**UNIT –2 :Problem Solving:** State-Space Search and Control Strategies: Introduction, General Problem Solving, Characteristics of Problem, Exhaustive searches, heuristic search techniques, iterative deepening a*, Constraint Satisfaction

**Problem Reduction and Game Playing**: Introduction, problem reduction, game playing, alpha-beta pruning, two-player perfect information games

## 2. Problem Solving – Basic Search methods

### Problem Characteristics

- Heuristic search is a very general method applicable to a large class of problem.
- In order to choose the most appropriate method (or combination of methods) for a particular problem it is necessary to analyze the problem along several key dimensions.
    - Is the problem decomposable into a set of independent smaller sub problems?
        - Decomposable problems can be solved by the divide-and-conquer technique.
- Use of decomposing problems:
    - Each sub-problem is simpler to solve.
    - Each sub-problem can be handed over to a different processor. Thus can be solved in parallel processing environment.
    - There are non decomposable problems.
    - For example, Block world problem is non decomposable.

### 2.1.Problem Solving

- AI programs have a clean separation of
    - computational components of data,
    - operations & control.
- Search forms the core of many intelligent processes.



www.Jntufastupdates.com

- It is useful to structure AI programs in a way that facilitates describing the search process.

**Production System – PS**

- PS is a formation for structuring AI programs which facilitates describing search process.
- It consists of
  - Initial or start state of the problem
  - Final or goal state of the problem
  - It consists of one or more databases containing information appropriate for the particular task.
- The information in databases may be structured
  - using knowledge representation schemes.

**Production Rules**

- PS contains set of production rules,
  - each consisting of a left side that determines the applicability of the rule and
  - a right side that describes the action to be performed if the rule is applied.
  - These rules operate on the databases.
  - Application of rules change the database.
- A control strategy that specifies the order in which the rules will be applied when several rules match at once.
- One of the examples of Production Systems is an Expert System.

**Advantages of PS**

- In addition to its usefulness as a way to describe search, the production model has other advantages as a formalism in AI.
  - It is a good way to model the strong state driven nature of intelligent action.



www.Jntufastupdates.com

- As new inputs enter the database, the behavior of the system changes.
- New rules can easily be added to account for new situations without disturbing the rest of the system, which is quite important in real-time environment.

**Example : Water Jug Problem**

- **Problem statement**:
  - Given two jugs, a 4-gallon and 3-gallon having no measuring markers on them. There is a pump that can be used to fill the jugs with water. How can you get exactly 2 gallons of water into 4-gallon jug.
- **Solution**:
  - State for this problem can be described as the set of ordered pairs of integers (X, Y) such that
    - X represents the number of gallons of water in 4-gallon jug and
    - Y for 3-gallon jug.
  - Start state is (0,0)
  - Goal state is (2, N) for any value of N.

**Production Rules**

- Following are the production rules for this problem.



www.Jntufastupdates.com

R1: $(X, Y \mid X < 4) \rightarrow (4, Y)$
{Fill 4-gallon jug}

R2: $(X, Y \mid Y < 3) \rightarrow (X, 3)$
{Fill 3-gallon jug}

R3: $(X, Y \mid X > 0) \rightarrow (0, Y)$
{Empty 4-gallon jug}

R4: $(X, Y \mid Y > 0) \rightarrow (X, 0)$
{Empty 3-gallon jug}

R5: $(X, Y \mid X+Y >= 4 \land Y > 0) \rightarrow (4, Y - (4 - X))$
{Pour water from 3- gallon jug into 4-gallon jug until 4-gallon jug is full}

