Heuristic Search Techniques

Generate and Test

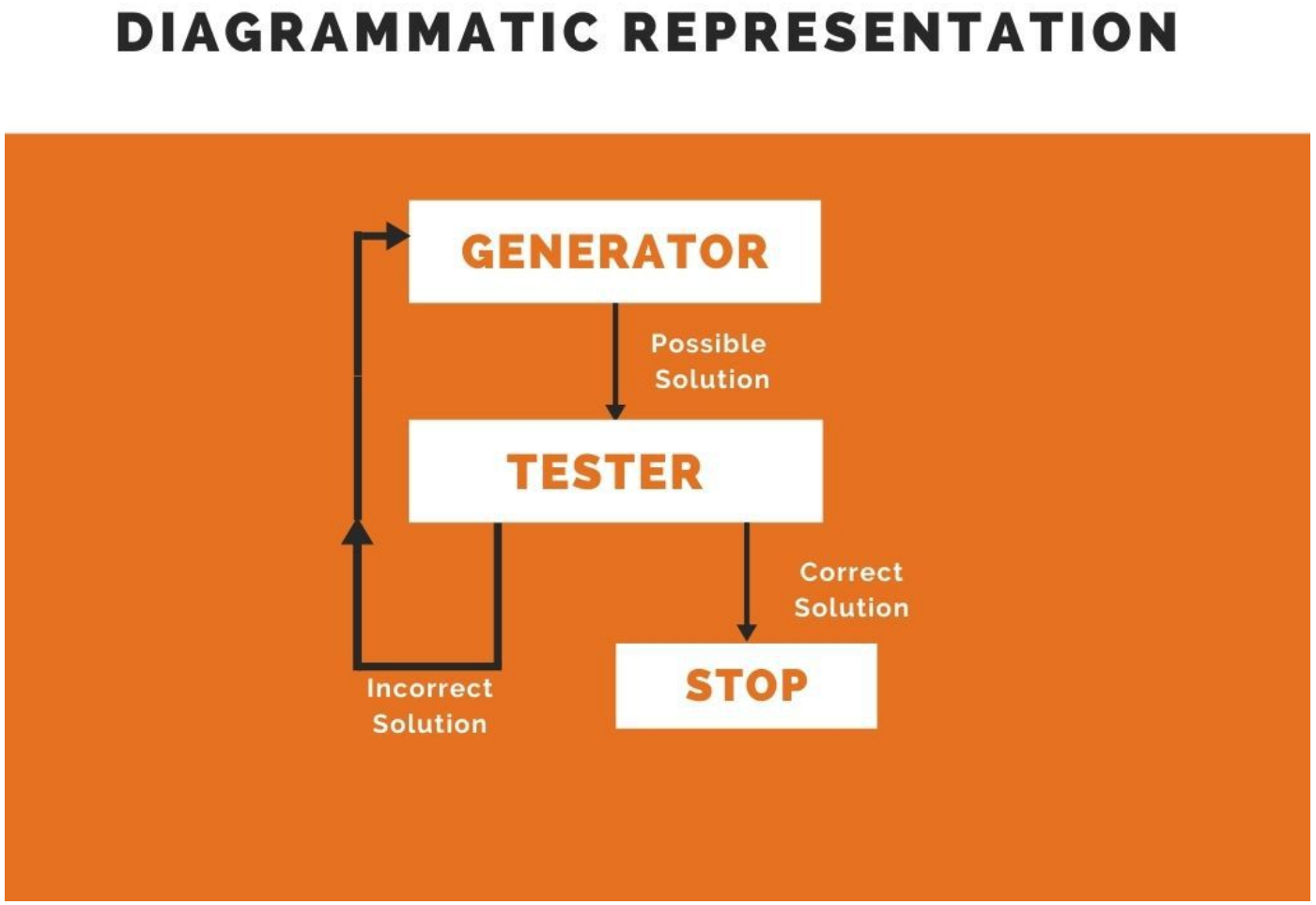
- Generate and Test Search is a heuristic search technique based on Depth First Search with Backtracking which guarantees to find a solution if done systematically and there exists a solution.
- In this technique, all the solutions are generated and tested for the best solution.
- It ensures that the best solution is checked against all possible generated solutions.
- It is also known as **British Museum Search Algorithm** as it's like looking for an exhibit at random or finding an object in the British Museum by wandering randomly.

Generate and Test

Algorithm:

1. Generate a possible solution. For example, generating a particular point in the problem space or generating a path for a start state.
2. Test to see if this is a actual solution by comparing the chosen point or the endpoint of the chosen path to the set of acceptable goal states.
3. If a solution is found, quit. Otherwise go to Step 1

Generate and Test



Generate and Test-Example

Problem:

“Arrange four 6-sided cubes in a row, with each side of each cube painted one of four colors, such that on all four sides of the row one block face of each color is showing.”

Heuristic:

If there are more red faces than other colors then, when placing a block with several red faces, use few of them as possible as outside faces.

Generate and Test

The evaluation is carried out by the heuristic function as all the solutions are generated systematically in generate and test algorithm but if there are some paths which are most unlikely to lead us to result then they are not considered.

Systematic Generate and Test may prove to be ineffective while solving complex problems.

But there is a technique to improve in complex cases as well by combining generate and test search with other techniques so as to reduce the search space.

