = {<s, nil>}
while OPEN is not empty
    remove from OPEN the node <*n*,*p*> with minimum *f(n)*
    place <*n*,*p*> on CLOSED
    if *n* is a goal node,
        return success (path *p*)
    for each edge connecting *n* & *m* with cost *c*
        if <*m*, *q*> is on CLOSED and {*p*|*e*} is cheaper than *q*
            then remove *n* from CLOSED,
                put <*m*,{*p*|*e*}> on OPEN
        else if <*m*,*q*> is on OPEN and {*p*|*e*} is cheaper than *q*
            then replace *q* with {*p*|*e*}
        else if *m* is not on OPEN
            then put <*m*,{*p*|*e*}> on OPEN
return failure

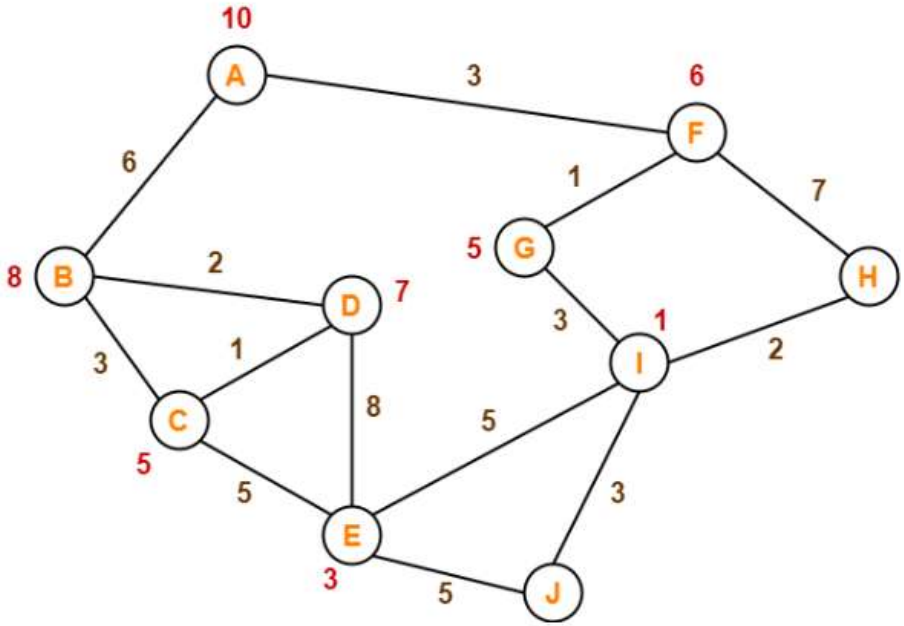
Advantages:

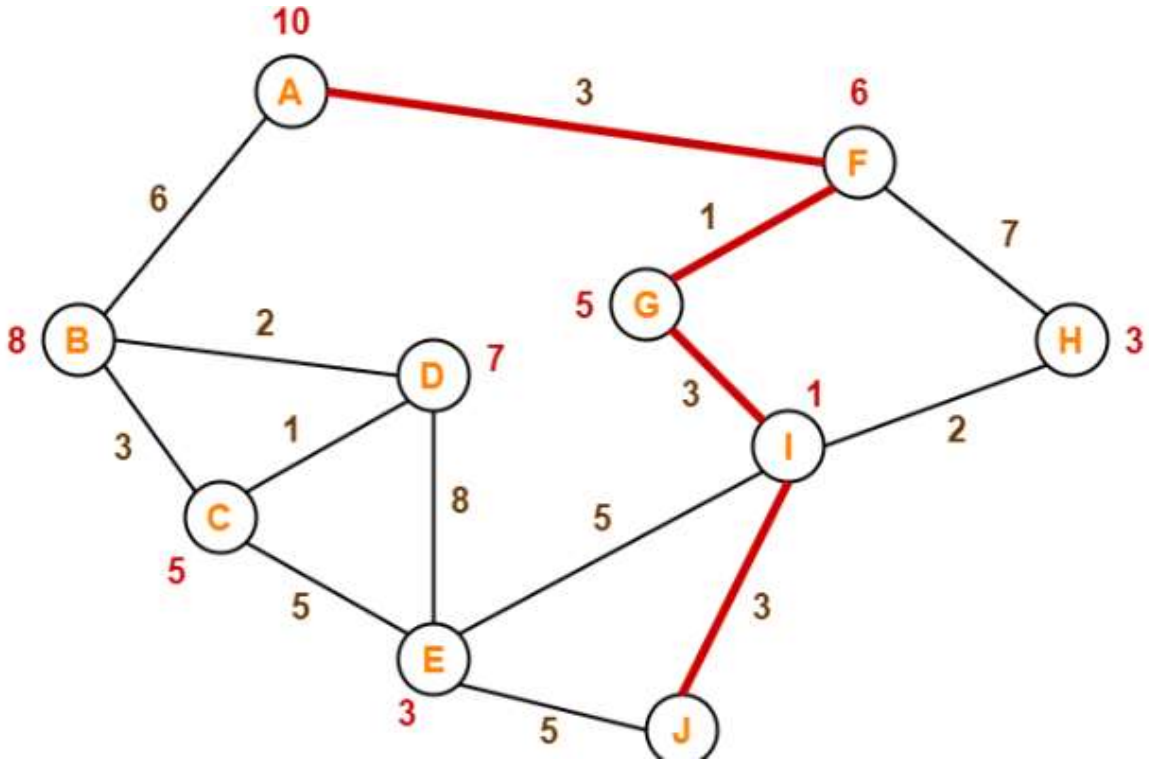
- A* search algorithm is the best algorithm than other search algorithms.
- A* search algorithm is optimal and complete.
- This algorithm can solve very complex problems.

Disadvantages:

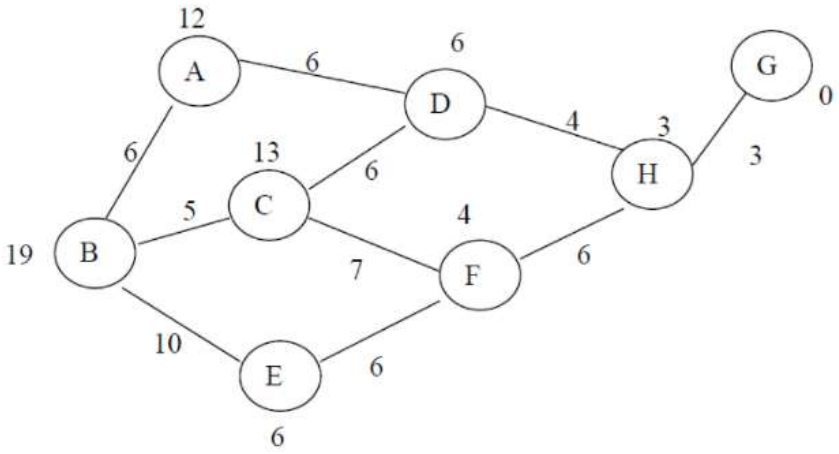
- A* search algorithm has some complexity issues.
- The main drawback of A* is memory requirement as it keeps all generated nodes in the memory, so it is not practical for various large-scale problems.