R6: $(X, Y \mid X+Y >= 3 \land X > 0) \rightarrow (X - (3 - Y), 3)$

{Pour water from 4-gallon jug into 3-gallon jug until 3-gallon jug is full}

R7: $(X, Y \mid X+Y <= 4 \land Y > 0) \rightarrow (X+Y, 0)$

{Pour all water from 3-gallon jug into 4-gallon jug }

R8: $(X, Y \mid X+Y <= 3 \land X > 0) \rightarrow (0, X+Y)$

{Pour all water from 4-gallon jug into 3-gallon jug }

**Superficial Rules:** {May not be used in this problem}

R9: $(X, Y \mid X > 0) \rightarrow (X - D, Y)$

{Pour some water D out from 4-gallon jug}

R10: $(X, Y \mid Y > 0) \rightarrow (X, Y - D)$

{Pour some water D out from 3- gallon jug}



www.Jntufastupdates.com

**Trace of steps involved in solving the water jug problem - First solution**

| *Number of Steps* | *Rules applied* | *4-g jug* | *3-g jug* | |
|---|---|---|---|---|
| 1 | **Initial State** | 0 | 0 | |
| 2 | R2 {Fill 3-g jug} | 0 | 3 | |
| 3 | R7{Pour all water from 3 to 4-g jug } | 3 | 0 | |
| 4 | R2 {Fill 3-g jug} | 3 | 3 | |
| 5 | R5 {Pour from 3 to 4-g jug until it is full} | 4 | 2 | |
| 6 | R3 {Empty 4-gallon jug} | 0 | 2 | |
| 7 | R7 {Pour all water from 3 to 4-g jug} | 2 | 0 | **Goal State** |

**Trace of steps involved in solving the water jug problem - Second solution**

- Note that there may be more than one solutions.

| *Number of steps* | *Rules applied* | *4-g jug* | *3-g jug* | |
|---|---|---|---|---|
| 1 | **Initial State** | 0 | 0 | |
| 2 | R1 {Fill 4-gallon jug} | 4 | 0 | |
| 3 | R6 {Pour from 4 to 3-g jug until it is full } | 1 | 3 | |
| 4 | R4 {Empty 3-gallon jug} | 1 | 0 | |
| 5 | R8 {Pour all water from 4 to 3-gallon jug} | 0 | 1 | |
| 6 | R1 {Fill 4-gallon jug} | 4 | 1 | |
| 7 | R6 {Pour from 4 to 3-g jug until it is full} | 2 | 3 | |
| 8 | R4 {Empty 3-gallon jug} | 2 | 0 | **Goal State** |

- For each problem
  - there is an initial description of the problem.



www.Jntufastupdates.com

- final description of the problem.
- more than one ways of solving the problem.
- a path between various solution paths based on some criteria of goodness or on some heuristic function is chosen.
- there are set of rules that describe the actions called production rules.
  - Left side of the rules is current state and right side describes new state that results from applying the rule.

- Summary: In order to provide a formal description of a problem, it is necessary to do the following things:
  - Define a state space that contains all the possible configurations of the relevant objects.
  - Specify one or more states within that space that describe possible situations from which the problem solving process may start. These states are called **initial states.**
  - Specify one or more states that would be acceptable as solutions to the problem called **goal states.**
  - Specify a set of rules that describe the actions. Order of application of the rules is called control strategy.
  - Control strategy should cause motion towards a solution.

### 2.2 Control Strategies

- Control Strategy decides which rule to apply next during the process of searching for a solution to a problem.
- Requirements for a good Control Strategy
  - **It should cause motion**

In water jug problem, if we apply a simple control strategy of starting each time from the top of rule list and choose the first applicable one, then we will never move towards solution.