Hill Climbing

- Hill Climbing is a heuristic search used for mathematical optimization problems in the field of Artificial Intelligence.
- Given a large set of inputs and a good heuristic function, it tries to find a sufficiently good solution to the problem. This solution may not be the global optimal maximum.
- Hill climbing is a variant of generate and test algorithm as it takes the feedback from the test procedure. Then this feedback is utilized by the generator in deciding the next move in search space.

Types of Hill Climbing

Simple Hill climbing :

Algorithm:

Step 1 : Evaluate the initial state. If it is a goal state then stop and return success. Otherwise, make initial state as current state.

Step 2 : Loop until the solution state is found or there are no new operators present which can be applied to current state.

a) Select an operator that has not been yet applied to the current state and apply it to produce a new state.

b) Evaluate the new state

i. If the current state is a goal state, then stop and return success.

ii. If it is not a goal state but it is better than the current state, then make it current state and proceed further.

iii. If it is not better than the current state, then continue in the loop.

Types of Hill Climbing

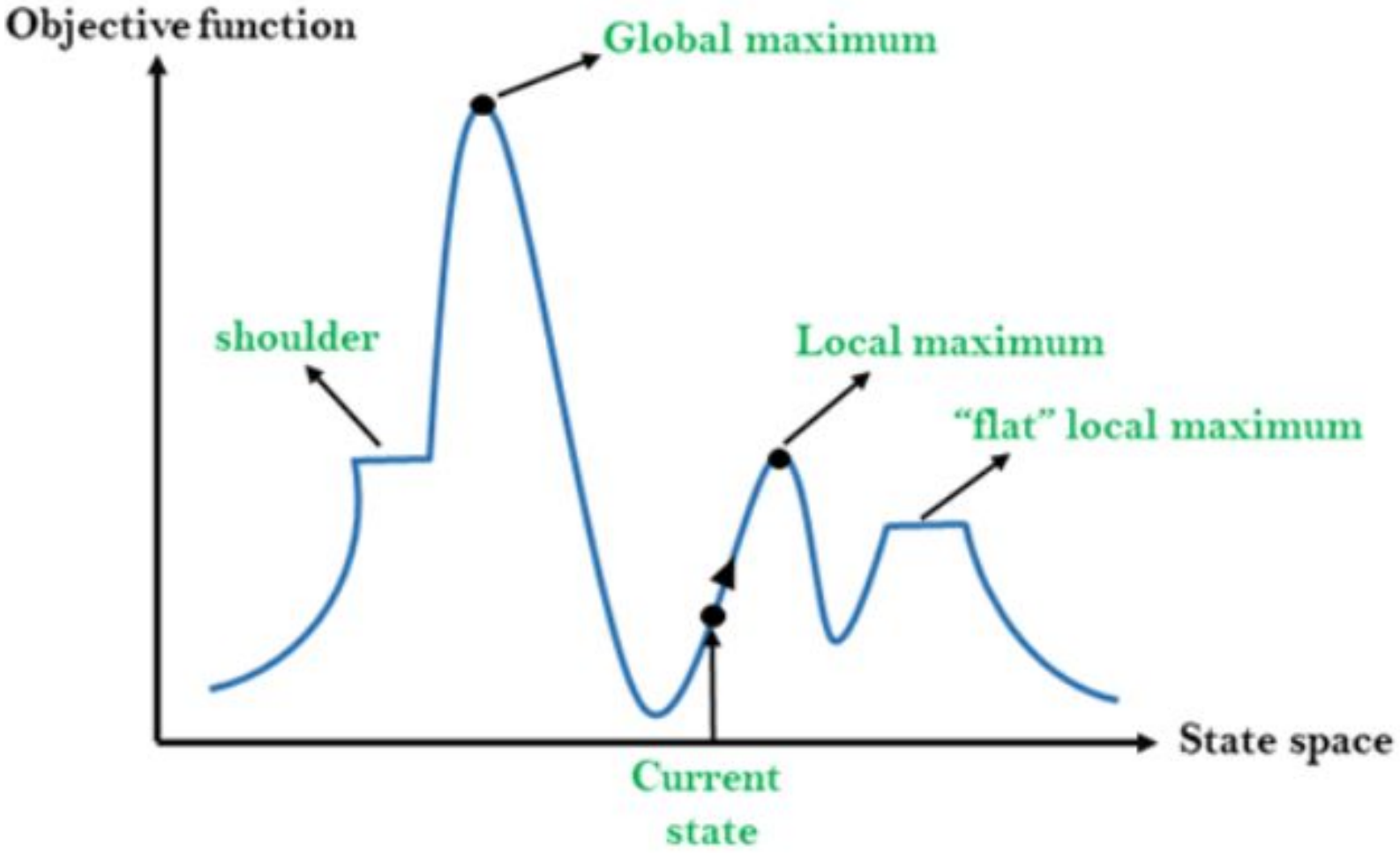
STEEPEST-ASCENT HILL CLIMBING:

Considers all the moves from the current state and Selects the best one as the next state.

