# Problem Solving through Searching

## Production Systems: Systems that generate (produce) rules (states) to reach a solution

**A *production system commonly consists of following four basic components:***

1. A *set of rules of the form* $C_i \rightarrow A_i$ *where* $C_i$ *refers to starting state and* $A_i$ *represents consequent state. Also* $C_i$ *the condition part and* $A_i$ *is the action* part.

2. One or more *knowledge databases that contain whatever information is relevant* for the given problem.

3. A *control strategy that ascertains the order in which the rules must be applied to the available* database

4. A *rule applier which is the computational system that implements the control* strategy and applies the rules to reach to goal (if it is possible).

## State Space

- **State space search** is a process used in which successive or *states* of an instance are considered, with the goal of finding a *goal state* with a desired property.
- State space search often differs from traditional search (sequential, indexed sequential, binary search etc) methods because the state space is *implicit*: the typical state space graph is much too large to generate and store in memory. Instead, nodes are generated as they are explored, and typically discarded thereafter.
- E.g. in a tic tack toe game, every move by a player forms a state space and the three similar (O or X) consecutive (row, column or diagonal) symbols forms goal state.
- In a chess game also state space forms a set of moves by players.
- We discuss water jug problem in the next section.

# State Space Search

The water jug problem: You are given two jugs, a 4-litre one and a 3-litre one. Neither has any measuring markers on it. There is a pump that can be used to fill the jugs with water. How can you get exactly 2 litres of water into 4-litre jug.

Let x and y be the amounts of water in 4-Lt and 3-Lt Jugs respectively

Then (x,y) refers to water available at any time in 4-Lt and 3-Lt jugs.

Also (x,y) $\longrightarrow$ (x-d,y+dd) means drop some unknown amount d of water from 4-Lt jug and add dd onto 3-Lt jug.

All possible production rules can be written as follows

| | | | |
|---|---|---|---|
| 1. | (x, y) if $x < 4$ | $\rightarrow$ (4, y) | if x<4, fill it to 4; y remains unchanged |
| 2. | (x, y) if $y < 3$ | $\rightarrow$ (x, 3) | if y<3, fill it to 3; x remains unchanged |
| 3. | (x, y) if $x > 0$ | $\rightarrow$ (x – d, y) | if there is some water in 4 Lt jug, drop some more water from it |
| 4. | (x, y) if $y > 0$ | $\rightarrow$ (x, y – d) | if there is some water in 3 Lt jug, drop some more water from it |

## State Space Search

5. $(x, y) \rightarrow (0, y)$ if there is some water in 4-Lt, empty it, y remains unchanged
   if $x > 0$
6. $(x, y) \rightarrow (x, 0)$ if there is some water in 3-Lt, empty it, x remains unchanged
   if $y > 0$
7. $(x, y) \rightarrow (4, y - (4 - x))$ if there is some water in 3-Lt, the sum of water of 4-Lt and 3-Lt jug is >=4, then fill water in 4-Lt jug to its capacity from 3-Lt jug
   if $x + y \geq 4, y > 0$
8. $(x, y) \rightarrow (x - (3 - y), 3)$ same as 7 with suitable change in x,y
   if $x + y \geq 3, x > 0$
9. $(x, y) \rightarrow (x + y, 0)$ if sum of water in both jugs <=4, then drop whole water from 3-Lt into 4-Lt
   if $x + y \leq 4, y > 0$
10. $(x, y) \rightarrow (0, x + y)$ if sum of water in both jugs <=3, then drop whole water from 4-Lt into 3-Lt
    if $x + y \leq 3, x > 0$
11. $(0, 2) \rightarrow (2, 0)$ Transfer 2-Lt from 3-Lt jug into empty 4-Lt jug
12. $(2, y) \rightarrow (0, y)$ Empty 2 Lt water onto ground from 4-Lt jug without disturbing 3 Lt jug

## State Space Search

### Solution of Water Jug Problem

Obviously to solve water jug problem, we can perform following sequence of actions,

(0,0) → (0,3) → (3,0) → (3,3) → (4,2) → (0,2) → (2,0)

By applying rules 2,9,2,7,5 and 9 with initial empty jugs

**Remember:** There is NO hard and fast rules to follow this sequence. In any state space search problem, there can be numerous ways to solve, your approach can be different to solve a problem and sequence of actions too.