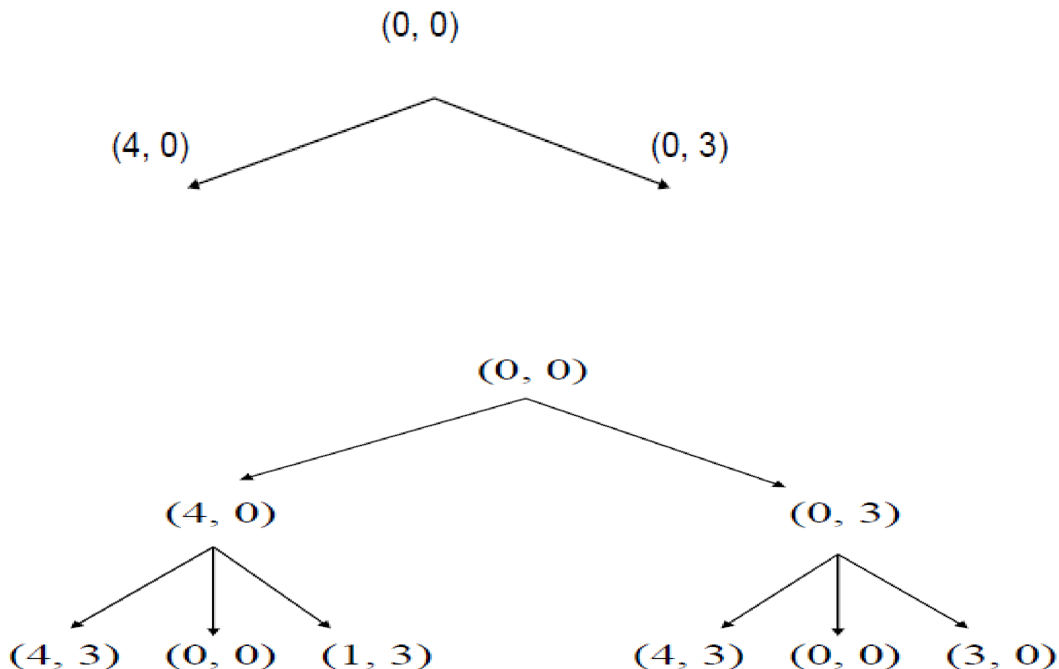
www.Jntufastupdates.com

- **It should explore the solution space in a systematic manner**

If we choose another control strategy, say, choose a rule randomly from the applicable rules then definitely it causes motion and eventually will lead to a solution. But one may arrive to same state several times. This is because control strategy is not systematic.

## Breadth First Search – BFS : Water Jug Problem

- BFS
  - Construct a tree with the initial state of the problem as its root.
  - Generate all the offspring of the root by applying each of the applicable rules to the initial state.
  - For each leaf node, generate all its successors by applying all the rules that are appropriate.
  - Repeat this process till we find a solution, if it exists.