Algorithm:

1. Evaluate the initial state. If it is also a goal state, then return it and quit. Otherwise, continue with the initial state as the current state.
2. Loop until a solution is found or until a complete iteration produces no change to current state:
   (a) Let *SUCC* be a state such that any possible successor of the current state will be better than *SUCC*.
   (b) For each operator that applies to the current state do:
      (i) Apply the operator and generate a new state.
      (ii) Evaluate the new state. If it is a goal state, then return it and quit. If not, compare it to *SUCC*. If it is better, then set *SUCC* to this state. If it is not better, leave *SUCC* alone.
   (c) If the *SUCC* is better than current state, then set current state to *SUCC*.

State-space Diagram for Hill Climbing



State-space Diagram for Hill Climbing

- **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/Plateau:** 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.

State-space Diagram for Hill Climbing

Disadvantages Ways Out:

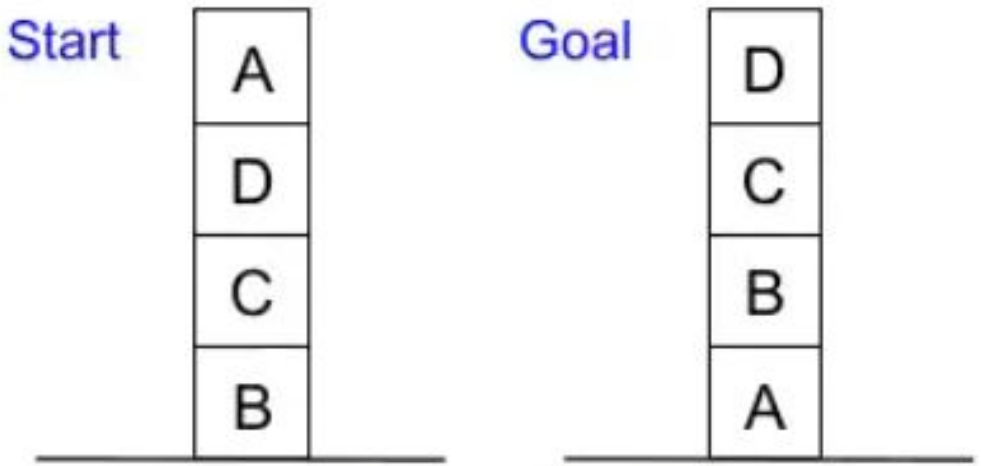
- Backtrack to some earlier node and try going in a different direction.
- Make a big jump to try to get in a new section.
- Moving in several directions at once.

Hill Climbing

Disadvantages:

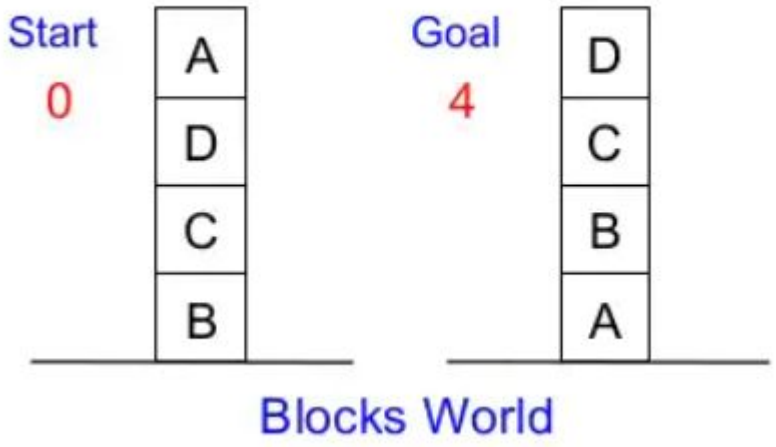
- Hill climbing is a local method: Decides what to do next by looking only at the “immediate” consequences of its choices.
- Global information might be encoded in heuristic functions.