## State Space Search

### Other problems

1. **Cannibals and missionaries problems:** In the missionaries (humans)and cannibals (human eaters) problem, three missionaries and three cannibals must cross a river using a boat which can carry at most two people. At any time number of cannibals on either side should not be greater than number of missionaries otherwise former will eat latter. Also The boat cannot cross the river by itself with no people on board.
2. **Tower of Hanoi Problem**: It consists of three pegs, and a number of disks ( usually 60) of different sizes which can slide onto any peg. The puzzle starts with the disks in a neat stack in ascending order of size on one rod, the smallest at the top, thus making a conical shape. The objective of the puzzle is to move the entire stack to another rod, obeying the following rules:
   - Only one disk must be moved at a time.
   - Each move consists of taking the upper disk from one of the rods and sliding it onto another rod, on top of the other disks that may already be present on that rod.
   - No disk may be placed on top of a smaller disk.
3. **Monkey Banana Problem**: A monkey is in a room. A bunch of bananas is hanging from the ceiling and is beyond the monkey's reach. However, in the room there are also a chair and a stick. The ceiling is just the right height so that a monkey standing on a chair could knock the bananas down with the stick. The monkey knows how to move around, carry other things around, reach for the bananas, and wave a stick in the air. What is the best sequence of actions for the monkey?

## 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)**
    - **– Iterative deepening (IDA)**
    - **– Bi-directional search**

**For the minimum cost path problem:**

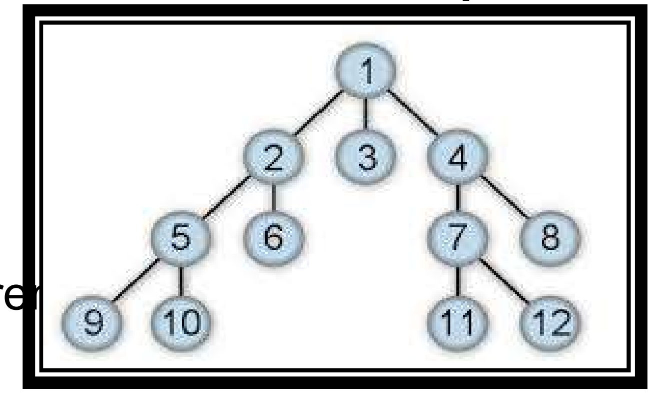
- **Uniform cost search**
- **We will discuss BFS and DFS in next section.**

## Un-informed Search Techniques

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

Space complexity Order is: O(| V |) where as time complexity is O(| V | + | E | ) a graph with V vertex vector and E Edges, |V| means cardinality of V

- **It is complete, optimal, best when space is no problem as it takes much space**



### 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.

## Un-informed Search Techniques

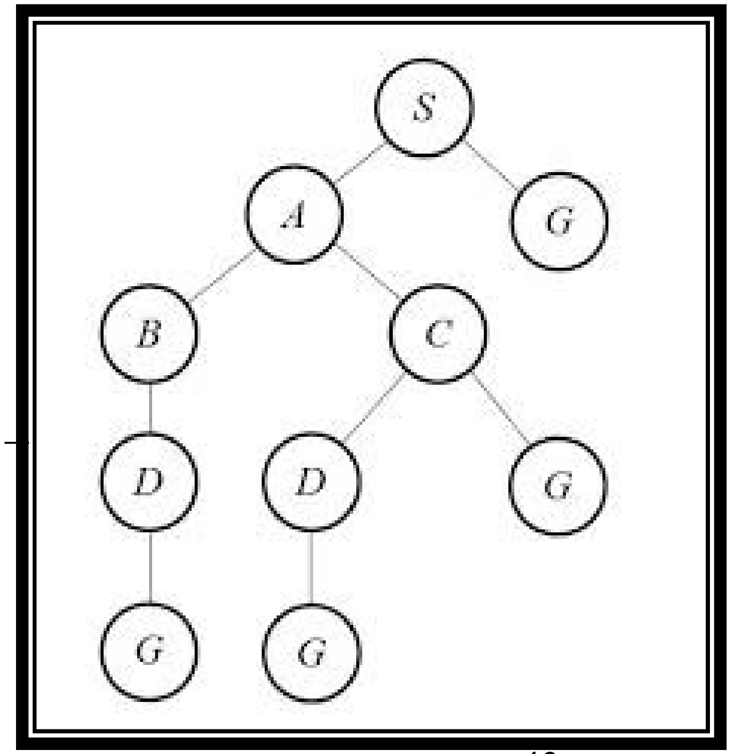
**Depth-first search (DFS):** We can start with a node and explore with all possible solutions available with this node.