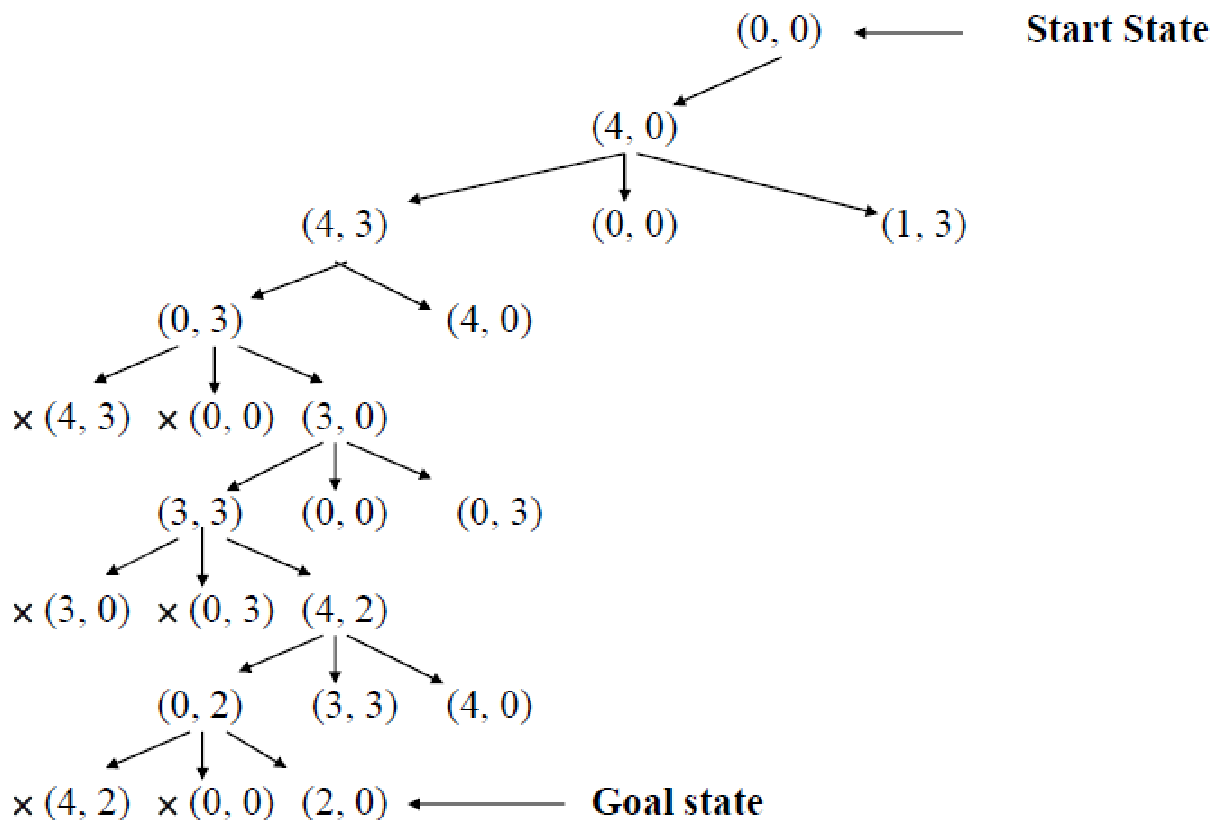





www.Jntufastupdates.com

# Depth First Search - DFS

- Here we pursue a single branch of the tree until it yields a solution or some pre-specified depth has reached.
- If solution is not found then
  - go back to immediate previous node and
  - explore other branches in DF fashion.
- Let us see the tree formation for water jug problem using DFS



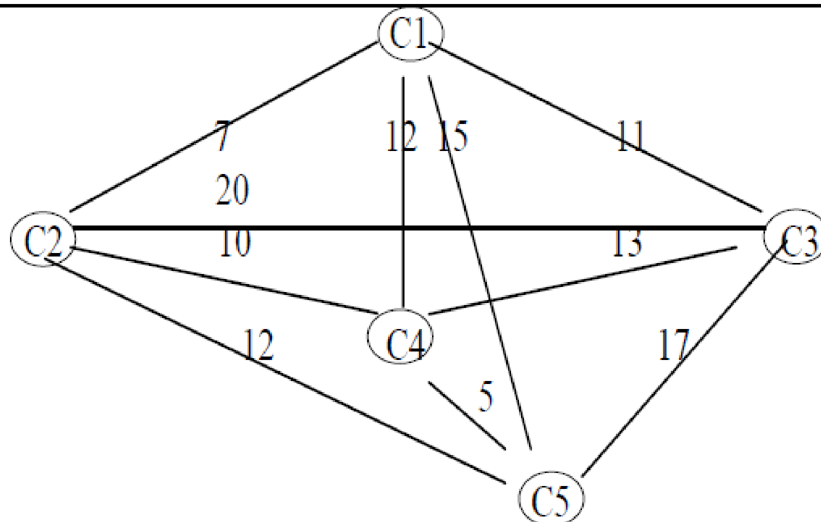



www.Jntufastupdates.com

Traveling Salesman Problem

- Consider 5 cities.
  - A salesman is supposed to visit each of 5 cities.
  - All cities are pair wise connected by roads.
  - There is one start city.
  - The problem is to find the shortest route for the salesman who has to
    - visit each city only once and
    - returns to back to start city.

**Traveling Salesman Problem (Example) – Start city is C1**




D(C1,C2) = 7; D(C1,C3) = 11; D(C1,C4) = 12; D(C1,C5) = 15; D(C2,C3) =20;

D(C2,C4) = 10; D(C2,C5) =12; D(C3,C4) =13; D(C3,C5) = 17; D(C4,C5) =5;

- A simple motion causing and systematic control structure could, in principle solve this problem.
- Explore the search tree of all possible paths and return the shortest path.
- This will require 4! paths to be examined.



www.Jntufastupdates.com

- If number of cities grow, say 25 cities, then the time required to wait a salesman to get the information about the shortest path is of 0(24!) which is not a practical situation.
- This phenomenon is called **combinatorial explosion**.
- We can improve the above strategy as follows:
    - Begin generating complete paths, keeping track of the shortest path found so far.
    - Give up exploring any path as soon as its partial length becomes greater than the shortest path found so far.
    - This algorithm is efficient than the first one, still requires exponential time $\propto$ some number raised to N (number of cities).

