Informed Search

A*, memory bounded, Local Search Algorithms

A* Algorithm

It works as follows:

It maintains a tree of paths originating at the start node.
It extends those paths one edge at a time.
It continues until its termination criterion is satisfied.

A* Algorithm extends the path that minimizes the following function-

$$\mathbf{f(n) = g(n) + h(n)}$$

Algorithm

The implementation of A* Algorithm involves maintaining two lists- OPEN and CLOSED.
OPEN contains those nodes that have been evaluated by the heuristic function but have not been expanded into successors yet.
CLOSED contains those nodes that have already been visited.

The algorithm is as follows-

Step-01:

Define a list OPEN.

Initially, OPEN consists solely of a single node, the start node S.

Step-02: If the list is empty, return failure and exit.

Step-03:

Remove node n with the smallest value of f(n) from OPEN and move it to list CLOSED.

If node n is a goal state, return success and exit.

Step-04: Expand node n.

Step-05:

If any successor to n is the goal node, return success and the solution by tracing the path from goal node to S.
Otherwise, go to Step-06.

Step-06:

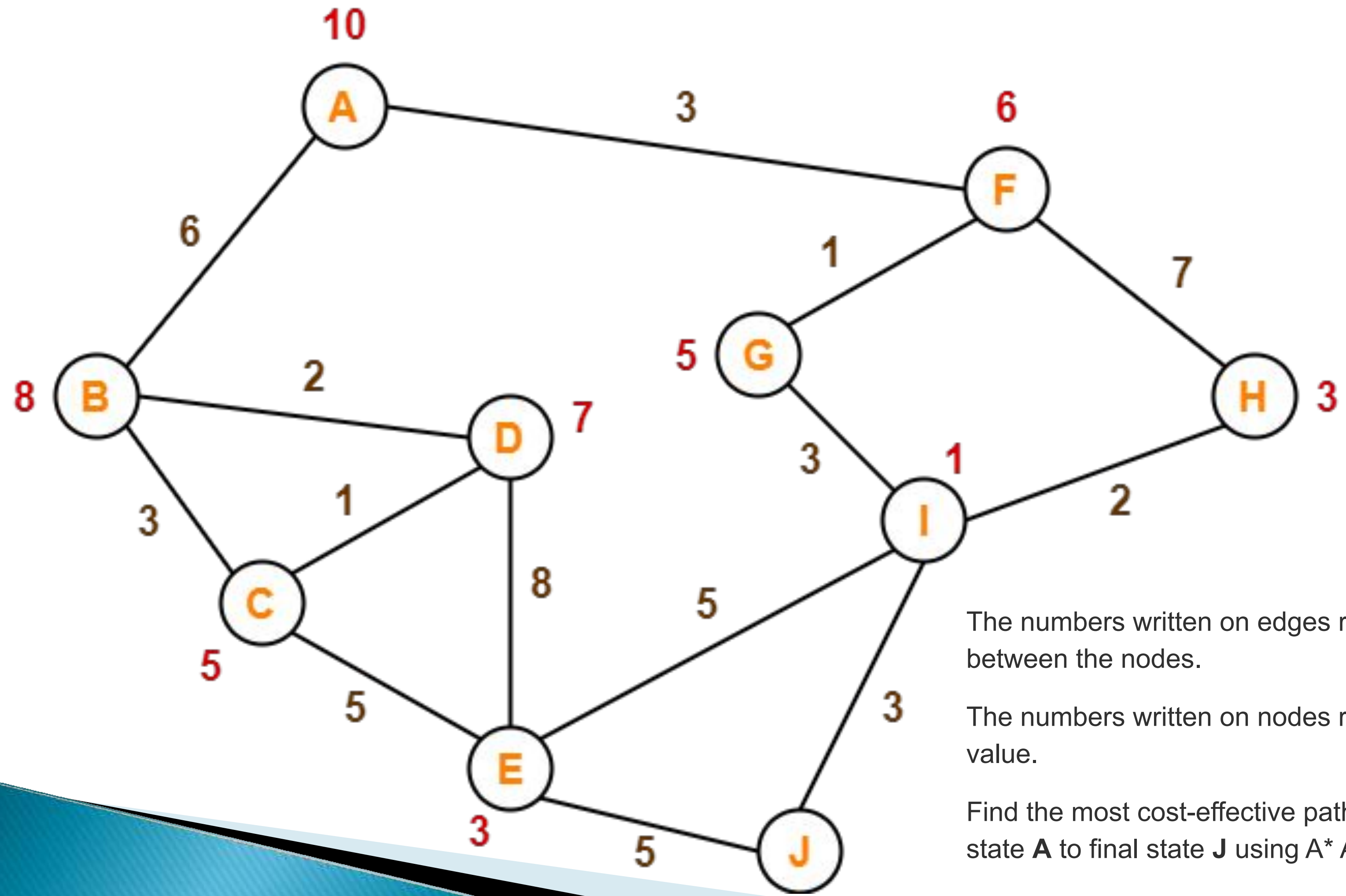
For each successor node,

Apply the evaluation function f to the node.

If the node has not been in either list, add it to OPEN.