**Time and Space complexity:** Time Order is: O(| V | + | E | ) , Space Order is: O(| V |)

- **It is not complete, non-optimal, may stuck in infinite loop**

**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.



# BFS for a water jug problem

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

# DFS for a water jug problem

$$\begin{tikzcd}
\boxed{(0,0)} \arrow[d] & & \\
\boxed{(4,0)} \arrow[d] & & \\
\boxed{(4,3)} \arrow[d] & & \\
\boxed{(0,3)} \arrow[d] & & \\
\boxed{(3,0)} \arrow[d] & & \\
\boxed{(3,3)} \arrow[d] & & \\
\boxed{(4,2)} \arrow[r] & \boxed{(0,2)} \arrow[r] & \boxed{(2,0)}
\end{tikzcd}$$

- **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.

- **Algorithm for Greedy Best First 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.

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.

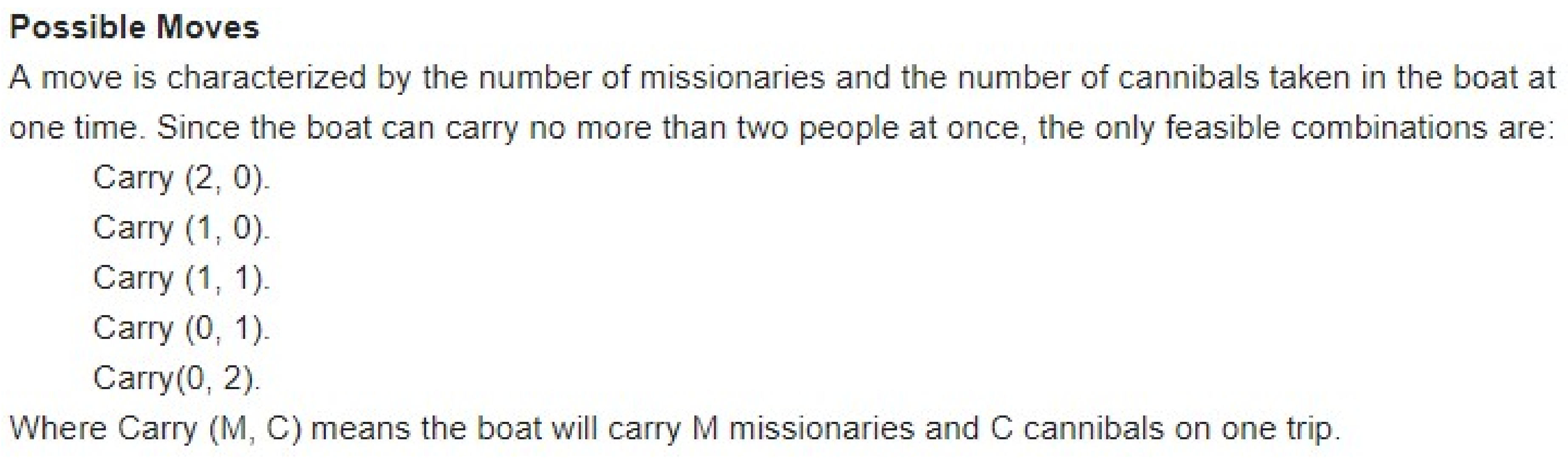
3. Repeat steps 1 and 2.

## 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



**Possible Moves**

A move is characterized by the number of missionaries and the number of cannibals taken in the boat at one time. Since the boat can carry no more than two people at once, the only feasible combinations are:

Carry (2, 0).

Carry (1, 0).

Carry (1, 1).

Carry (0, 1).

Carry(0, 2).

Where Carry (M, C) means the boat will carry M missionaries and C cannibals on one trip.

## Feasible Moves

Once we have found a possible move, we have to confirm that it is feasible. It is not a feasible to move more missionaries or more cannibals than that are present on one bank.