**a. Missionaries and Cannibals**

- *Problem Statement:* Three missionaries and three cannibals want to cross a river. There is a boat on their side of the river that can be used by either one or two persons.
    - How should they use this boat to cross the river in such a way that cannibals never outnumber missionaries on either side of the river? If the cannibals ever outnumber the missionaries (on either bank) then the missionaries will be eaten. How can they all cross over without anyone being eaten?
- PS for this problem can be described as the set of ordered pairs of left and right bank of the river as (L, R) where each bank is represented as a list [nM, mC, B]
    - n is the number of missionaries M, m is the number of cannibals C, and B represents boat.
- Start state: ( [3M, 3C, 1B], [0M, 0C, 0B] ),
    - 1B means that boat is present and 0B means it is not there on the bank of river.
- Goal state: ( [0M, 0C, 0B], [3M, 3C, 1B] )



www.Jntufastupdates.com

- Any state: ([$n_1$M, $m_1$C, 1B], [$n_2$ M, $m_2$ C, 0B]) , with constraints/conditions as $n_1$ ($\neq$0) $\geq$ $m_1$; $n_2$ ($\neq$0) $\geq$ $m_2$; $n_1 + n_2 = 3$, $m_1 + m_2 = 3$
  - By no means, this representation is unique.
  - In fact one may have number of different representations for the same problem.
  - The table on the next slide consists of production rules based on the chosen representation.

**Set of Production Rules**

Applied keeping constrains in mind

| RN | Left side of rule | → | Right side of rule |
|---|---|---|---|
| | *Rules for boat going from left bank to right bank of the river* | | |
| L1 | ([$n_1$M, $m_1$C, 1B], [$n_2$M, $m_2$C, 0B]) | → | ([($n_1$-2)M, $m_1$C, 0B], [($n_2$+2)M, $m_2$C, 1B]) |
| L2 | ([$n_1$M, $m_1$C, 1B], [$n_2$M, $m_2$C, 0B]) | → | ([($n_1$-1)M,($m_1$-1)C,0B],[($n_2$+1)M,($m_2$+1)C, 1B]) |
| L3 | ([$n_1$M, $m_1$C, 1B], [$n_2$M, $m_2$C, 0B]) | → | ([$n_1$M, ($m_1$-2)C, 0B], [$n_2$M, ($m_2$+2)C, 1B]) |
| L4 | ([$n_1$M, $m_1$C, 1B], [$n_2$M, $m_2$C, 0B]) | → | ([($n_1$-1)M, $m_1$C,0B],[($n_2$+1)M, $m_2$C, 1B]) |
| L5 | ([$n_1$M, $m_1$C, 1B], [$n_2$M, $m_2$C, 0B]) | → | ([$n_1$M, ($m_1$-1)C, 0B], [$n_2$M, ($m_2$+1)C, 1B]) |
| | *Rules for boat coming from right bank to left bank of the river* | | |
| R1 | ([$n_1$M, $m_1$C, 0B], [$n_2$M, $m_2$C, 1B]) | → | ([($n_1$+2)M, $m_1$C, 1B], [($n_2$-2)M, $m_2$C, 0B]) |
| R2 | ([$n_1$M, $m_1$C, 0B], [$n_2$M, $m_2$C, 1B]) | → | ([($n_1$+1)M,($m_1$+1)C,1B],[($n_2$-1)M,($m_2$-1)C, 0B]) |
| R3 | ([$n_1$M, $m_1$C, 0B], [$n_2$M, $m_2$C, 1B]) | → | ([$n_1$M, ($m_1$+2)C, 1B], [$n_2$M, ($m_2$-2)C, 0B]) |
| R4 | ([$n_1$M, $m_1$C, 0B], [$n_2$M, $m_2$C, 1B]) | → | ([($n_1$+1)M, $m_1$C,1B],[($n_2$-1)M, $m_2$C, 0B]) |
| R5 | ([$n_1$M, $m_1$C, 0B], [$n_2$M, $m_2$C, 1B]) | → | ([$n_1$M, ($m_1$+1)C, 1B], [$n_2$M, ($m_2$-1)C, 0B]) |