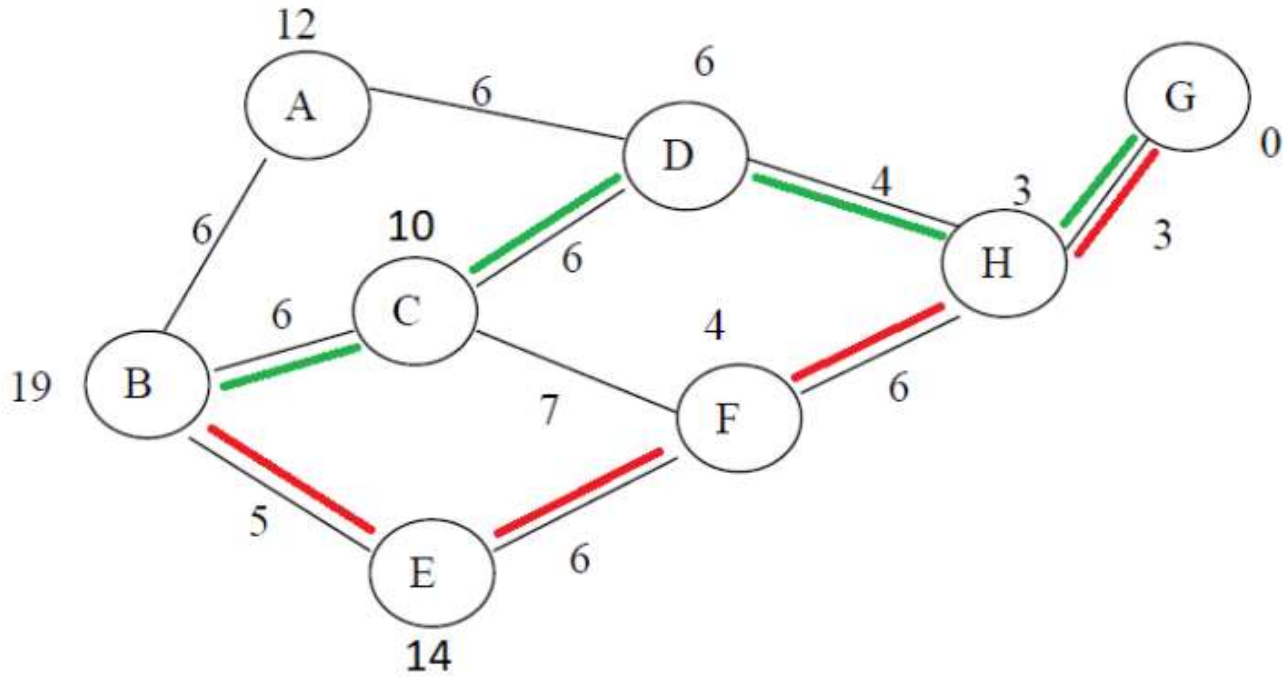
Find the most cost-effective path to reach from start state A to final state J using A* Algorithm.





Find the path by using Greedy Algorithm and Heuristic A* algorithm.





Path highlighted with red shows the path taken by Greedy Algorithm and the path highlighted with green shows the path taken by Heuristic A* algorithm.

8-Puzzle using A* Algorithm.

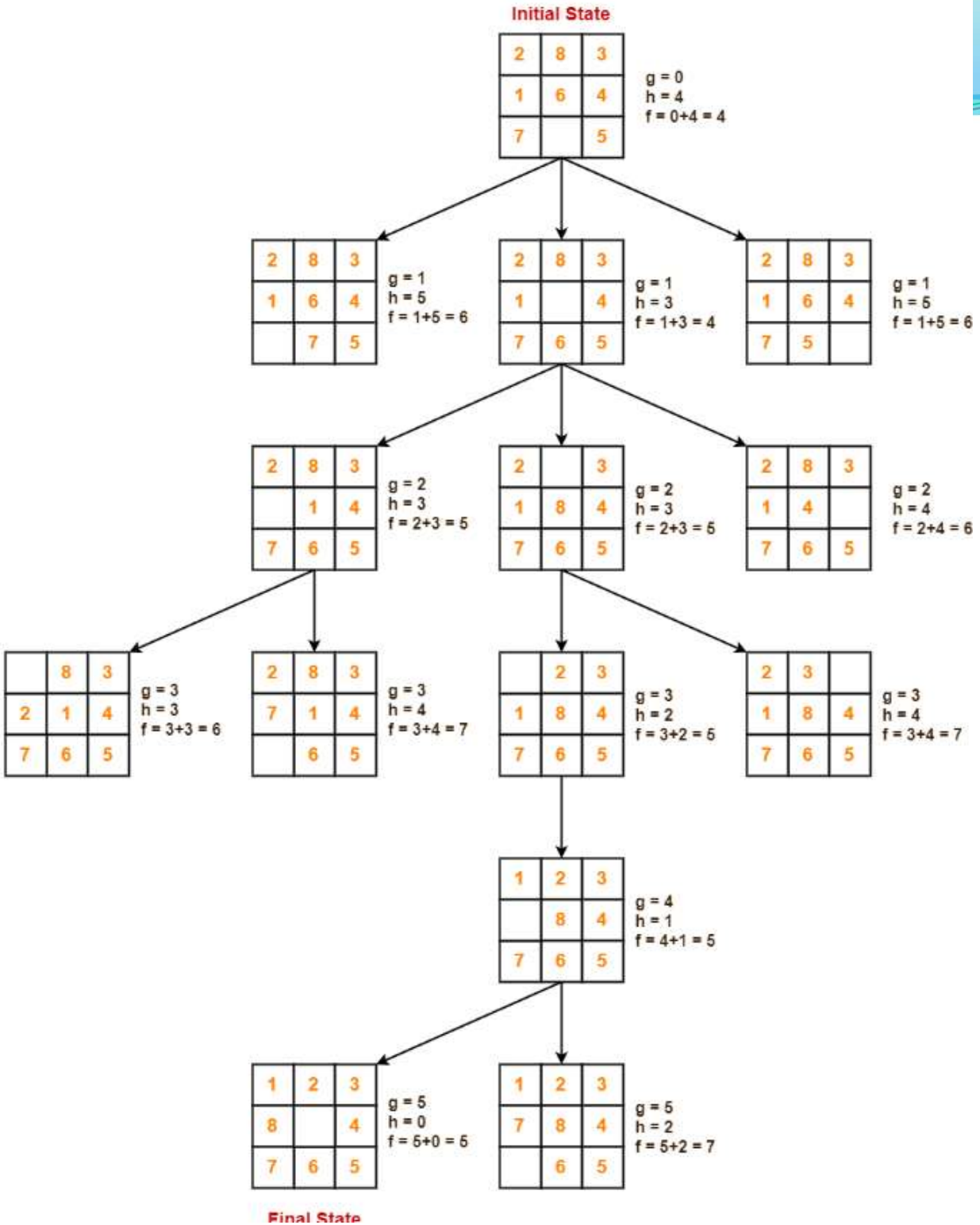
Find the most cost-effective path to reach the final state from initial state using A* Algorithm.

Initial State

Final State

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

Consider g(n) = Depth of node
h(n) = Number of misplaced tiles.


Initial State
2 8 3
1 6 4
7 5
g = 0
h = 4
f = 0+4 = 4
2 8 3
1 6 4
7 5
g = 1
h = 5
f = 1+5 = 6
2 8 3
1 4
7 6 5
g = 1
h = 3
f = 1+3 = 4
2 8 3
1 6 4
7 5
g = 1
h = 5
f = 1+5 = 6
2 8 3
1 4
7 6 5
g = 2
h = 3
f = 2+3 = 5
2 3
1 8 4
7 6 5
g = 2
h = 3
f = 2+3 = 5
2 8 3
1 4
7 6 5
g = 2
h = 4
f = 2+4 = 6
8 3
2 1 4
7 6 5
g = 3
h = 3
f = 3+3 = 6
2 8 3
7 1 4
6 5
g = 3
h = 4
f = 3+4 = 7
2 3
1 8 4
7 6 5
g = 3
h = 2
f = 3+2 = 5
2 3
1 8 4
7 6 5
g = 3
h = 4
f = 3+4 = 7
1 2 3
8 4
7 6 5
g = 4
h = 1
f = 4+1 = 5
1 2 3
8 4
7 6 5
g = 5
h = 0
f = 5+0 = 5
1 2 3
7 8 4
6 5
g = 5
h = 2
f = 5+2 = 7
Final State

Hill Climbing

- Hill climbing algorithm is a local search algorithm which continuously moves in the direction of increasing elevation/value to find the peak of the mountain or best solution to the problem.
- It terminates when it reaches a **peak value** where no neighbor has a higher value.
- It is also called **greedy local search** as it only looks to its good immediate neighbor state and not beyond that.
- Hill Climbing is mostly used when a good heuristic is available.
- In this algorithm, we **don't need to maintain and handle the search tree or graph** as it only keeps a single current state.

Features of Hill Climbing

Greedy approach: Hill-climbing algorithm search moves in the direction which optimizes the cost.

No backtracking: It does not backtrack the search space, as it does not remember the previous states.

Feedback Mechanism: The feedback from the previous computation helps in deciding the next course of action i.e. whether to move up or down the slope

Types of Hill Climbing Algorithm:

1) Simple hill Climbing:
2) Steepest-Ascent hill-climbing:
3) Stochastic hill Climbing:

1. Simple Hill Climbing:

- Simple hill climbing is the simplest way to implement a hill climbing algorithm. **It only evaluates the neighbor node state at a time and selects the first one which optimizes current cost and set it as a current state**. It only checks it's one successor state, and if it finds better than the current state, then move else be in the same state.