Hill Climbing

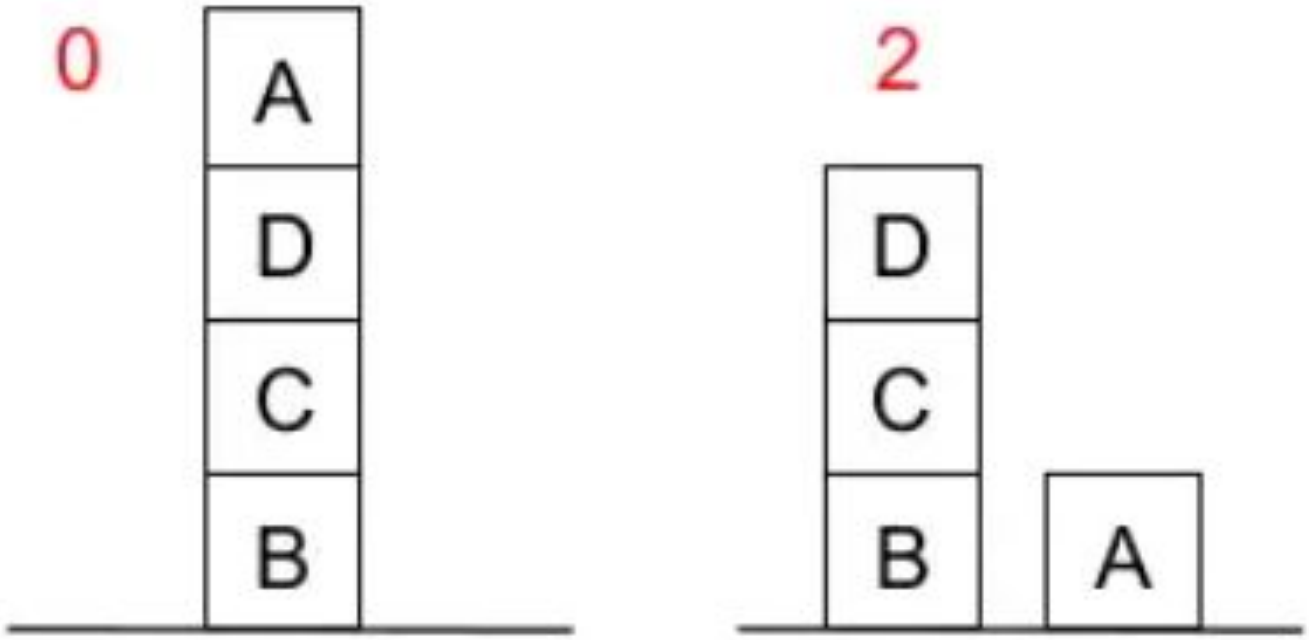


Hill Climbing

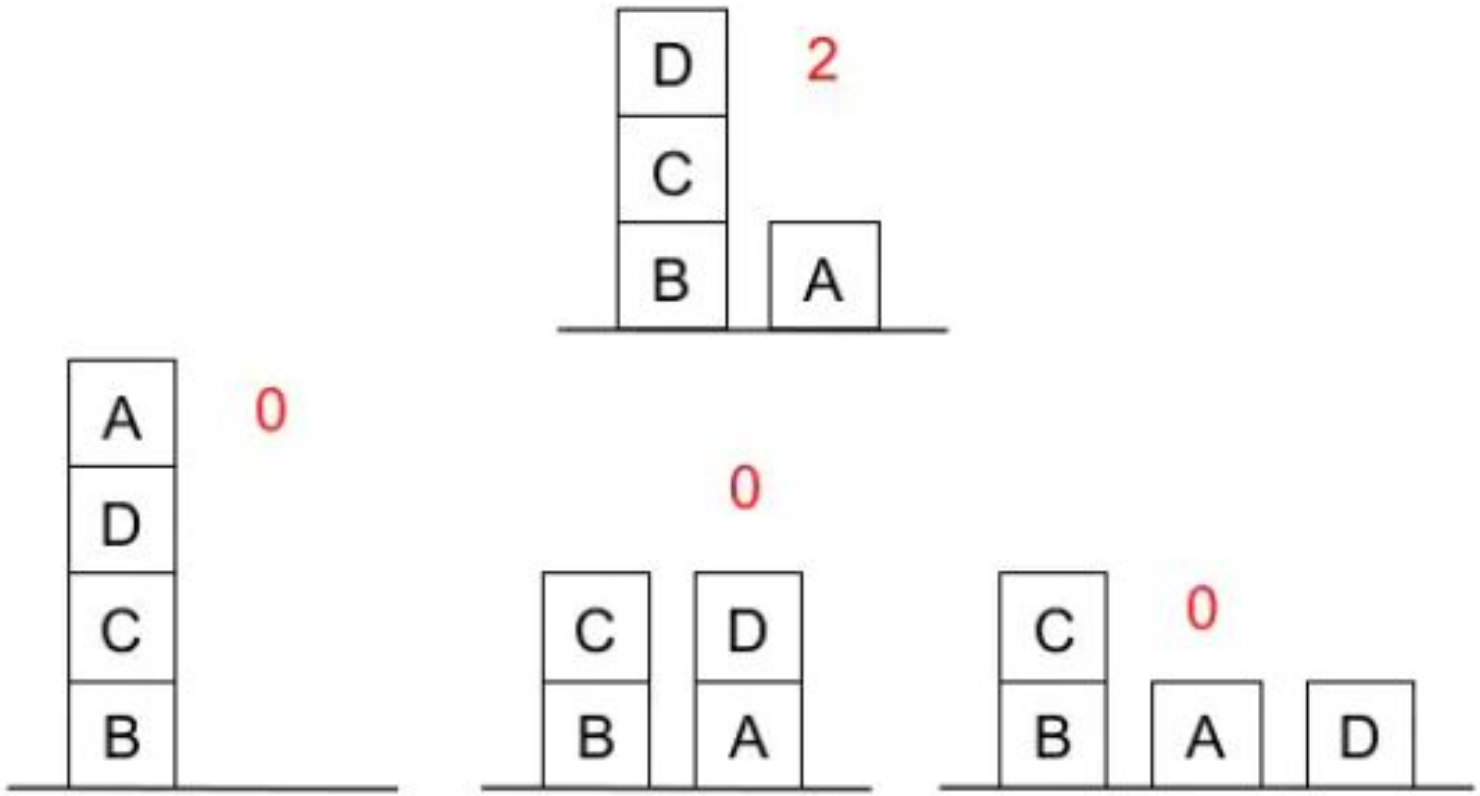
Local heuristic: +1 for each block that is resting on the thing it is supposed to be resting on.
−1 for each block that is resting on a wrong thing.