www.Jntufastupdates.com

One of the possible paths

| Start → | ([3M, 3C, 1B], [0M, 0C, 0B]] | |
|---|---|---|
| L2: | ([2M, 2C, 0B], [1M, 1C, 1B]) | 1M,1C → |
| R4: | ([3M, 2C, 1B], [0M, 1C, 0B]) | 1M ← |
| L3: | ([3M, 0C, 0B], [0M, 3C, 1B]) | 2C → |
| R4: | ([3M, 1C, 1B], [0M, 2C, 0B]) | 1C ← |
| L1: | ([1M, 1C, 0B], [2M, 2C, 1B]) | 2M → |
| R2: | ([2M, 2C, 1B], [1M, 1C, 0B]) | 1M,1C ← |
| L1: | ([0M, 2C, 0B], [3M, 1C, 1B]) | 2M → |
| R5: | ([0M, 3C, 1B], [3M, 0C, 0B]) | 1C ← |
| L3: | ([0M, 1C, 0B], [3M, 2C, 1B]) | 2C → |
| R5: | ([0M, 2C, 1B], [3M, 1C, 0B]) | 1C ← |
| L3: | ([0M, 0C, 0B], [3M, 3C, 1B]) | 2C → Goal state |

### 2.3 State Space Search for Solving problems

- State space is another method of problem representation that facilitates easy search similar to PS.
- In this method also problem is viewed as finding a path from start state to goal state.
- A solution path is a path through the graph from a node in a set S to a node in set G.
- Set S contains start states of the problem.
- A set G contains goal states of the problem.
- The aim of search algorithm is to determine a solution path in the graph.
- A state space consists of four components.
    - Set of nodes (states) in the graph/tree. Each node represents the state in problem solving process.



www.Jntufastupdates.com

- Set of arcs connecting nodes. Each arc corresponds to operator that is a step in a problem solving process.
- Set S containing start states of the problem.
- Set G containing goal states of the problem.

### 2.3.1.The 8-Puzzle

*Problem Statement*:

- The eight puzzle problem consists of a 3 x 3 grid with 8 consecutively numbered tiles arranged on it.
- Any tile adjacent to the space can be moved on it.
- Solving this problem involves arranging tiles in the goal state from the start state.

Start state

| | | |
|---|---|---|
| 3 | 7 | 6 |
| 5 | 1 | 2 |
| 4 | □ | 8 |

Goal state

| | | |
|---|---|---|
| 5 | 3 | 6 |
| 7 | □ | 2 |
| 4 | 1 | 8 |

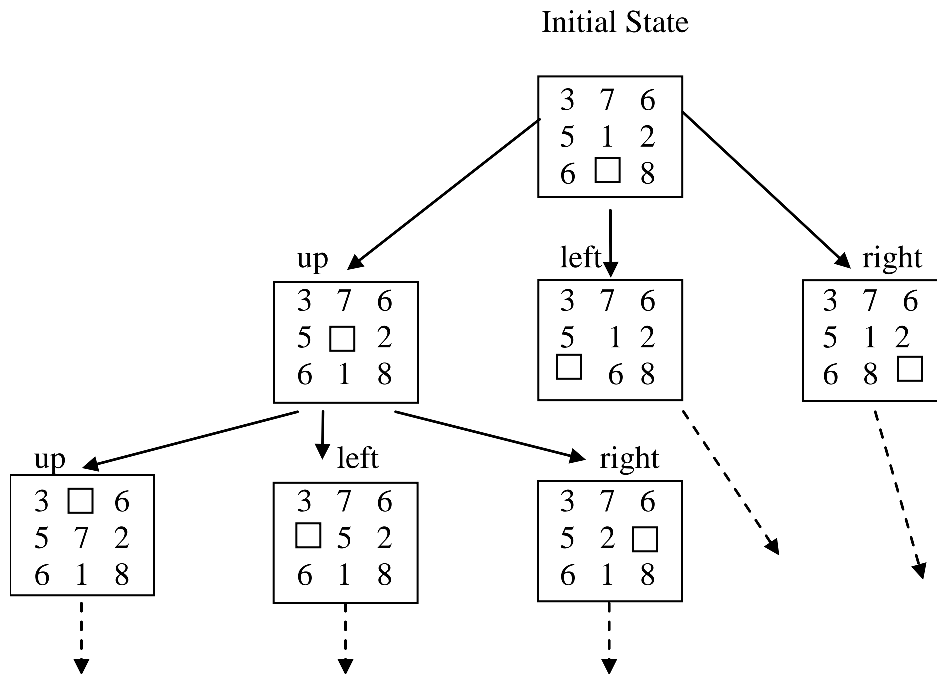
Solution by State Space method

- The start state could be represented as: [ [3,7,2], [5,1, 2], [4,0,6] ]
- The goal state could be represented as: [ [5,3,6] [7,0,2], [4,1,8] ]