This algorithm has the following features:

- Less time consuming
- Less optimal solution and the solution is not guaranteed

Algorithm for Simple Hill Climbing:

Step 1: Evaluate the initial state, if it is goal state then return success and Stop.

Step 2: Loop Until a solution is found or there is no new operator left to apply.

Step 3: Select and apply an operator to the current state.

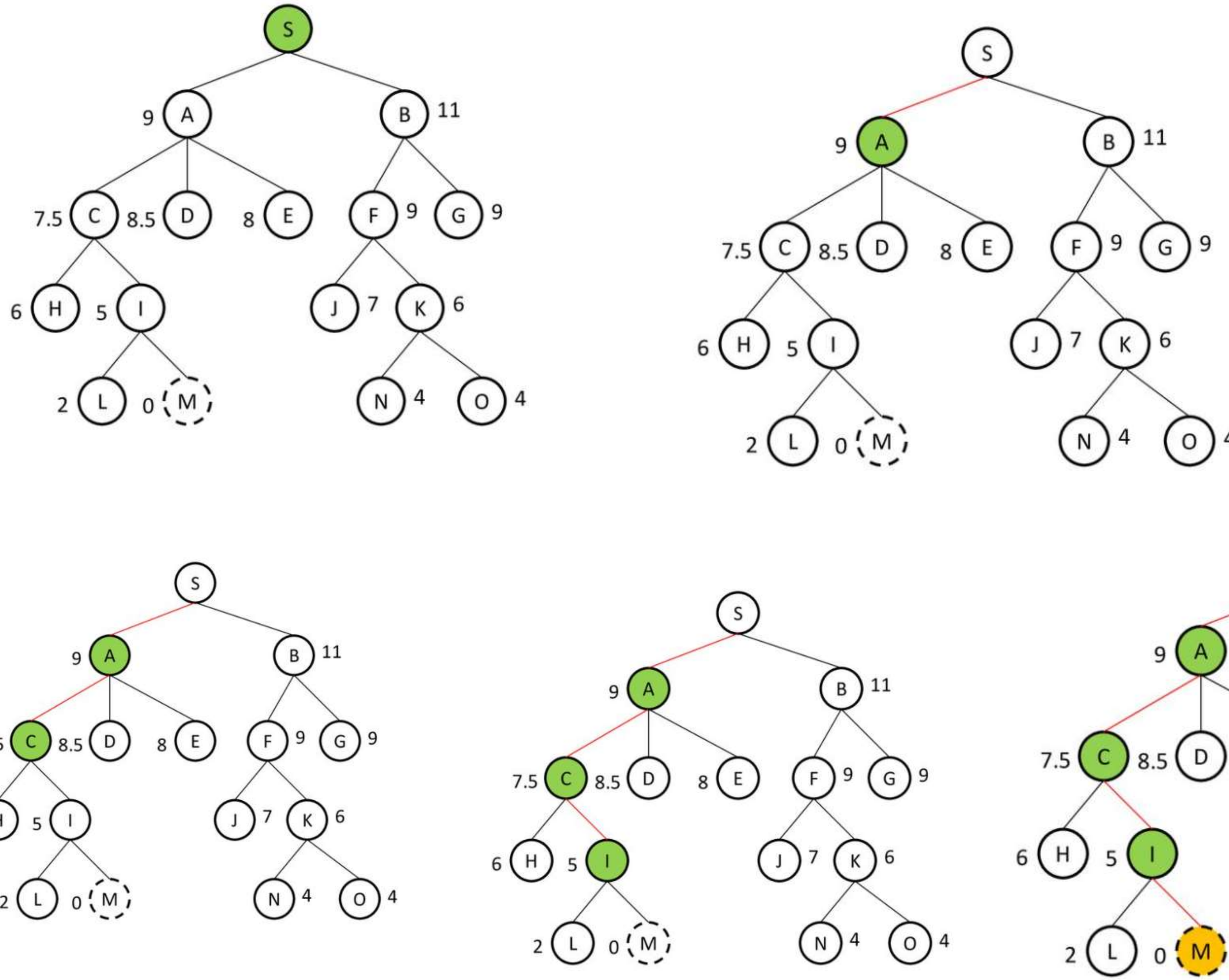
Step 4: Check new state:

If it is goal state, then return success and quit.

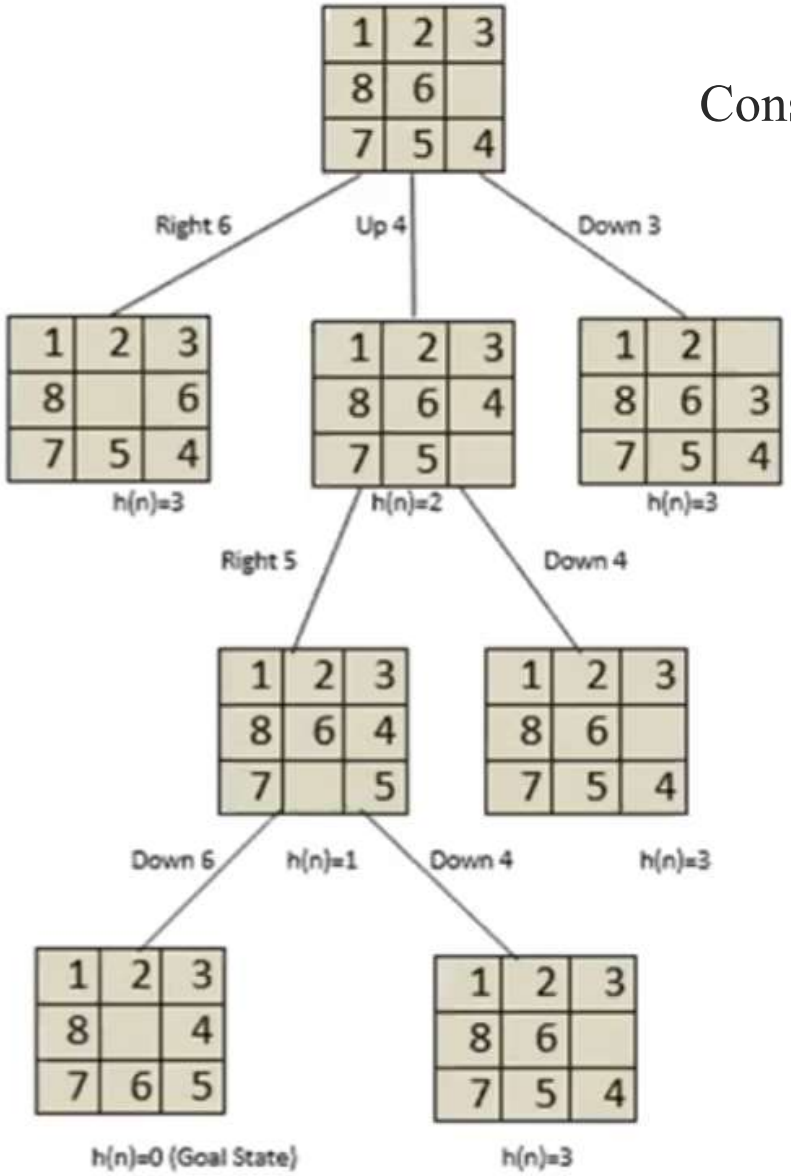
Else if it is better than the current state then assign new state as a current state.

Else if not better than the current state, then return to step2.

Step 5: Exit.


S
9 A B 11
7.5 C 8.5 D 8 E F 9 G 9
6 H 5 I J 7 K 6
2 L 0 M N 4 O 4
S
9 A B 11
7.5 C 8.5 D 8 E F 9 G 9
6 H 5 I J 7 K 6
2 L 0 M N 4 O 4
S
9 A B 11
7.5 C 8.5 D 8 E F 9 G 9
6 H 5 I J 7 K 6
2 L 0 M N 4 O 4
S
9 A B 11
7.5 C 8.5 D 8 E F 9 G 9
6 H 5 I J 7 K 6
2 L 0 M N 4 O 4
S
9 A B 11
7.5 C 8.5 D 8 E F 9 G 9
6 H 5 I J 7 K 6
2 L 0 M N 4 O 4

8 Puzzle Problem using Hill Climbing



Consider h(n) = Number of misplaced tiles.

2. Steepest-Ascent hill-climbing:

- Steepest-Ascent Hill-Climbing algorithm (gradient search) is a variant of Hill Climbing algorithm.