Hill Climbing



Hill Climbing



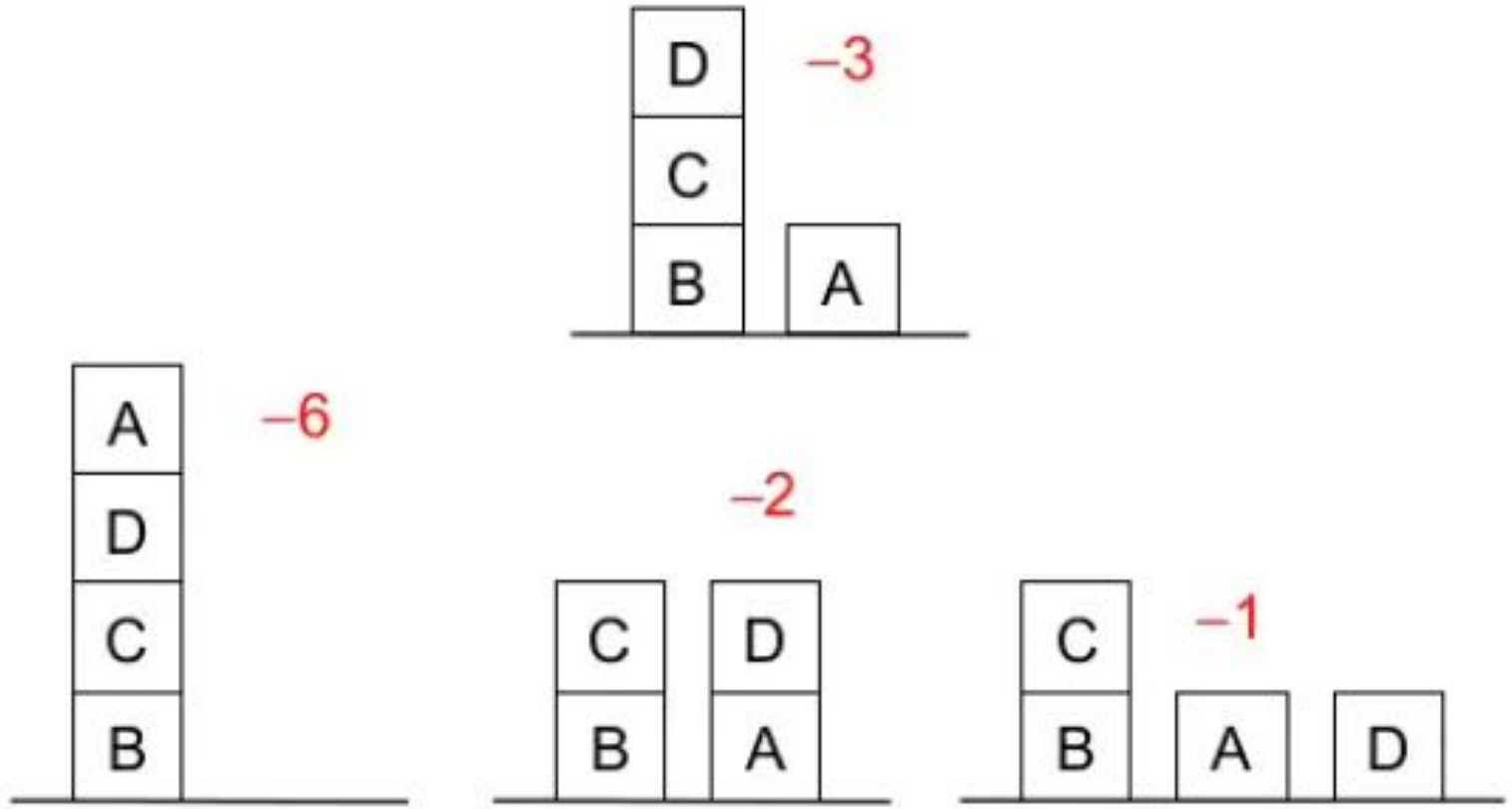
Hill Climbing

Global heuristic:

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

For each block that has a wrong support structure: −1 to every block in the support structure.

Hill Climbing



Hill Climbing

- Can be very inefficient in a large, rough problem space.
- Global heuristic may have to pay for computational complexity.
- Often useful when combined with other methods, getting it started right in the right general neighborhood.

Simulated Annealing

- A variation of hill climbing in which, at the beginning of the process, some downhill moves may be made.
- To do enough exploration of the whole space early on, so that the final solution is relatively insensitive to the starting state.
- Lowering the chances of getting caught at a local maximum, or plateau, or a ridge.

Simulated Annealing

Physical Annealing

- Physical substances are melted and then gradually cooled until some solid state is reached.
- The goal is to produce a minimal-energy state.
- Annealing schedule: An optimal annealing schedule for each particular annealing problem must be discovered.
- Nevertheless, there is some probability for a transition to a higher energy state: $e^{-\Delta E/kT}$.

Simulated Annealing

Algorithm:

1. Evaluate the initial state.
2. Loop until a solution is found or there are no new operators left to be applied:
   - Set T according to an annealing schedule
   - Selects and applies a new operator
   - Evaluate the new state:

     goal → quit

     ΔE = Val(current state) − Val(new state)

     if it is better than the current state, make it as new current state

     else → new current state with probability $e^{-\Delta E/T}$.

BEST-FIRST SEARCH

Combines the advantages of both DFS and BFS into a single method.

Depth-first search: not all competing branches having to be expanded.