The operators can be thought of moving {up, down, left, right}, the direction in which blank space effectively moves.



www.Jntufastupdates.com




**Searching for a Solution**

- Problem can be solved by searching for a solution.
- Transform initial state of a problem into some final goal state.
- Problem can have more than one intermediate states between start and goal states.
- All possible states of the problem taken together are said to form
    - a **state space** or
    - problem state and
    - search is called **state space search**.
- Search is basically a procedure to discover a path through a problem space from initial state to a goal state.
- There are two directions in which such a search could proceed.
    - Data driven search, **forward**, from the start state

Goal driven search, **backward**, from the goal state



www.Jntufastupdates.com

### 2.3.2.Forward Reasoning (Chaining):

- It is a control strategy that starts with known facts and works towards a conclusion.
- For example in 8 puzzle problem, we start from initial state to goal state.
- In this case we begin building a tree of move sequences with initial state as the root of the tree.
- Generate the next level of the tree by finding all rules whose left sides match with root and use their right side to create the new state.
- Continue until a configuration that matches the goal state is generated.
- Language OPS5 uses forward reasoning rules. Rules are expressed in the form of “if-then rule”.
- Find out those sub-goals which could generate the given goal.

### 2.3.3. Backward Reasoning (Chaining)

- It is a goal directed control strategy that begins with the final goal.
- Continue to work backward, generating more sub goals that must also be satisfied in order to satisfy main goal.
- Prolog (Programming in Logic) uses this strategy.

### 2.4. General Purpose Search Strategies:

- **Breadth First Search (BFS)**
  - It expands all the states one step away from the initial state, then expands all states two steps from initial state, then three steps etc., until a goal state is reached.
  - It expands all nodes at a given depth before expanding any nodes at a greater depth.
  - All nodes at the same level are searched before going to the next level down.
  - For implementation, two lists called OPEN and CLOSED are maintained.



www.Jntufastupdates.com

- The OPEN list contains those states that are to be expanded and CLOSED list keeps track of states already expanded.
- Here OPEN list is used as a queue.

**Algorithm (BFS)**

Input: Two states in the state space START and GOAL

Local Variables: OPEN, CLOSED, STATE-X, SUCCS

Output: Yes or No

Method:

- Initially OPEN list contains a START node and CLOSED list is empty; Found = false;
- While (OPEN ≠ empty and Found = false)

Do {

- Remove the first state from OPEN and call it STATE-X;
- Put STATE-X in the front of CLOSED list;
- If STATE-X = GOAL then Found = true else

{- perform EXPAND operation on STATE-X, producing a list of SUCCESSORS;

- Remove from successors those states, if any, that are in the CLOSED list;

- Append SUCCESSORS at the end of the OPEN list /*queue*/

} } /* end while */

If Found = true then return Yes else return No and Stop

**Depth-First Search**

- In depth-first search we go as far down as possible into the search tree / graph before backing up and trying alternatives.
  - It works by always generating a descendent of the most recently expanded node until some depth cut off is reached



www.Jntufastupdates.com

    - then backtracks to next most recently expanded node and generates one of its descendants.

- So only path of nodes from the initial node to the current node is stored in order to execute the algorithm.
- For implementation, two lists called OPEN and CLOSED with the same conventions explained earlier are maintained.
    - Here OPEN list is used as a stack.
    - If we discover that first element of OPEN is the Goal state, then search terminates successfully else move it to closed list and stack its successor in open list.

**Algorithms (DFS)**

**Input**: Two states in the state space, START and GOAL

**LOCAL Variables**: OPEN, CLOSED, RECORD-X, SUCCESSORS

**Output:** A path sequence if one exists, otherwise return No

**Method:**

- Form a stack consisting of (START, nil) and call it OPEN list. Initially set CLOSED list as empty; Found = false;
- While (OPEN ≠ empty and Found = false) DO

{

- Remove the first state from OPEN and call it RECORD-X;
- Put RECORD-X in the front of CLOSED list;
- If the state variable of RECORD-X= GOAL,

then **Found = true**

Else

{ - Perform EXPAND operation on STATE-X, a state variable of RECORD-X, producing a list of action records called SUCCESSORS; create each action record by associating with each state its parent.



www.Jntufastupdates.com

- Remove from SUCCESSORS any record whose state variables are in the record already in the CLOSED list.