- In the case of hill climbing technique we picked any state as a successor which was closer to the goal than the current state whereas, in Steepest-Ascent Hill Climbing algorithm, we choose the best successor among all possible successors and then update the current state.

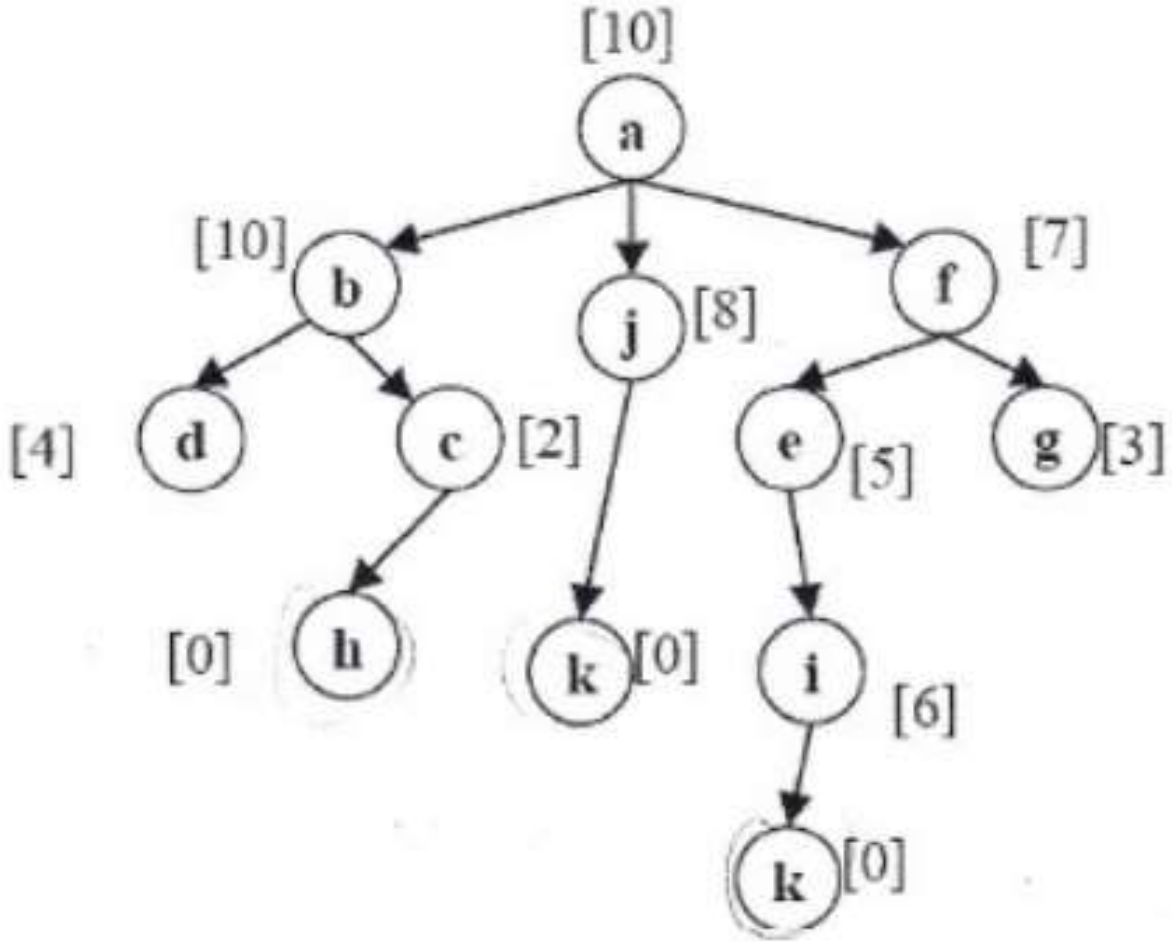
Algorithm for Steepest-Ascent hill climbing

- **Step 1:** Evaluate the initial state, if it is goal state then return success and stop, else make current state as initial state.
- **Step 2:** Loop until a solution is found or the current state does not change.
    a. Let SUCC be a state such that any successor of the current state will be better than it.
    b. For each operator that applies to the current state:
        a. Apply the new operator and generate a new state.
        b. Evaluate the new state.
        c. If it is goal state, then return it and quit, else compare it to the SUCC.
        d. If it is better than SUCC, then set new state as SUCC.
        e. If the SUCC is better than the current state, then set current state to SUCC.
- **Step 5:** Exit.

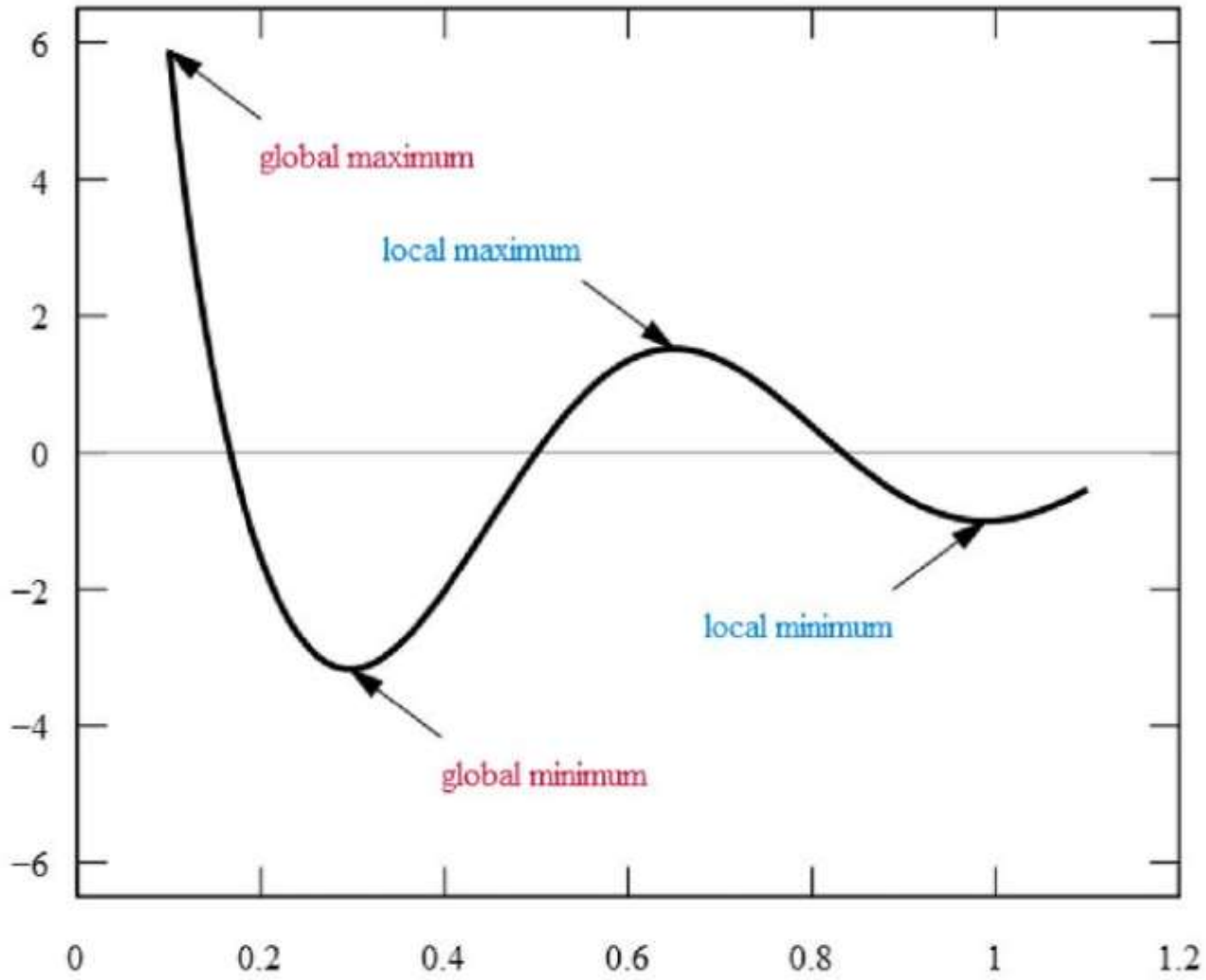
3. Stochastic hill climbing

- Stochastic hill climbing chooses at random from among the uphill moves.
- Stochastic hill climbing does not examine for all its neighbor before moving. Rather, this search algorithm **selects one neighbor node at random and decides whether** to choose it as a current state or examine another state.
- The probability of selection can vary with the steepness of the uphill move.
- Stochastic hill climbing usually converges more slowly than steepest ascent, but in some state landscapes, it finds better solutions.
- Stochastic hill climbing is NOT complete, but it may be less likely to get stuck.