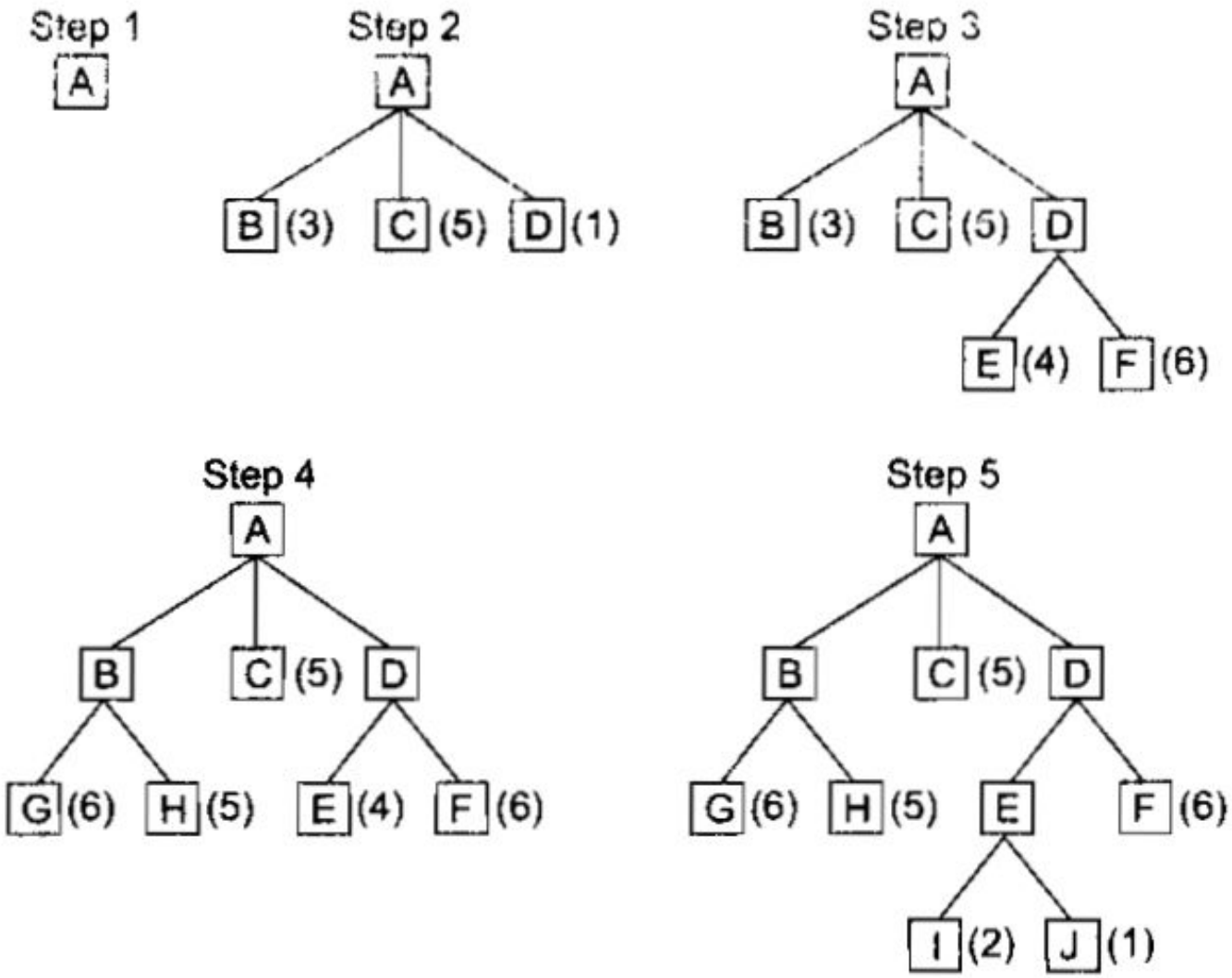
Breadth-first search: not getting trapped on dead-end paths.

Combining the two is to follow a single path at a time, but switch paths whenever some competing path look more promising than the current one

BEST-FIRST SEARCH

- At each step of the BFS search process, we select the most promising of the nodes we have generated so far.
- This is done by applying an appropriate heuristic function to each of them.
- We then expand the chosen node by using the rules to generate its successors.
- If one of them is a solution, quit.
- If not, all those new nodes are added to the set of nodes generated so far and repeat the process.
- This is called OR-graph, since each of its branches represents an alternative problem solving path.

BEST-FIRST SEARCH



BEST FIRST SEARCH VS HILL CLIMBING

Similar to Steepest ascent hill climbing with two exceptions:

- In hill climbing, one move is selected and all the others are rejected, never to be reconsidered. In BFS, one move is selected, but the others are kept around so that they can be revisited later if the selected path becomes less promising.
- The best available state is selected in the BFS, even if that state has a value that is lower than the value of the state that was just explored. Whereas in hill climbing the progress stop if there are no better successor nodes.

BEST FIRST SEARCH

To implement a graph search procedure, we will need to use two lists of nodes:

- OPEN – nodes that have been generated and have had the heuristic function applied to them but which have not yet been examined.
- CLOSED – nodes that have already been examined.
- A heuristic function that estimates the merits of each node we generate.