Step-07:

Go back to Step-02.



The numbers written on edges represent the distance between the nodes.

The numbers written on nodes represent the heuristic value.

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

We start with node A.
Node B and Node F can be reached from node A.

A* Algorithm calculates f(B) and f(F).

$f(B) = 6 + 8 = 14$
$f(F) = 3 + 6 = 9$

Since $f(F) < f(B)$, so it decides to go to node F.

Path- A → F

Node G and Node H can be reached from node F.

A* Algorithm calculates f(G) and f(H).

$f(G) = (3+1) + 5 = 9$
$f(H) = (3+7) + 3 = 13$

Since f(G) < f(H), so it decides to go to node G.

Path- A → F → G

Node I can be reached from node G.

A* Algorithm calculates f(I).

f(I) = (3+1+3) + 1 = 8

It decides to go to node I.

Path- A → F → G → I

Node E, Node H and Node J can be reached from node I.

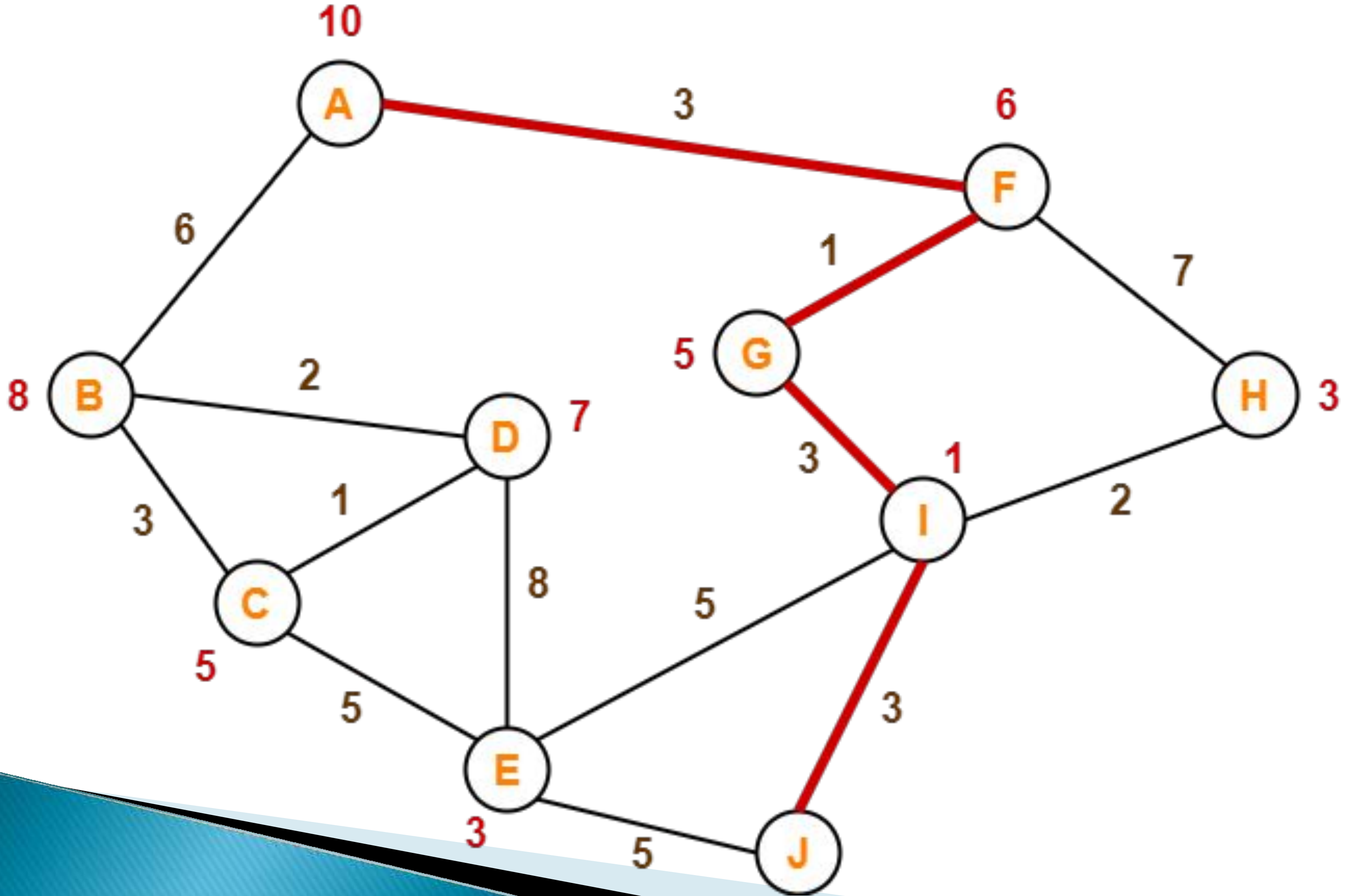
A* Algorithm calculates f(E), f(H) and f(J).

$$f(E) = (3+1+3+5) + 3 = 15$$
$$f(H) = (3+1+3+2) + 3 = 12$$
$$f(J) = (3+1+3+3) + 0 = 10$$

Since f(J) is least, so it decides to go to node J.