Hill Climbing Algorithm Example



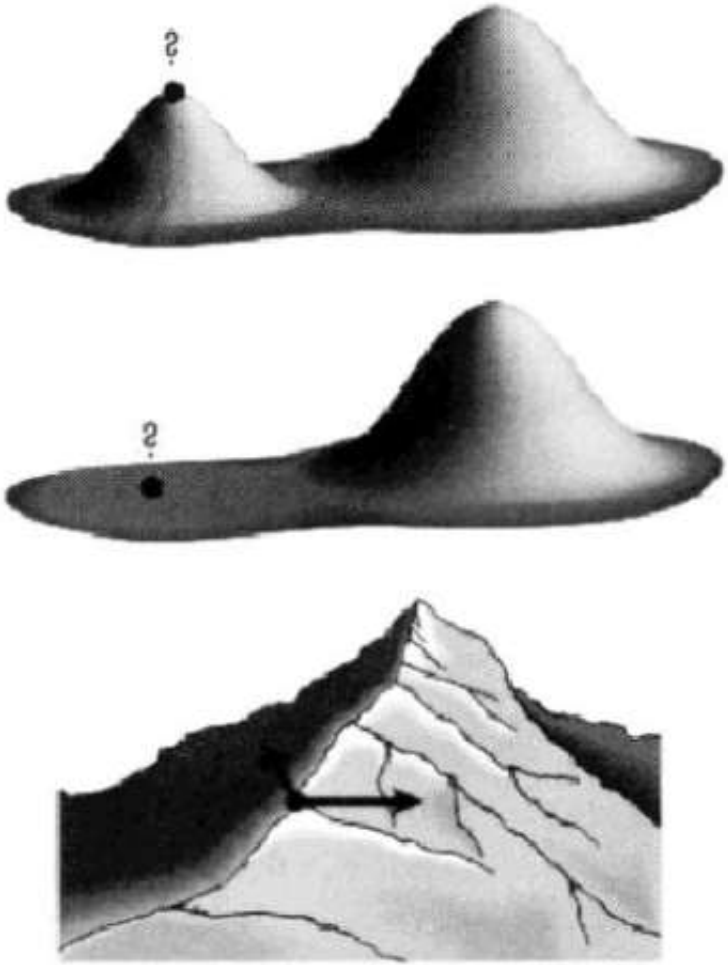
Local Maximum and Local Minimum

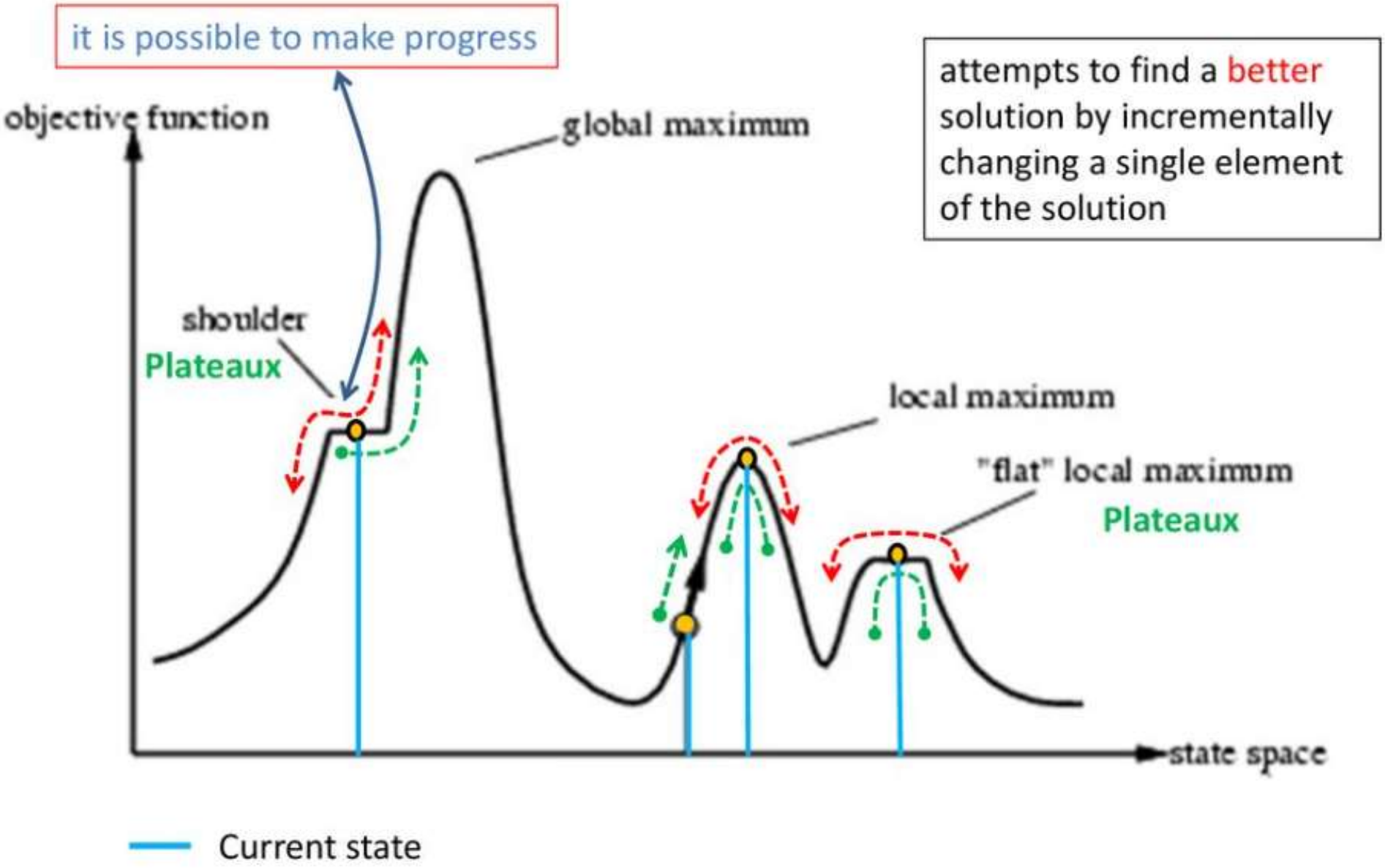


Local maxima: a local maximum is a peak that is higher than each of its neighboring states, but lower than the global maximum. Hill-climbing algorithms that reach the vicinity of a local maximum will be drawn upwards towards the peak, but will then be stuck with nowhere else to go.