BEST FIRST SEARCH

Algorithm:

1. Start with *OPEN* containing just the initial state.
2. Until a goal is found or there are no nodes left on *OPEN* do:
   (a) Pick them best node on *OPEN*.
   (b) Generate its successors.
   (c) For each successor do:
      (i) If it has not been generated before, evaluate it, add it to *OPEN*, and record its parent.
      (ii) If it has been generated before, change the parent if this new path is better than the previous one. In that case, update the cost of getting to this node and to any successors that this node may already. have.

A* Algorithm

```
1.  Initialize the open list
2.  Initialize the closed list
    put the starting node on the open
    list (you can leave its f at zero)

3.  while the open list is not empty
    a) find the node with the least f on
       the open list, call it "q"

    b) pop q off the open list

    c) generate q's successors and set their parents to q

    d) for each successor
        i) if successor is the goal, stop search
          successor.g = q.g + distance between successor and q

          successor.h = distance from goal to successor

          successor.f = successor.g + successor.h

        ii) if a node with the same position as
            successor is in the OPEN list which has a
           lower f than successor, skip this successor

        iii) if a node with the same position as
            successor is in the CLOSED list which has
            a lower f than successor, skip this successor
            otherwise, add the node to the open list
     end (for loop)

    e) push q on the closed list
    end (while loop)
```

Problem Reduction

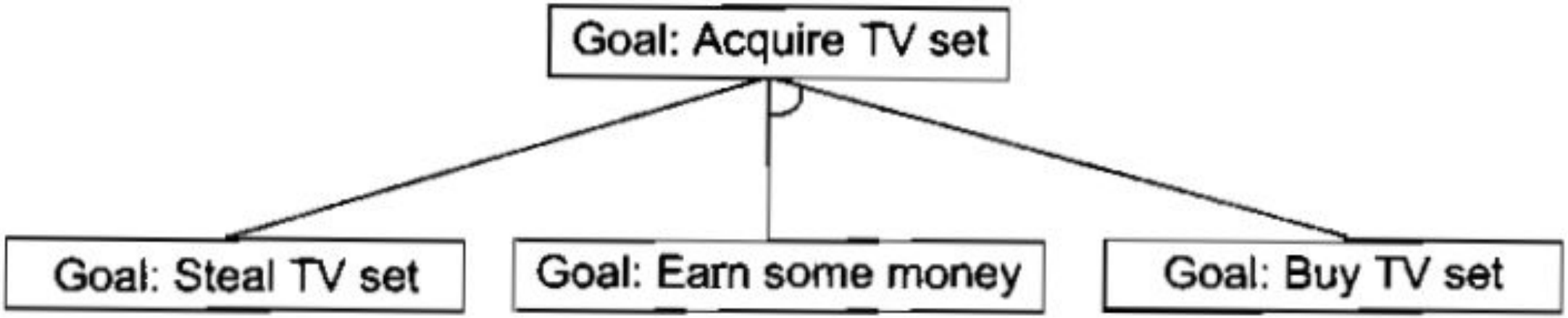
When a problem can be divided into a set of sub problems, where each sub problem can be solved separately and a combination of these will be a solution, AND-OR graphs or AND - OR trees are used for representing the solution.

The decomposition of the problem or problem reduction generates AND arcs.

One AND arc may point to any number of successor nodes. All these must be solved so that the arc will rise to many arcs, indicating several possible solutions.

Hence the graph is known as AND - OR instead of AND.

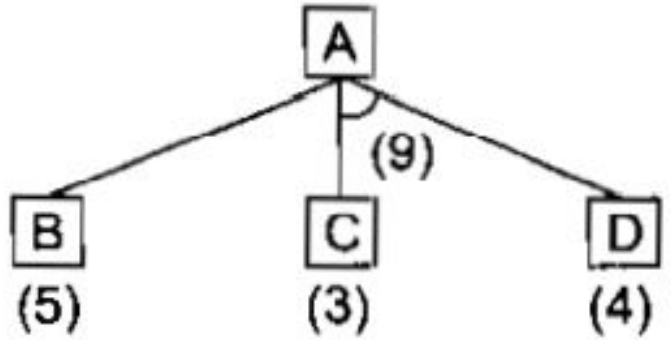
Problem Reduction



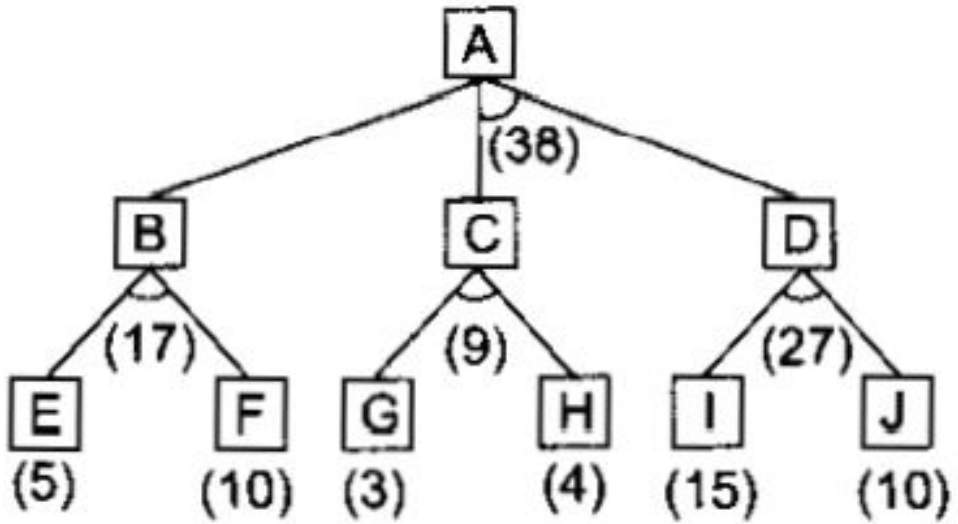
Problem Reduction

An algorithm to find a solution in an AND - OR graph must handle AND area appropriately.