When the state is state(M1, C1, left) and we try carry (M,C) then

    M <= M1 and C <= C1

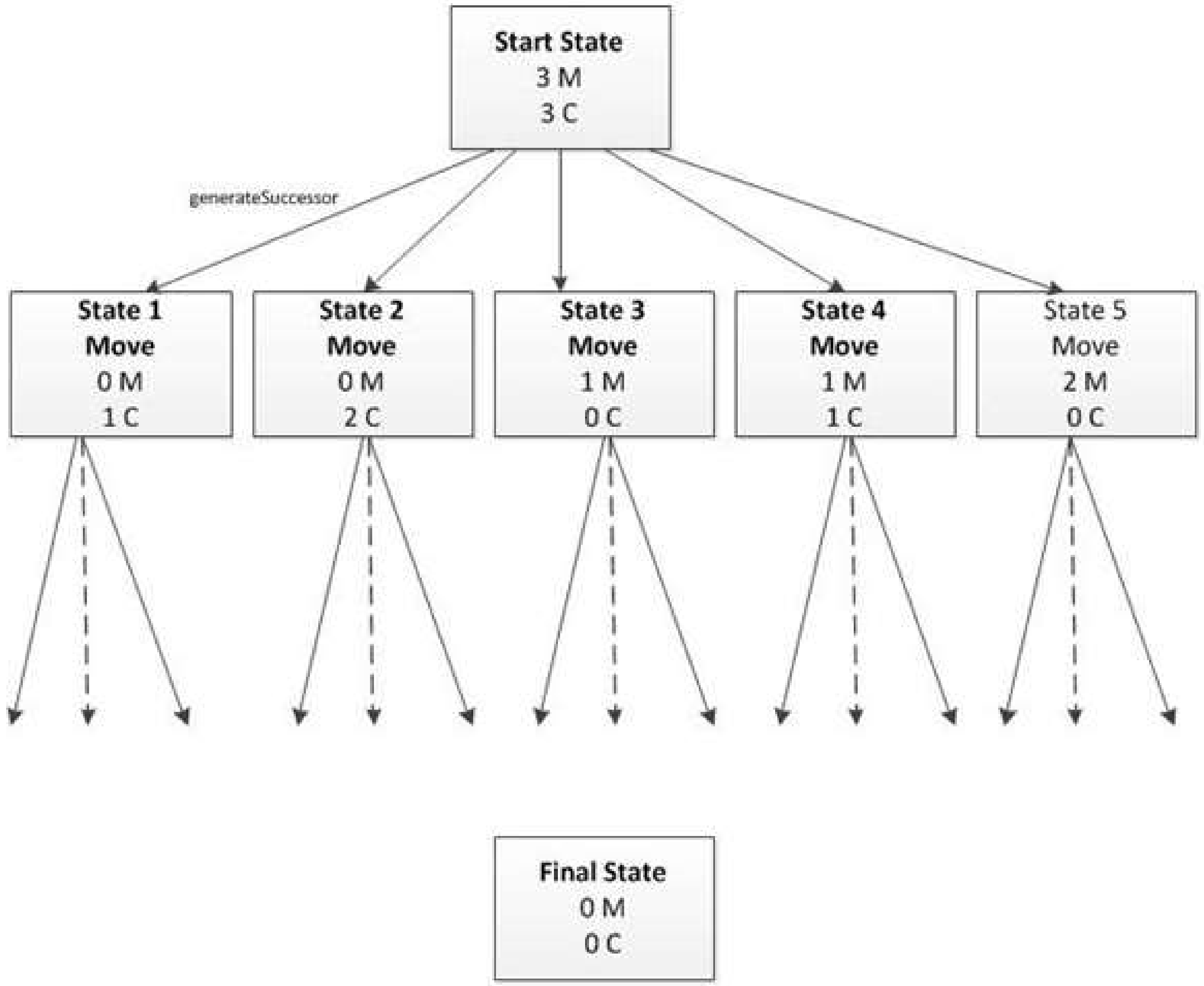
must be true.

When the state is state(M1, C1, right) and we try carry(M, C) then

    M + M1 <= 3 and C + C1 <= 3

must be true.

## Generating the next state


Start State
3 M
3 C
generateSuccessor
State 1
Move
0 M
1 C
State 2
Move
0 M
2 C
State 3
Move
1 M
0 C
State 4
Move
1 M
1 C
State 5
Move
2 M
0 C
Final State
0 M
0 C