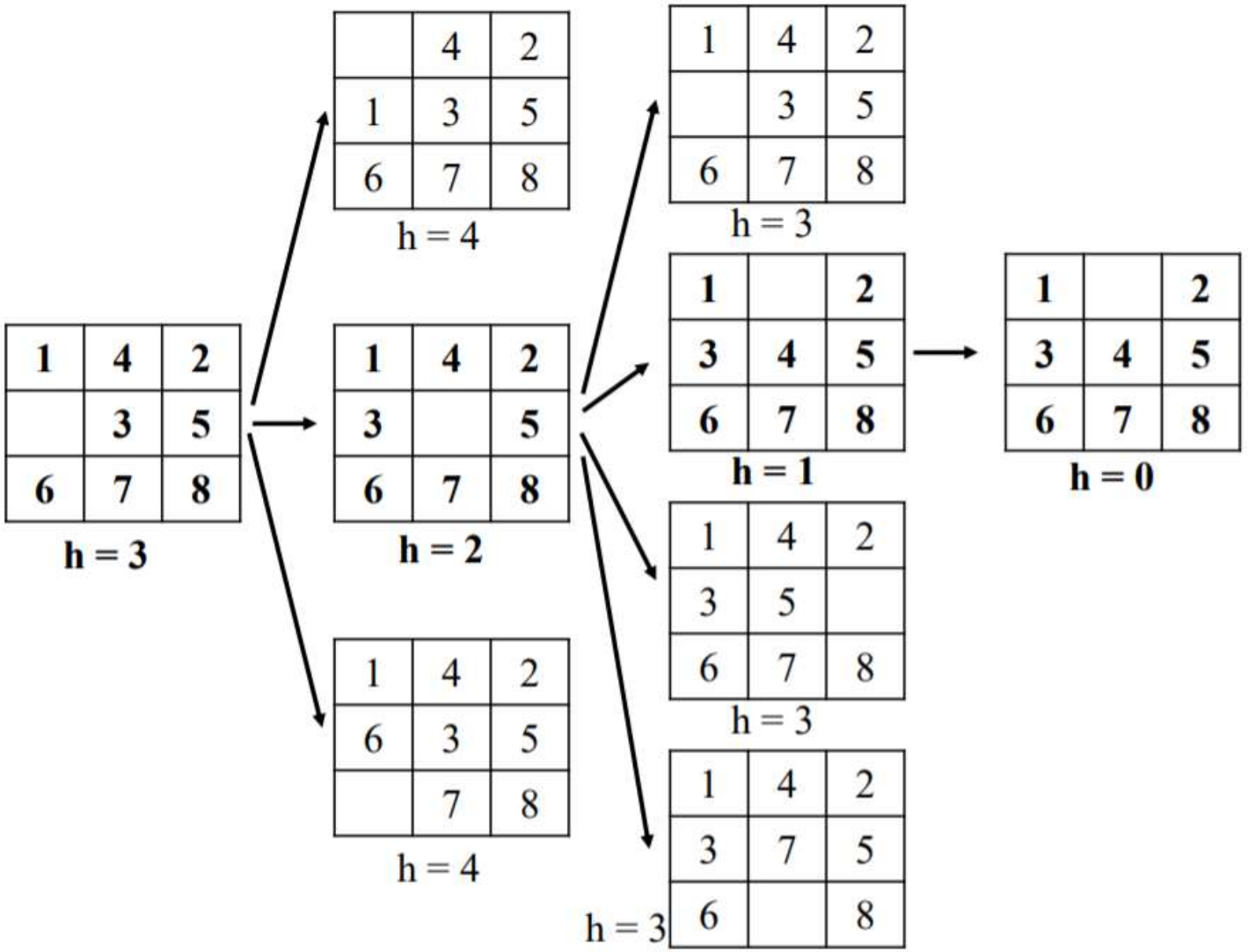
Plateaux: a plateau is an area of the state space landscape where the evaluation function is flat. It can be a flat local maximum, from which no uphill exit exists, or a **shoulder,** from which it is possible to make progress.

Ridges: Ridges result in a sequence of local maxima that is very difficult for greedy algorithms to navigate. (the search direction is not towards the top but towards the side)

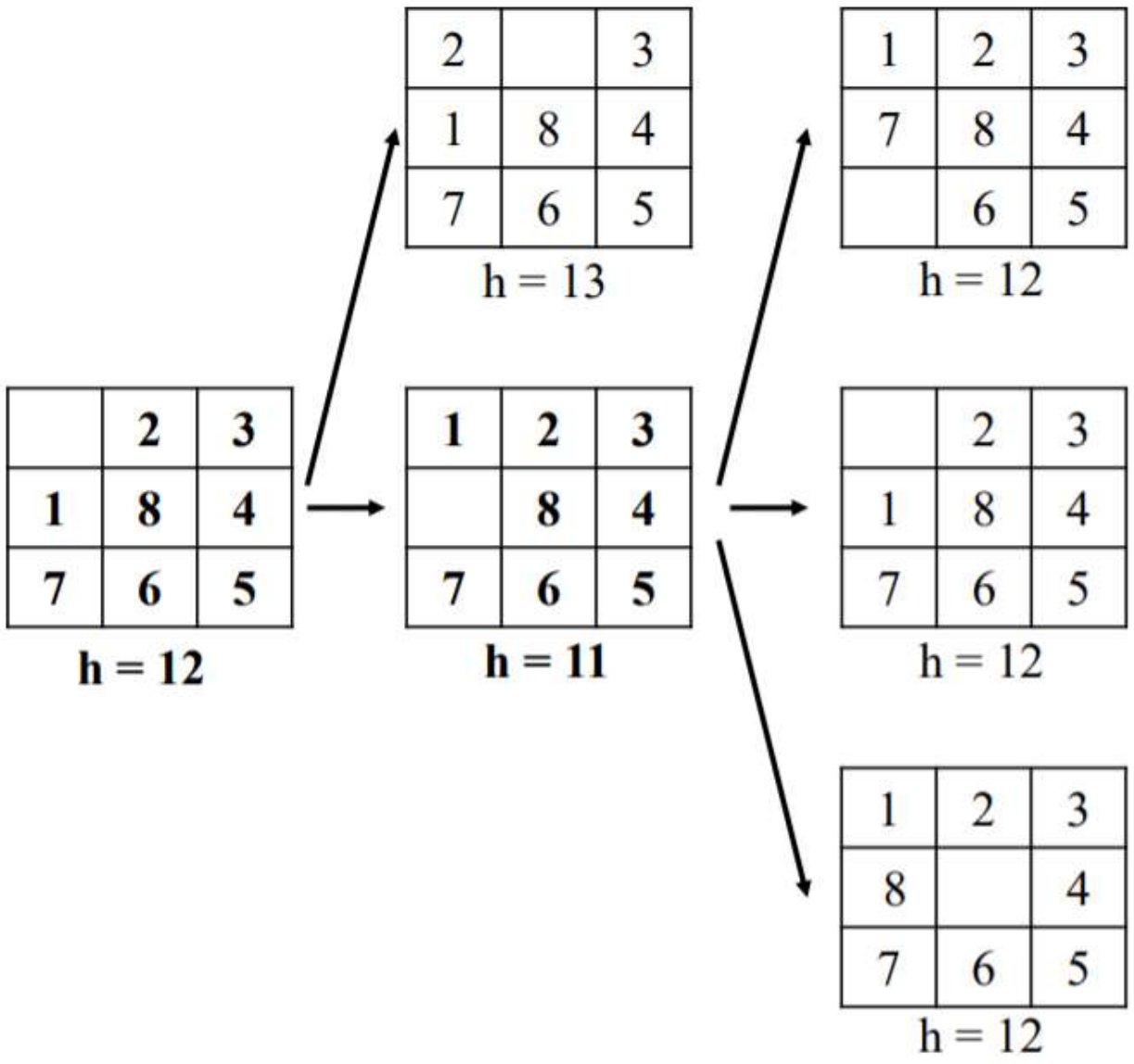



it is possible to make progress
attempts to find a better solution by incrementally changing a single element of the solution
objective function
global maximum
shoulder
Plateaux
local maximum
"flat" local maximum
Plateaux
state space
Current state

8-puzzle: a solution case



8-puzzle: stuck at local maximum



Blocks World Problem with Local Heuristic Function

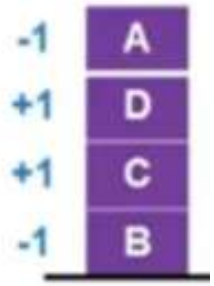
+1 for each block that is resting on the thing it is suppose to be resting on.
-1 for each block that is resting on wrong thing.