Path- A → F → G → I → J

This is the required shortest path from node A to node J.


10
A
3
6
F
6
1
7
G
5
8
B
2
H
3
D
7
3
1
1
3
I
2
C
8
5
5
5
3
E
3
5
J

Memory Bounded Heuristic Search: Recursive BFS

How can we solve the memory problem for A* search?

Idea: Try something like depth first search, but let’s not forget everything about the branches we have partially explored.

We remember the best f-value we have found so far in the branch we are deleting.

Simple Memory Bounded A*

This is like A*, but when memory is full we delete the worst node (largest f-value).

Like RBFS, we remember the best descendent in the branch we delete.

If there is a tie (equal f-values) we delete the oldest nodes first.

simple-MBA* finds the optimal reachable solution given the memory constraint.

Time can still be exponential.

Local search algorithms

In many optimization problems, the path to the goal is irrelevant; the goal state itself is the solution

State space = set of "complete" configurations

Find configuration satisfying constraints, e.g., n-queens

In such cases, we can use local search algorithms

keep a single "current" state, try to improve it.

Very memory efficient (only remember current state)

Hill Climbing

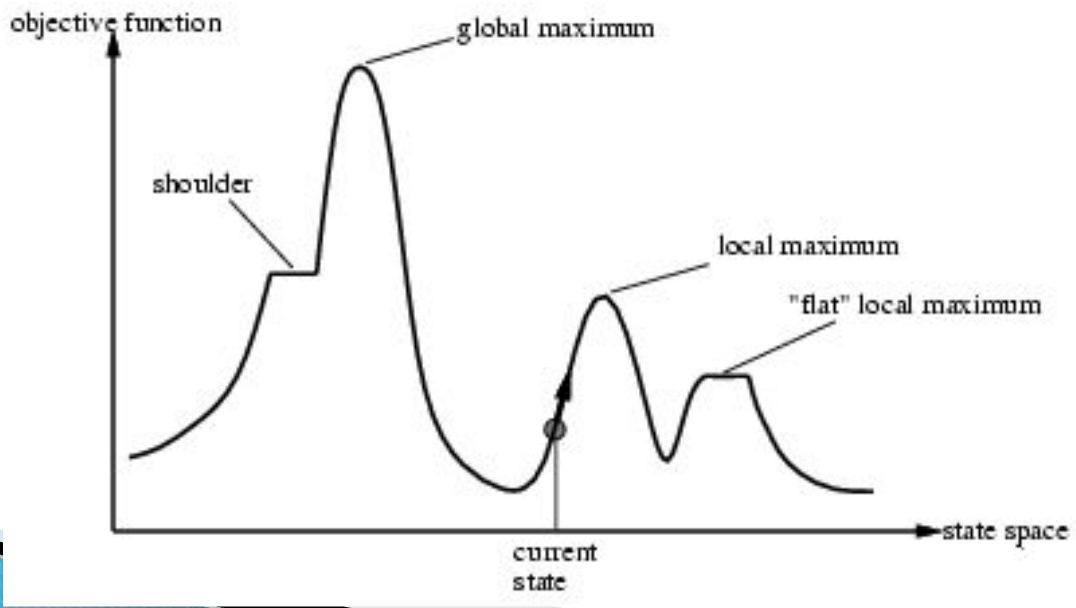
- Hill Climbing is a heuristic search used for **mathematical optimisation problems** in the field of Artificial Intelligence.

- Hill-climbing solves the problems where we need to maximise or minimise a given real function by selecting values from the given inputs.

- From a large set of inputs and a good heuristic function, the algorithm tries to find the best possible solution to the problem in the most reasonable time period. This solution may not be the absolute best(global optimal maximum) but it is sufficiently good considering the time allotted.

Hill-climbing search

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



Simulated annealing search

Idea: escape local maxima by allowing some "bad" moves but gradually decrease their frequency.

This is like smoothing the cost landscape.

Properties of simulated annealing search

One can prove: If T decreases slowly enough, then simulated annealing search will find a global optimum with probability approaching 1 (however, this may take VERY long)

Widely used in VLSI layout, airline scheduling, etc.

Local beam search

Keep track of k states rather than just one.

Start with k randomly generated states.

At each iteration, all the successors of all k states are generated.

If any one is a goal state, stop; else select the k best successors from the complete list and repeat.

Genetic algorithms

A successor state is generated by combining two parent states

Start with k randomly generated states (population)

A state is represented as a string over a finite alphabet (often a string of 0s and 1s)

Evaluation function (fitness function). Higher values for better states.

Produce the next generation of states by selection, crossover, and mutation

Summary

Blind search: Depth-First, Breadth-First, IDS

– Do not use knowledge of problem space to find solution.

Informed search

Best-first search: Order agenda based on some measure of how ‘good’ each state is.

Uniform-cost: Cost of getting to current state from initial state = **`g(n)`**

Greedy search: Estimated cost of reaching goal from current state

Heuristic evaluation functions, **`h(n)`**

A search:* **`f(n) = g(n) + h(n)`**