A* algorithm can not search AND - OR graphs efficiently.



(a)



(b)

Problem Reduction Algorithm

1. Initialize the graph to the starting state.
2. Loop until the starting node is solved, the following three things must be done:

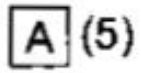
i. Traverse the graph starting at the initial node and following the current best path, and accumulate the set of nodes that are on the path and have not yet been expanded.

ii. Pick one of these unexpanded nodes and expand it. Add its successors to the graph and computer f ' (cost of the remaining distance) for each of them.

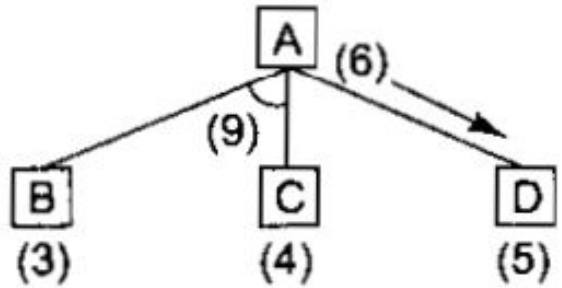
iii. Change the f ' estimate of the newly expanded node to reflect the new information produced by its successors. Propagate this change backward through the graph. Decide which of the current best path.

Problem Reduction

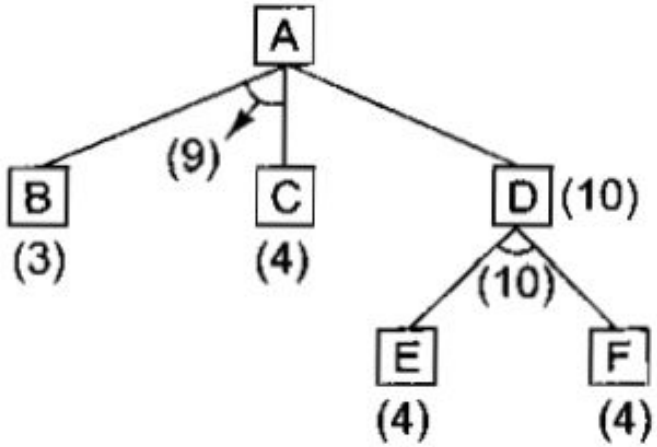
Before step 1



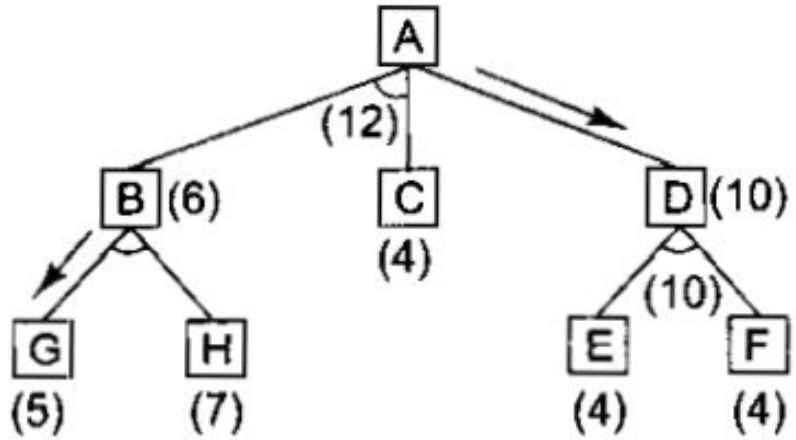
Before step 2



Before step 3



Before step 4



Problem Reduction – AO * Algorithm

Unlike A* algorithm which used two lists OPEN and CLOSED, the AO* algorithm uses a single structure G. G represents the part of the search graph generated so far.

Each node in G points down to its immediate successors and up to its immediate predecessors, and also has with it the value of h' cost of a path from itself to a set of solution nodes.

The cost of getting from the start nodes to the current node "g" is not stored as in the A* algorithm.

This is because it is not possible to compute a single such value since there may be many paths to the same state.

AO* Algorithm

1. Place the start node on open.
2. Using the search tree, compute the most promising solution tree TP .
3. Select node n that is both on open and a part of tp, remove n from open and place it no closed.
4. If n is a goal node, label n as solved. If the start node is solved, exit with success where tp is the solution tree, remove all nodes from open with a solved ancestor.
5. If n is not solvable node, label n as unsolvable. If the start node is labeled as unsolvable, exit with failure. Remove all nodes from open ,with unsolvable ancestors.
6. Otherwise, expand node n generating all of its successor compute the cost of for each newly generated node and place all such nodes on open.
7. Go back to step(2)