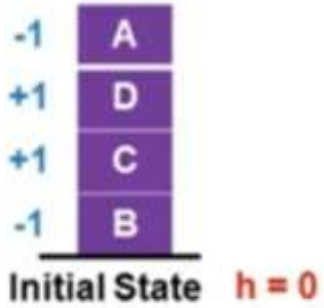
A
D
C
B

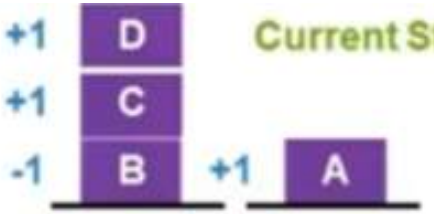
Initial

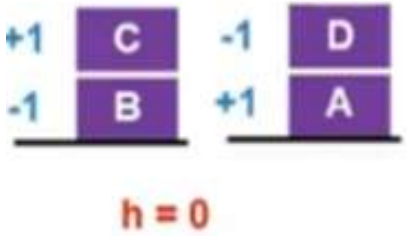
D
C
B
A

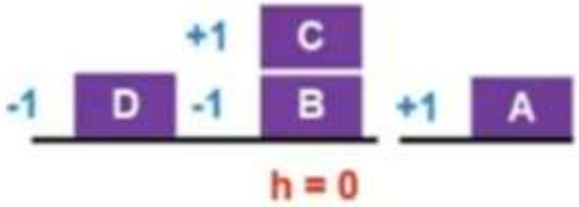
Goal

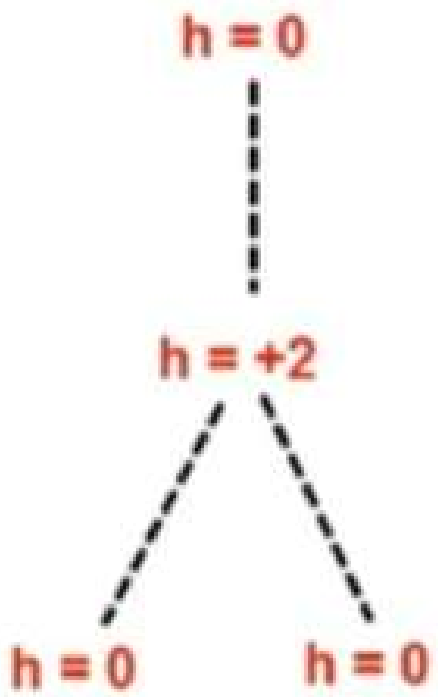


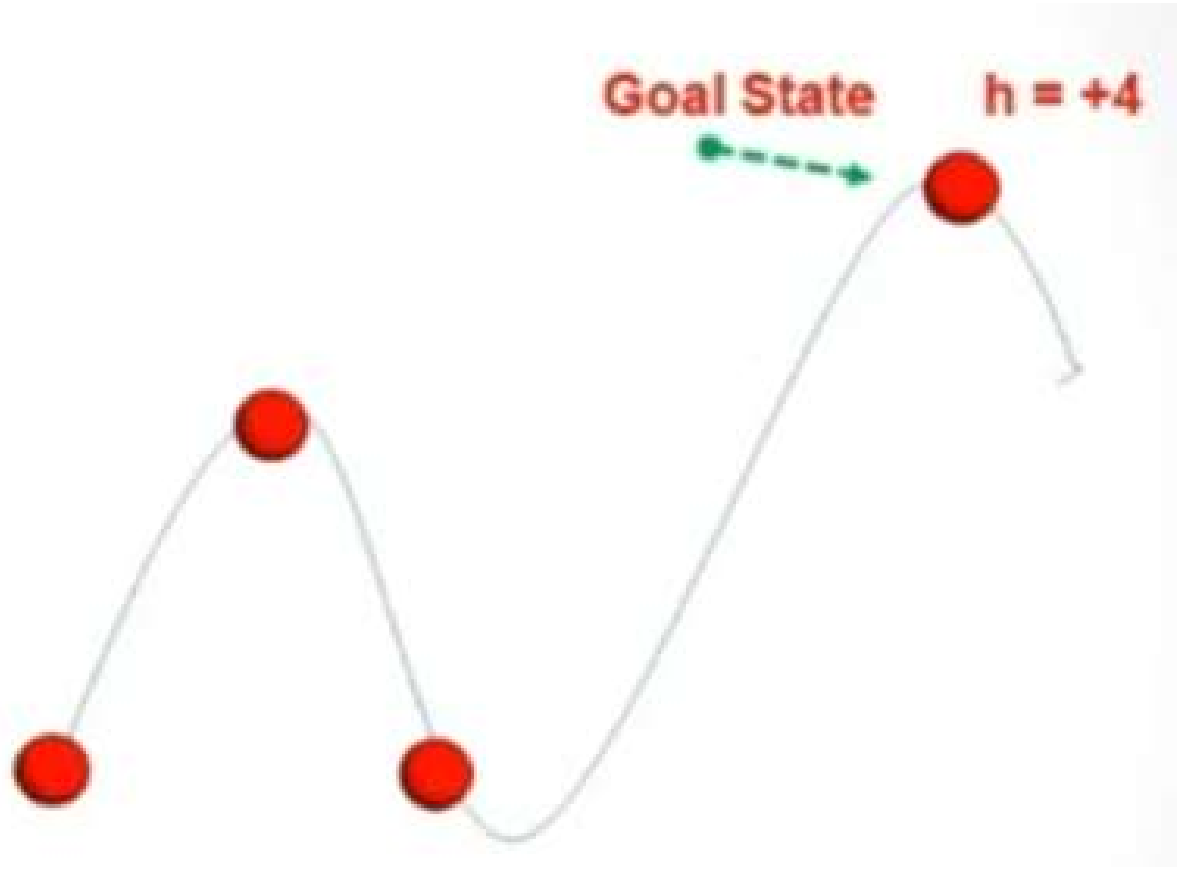









h = 0
h = +2
h = 0
h = 0


Goal State
h = +4

Blocks World Problem with Global Heuristic Function

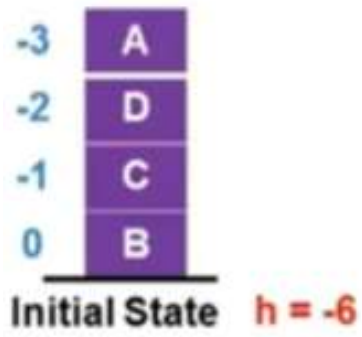
- *For each block that has the correct support structure : +1 to every block in the support structure.*
- *For each block that has the wrong support structure : -1 to every block in the support structure.*

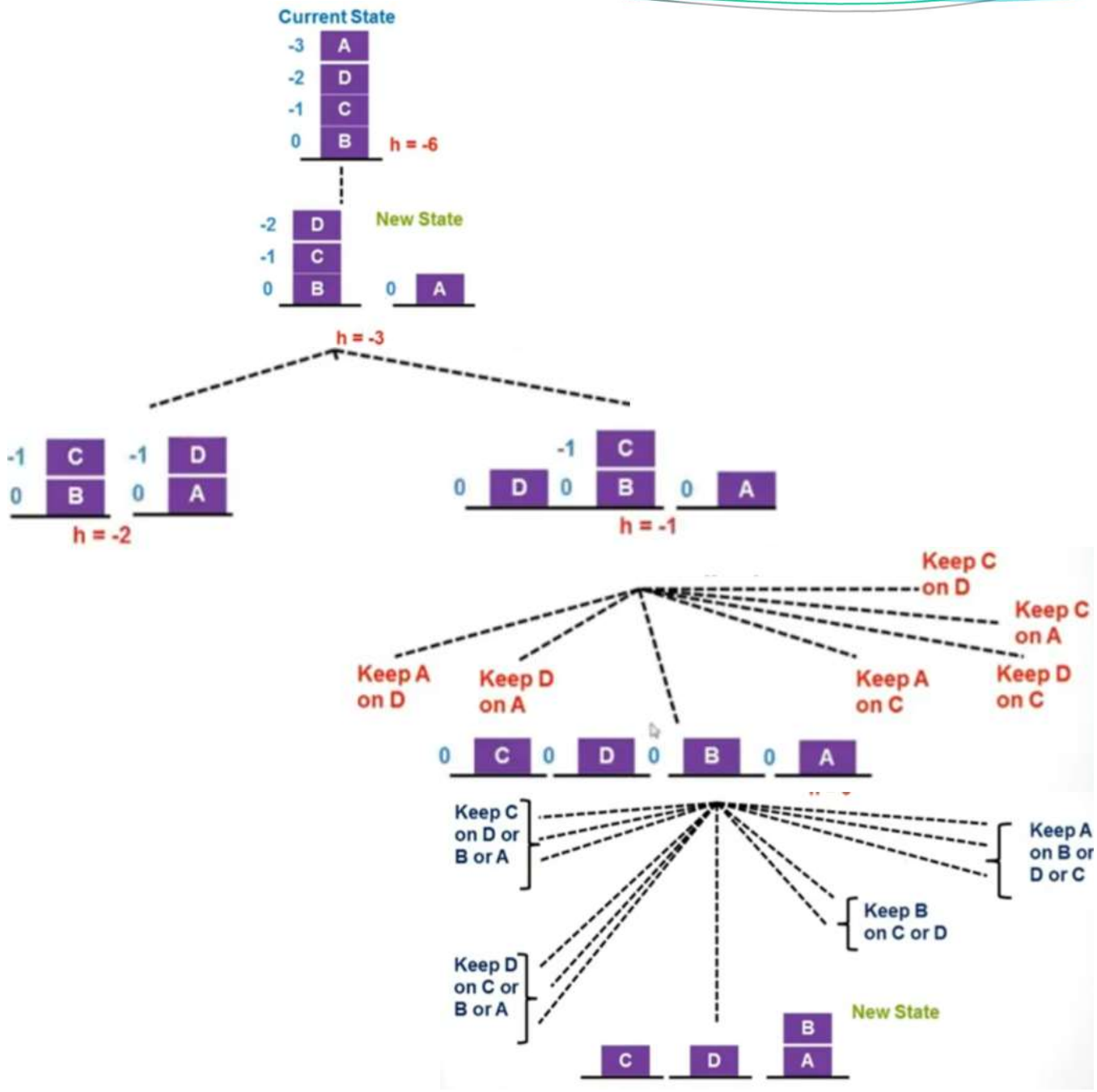
A
D
C
B

Initial

D
C
B
A

Goal




Current State
-3 A
-2 D
-1 C
0 B
h = -6
New State
-2 D
-1 C
0 B
0 A
h = -3
-1 C
-1 D
0 B
0 A
h = -2
-1 C
0 D
0 B
0 A
h = -1
Keep C on D
Keep C on A
Keep A on D
Keep D on A
Keep A on C
Keep D on C
0 C
0 D
0 B
0 A
Keep C on D or B or A
Keep A on B or D or C
Keep B on C or D
Keep D on C or B or A
New State
C
D
B
A

Blocks World Problem with Local Heuristic Function

+1 for each block that is resting on the thing it is suppose to be resting on.
-1 for each block that is resting on wrong thing.

A
H
G
F
E
D
C
B

Initial

H
G
F
E
D
C
B
A

Goal

Blocks World Problem with Global Heuristic Function

- ***For each block that has the correct support structure : +1 to every block in the support structure.***
- ***For each block that has the wrong support structure : -1 to every block in the support structure.***

A
H
G
F
E
D
C
B

Initial

H
G
F
E
D
C
B
A

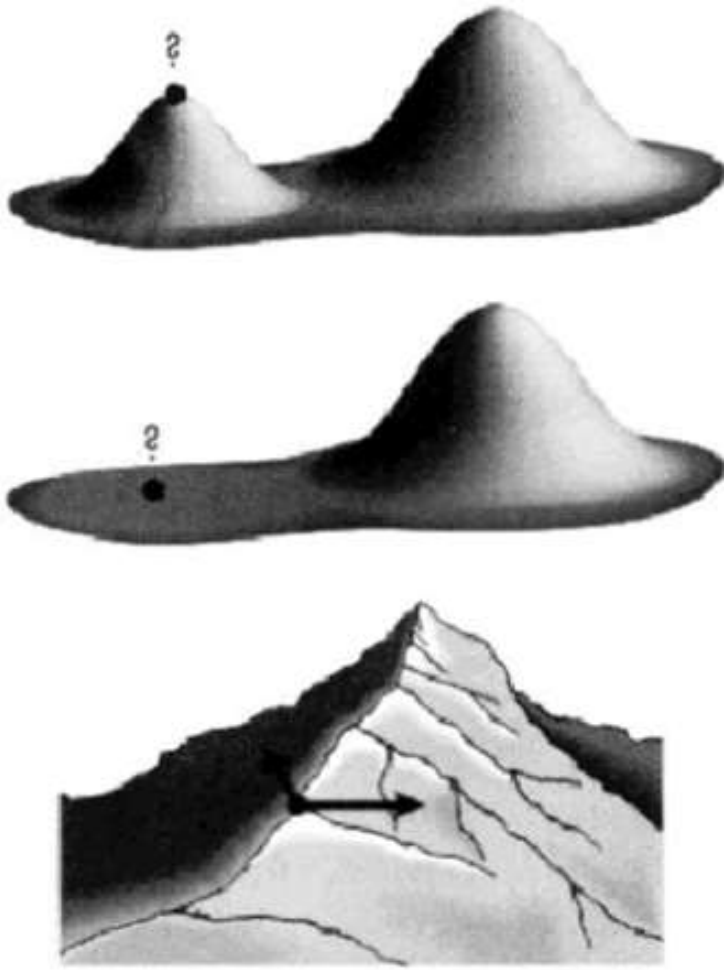
Goal

Solution to Overcome Problems in Hill Climbing

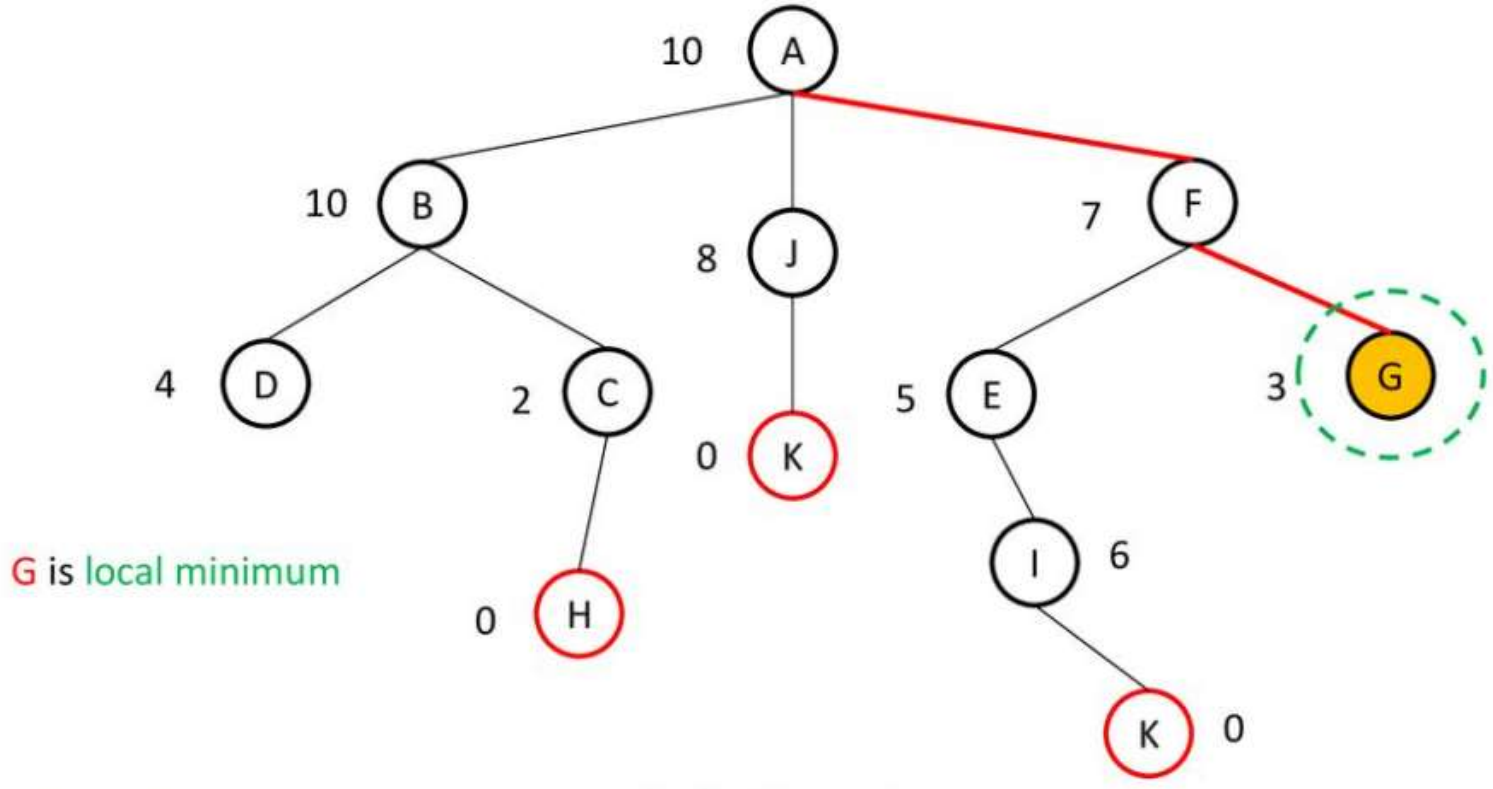
1. **Local maximum:**
   Utilize the ***backtracking* technique**. Maintain a list of visited states. If the search reaches an undesirable state, it can backtrack to the previous configuration and explore a new path.

2. **Plateau:** Make a **big jump**. Randomly select a state far away from the current state. Chances are that we will land in a non-plateau region.

3. **Ridge:** In this kind of obstacle, use **two or more rules** before testing. It implies moving in several directions at once.



Hill Climbing Search Example with Local Maxima



- Hill Climbing is sometimes called greedy local search because it grabs a good neighbor state without thinking ahead about where to go next.