- Insert SUCCESSORS in the front of the OPEN list /* Stack */

}

}/* end while */

- If Found = true then return the plan used /* find it by tracing through the pointers on the CLOSED list */ else return **No**
- Stop

**Comparisons**

- **DFS**
  - is effective when there are few sub trees in the search tree that have only one connection point to the rest of the states.
  - can be dangerous when the path closer to the START and farther from the GOAL has been chosen.
  - Is best when the GOAL exists in the lower left portion of the search tree.
  - Is effective when the search tree has a low branching factor.
- **BFS**
  - can work even in trees that are infinitely deep.
  - requires a lot of memory as number of nodes in level of the tree increases exponentially.
  - is superior when the GOAL exists in the upper right portion of a search tree.

**Depth First Iterative Deepening (DFID)**

- DFID is an iterative method that expands all nodes at a given depth before expanding any nodes at greater depth.



www.Jntufastupdates.com

- For a given depth d, DFID performs a DFS and never searches deeper than depth d and d is increased by 1 in next iteration if solution is not found.
- Advantages:
  - It takes advantages of both the strategies (BFS & DFS) and suffers neither the drawbacks of BFS nor of DFS on trees
  - It is guaranteed to find a shortest - length (path) solution from initial state to goal state (same as BFS).
  - Since it is performing a DFS and never searches deeper than depth d. the space it uses is O(d) (same as DFS).
- Disadvantages:
  - DFID performs wasted computation prior to reaching the goal depth but time complexity remains same as that of BFS and DFS

## 2.5. Various Heuristic Searches:

### Heuristic Search

- Heuristics are criteria for deciding which among several alternatives be the most effective in order to achieve some goal.
- Heuristic is a technique that
  - improves the efficiency of a search process possibly by sacrificing claims of systematicity and completeness.
  - It no longer guarantees to find the best answer but almost always finds a very good answer.
  - Using good heuristics, we can hope to get good solution to hard problems (such as travelling salesman) in less than exponential time.
  - There are **general-purpose** heuristics that are useful in a wide variety of problem domains.
  - We can also construct **special purpose** heuristics, which are domain specific.



www.Jntufastupdates.com

### 2.5.1.General Purpose Heuristics

- A general-purpose heuristics for combinatorial problem is
  - **Nearest neighbor algorithms** which works by selecting the locally superior alternative.
  - For such algorithms, it is often possible to prove an upper bound on the error which provide reassurance that one is not paying too high a price in accuracy for speed.
- In many AI problems,
  - it is often hard to measure precisely the goodness of a particular solution.
  - But still it is important to keep performance question in mind while designing algorithm.
- For real world problems,
  - it is often useful to introduce heuristics based on relatively unstructured knowledge.
  - It is impossible to define this knowledge in such a way that mathematical analysis can be performed.
- In AI approaches,
  - behavior of algorithms are analyzed by running them on computer as contrast to analyzing algorithm mathematically.
- There are at least two reasons for the adhoc approaches in AI.
  - It is a lot more fun to see a program do something intelligent than to prove it.
  - AI problem domains are usually sufficiently complex, so generally not possible to produce analytical proof that a procedure will work.
  - It is even not possible to describe the range of problems well enough to make statistical analysisof program behavior meaningful.
- One of the most important analysis of the search process is straightforward i.e.,



www.Jntufastupdates.com

   - Number of nodes in a complete search tree of depth D and branching factor F is F*D .

- This simple analysis motivates to
   - look for improvements on the exhaustive search.
   - find an upper bound on the search time which can be compared with exhaustive search procedures.

**2.5.2. Informed Search Strategies- Branch & Bound Search**

- It expands the least-cost partial path. Sometimes, it is called **uniform cost search**.
- **Function g(X)** assigns some cumulative expense to the path from *Start* node to *X* by applying the sequence of operators .
   - *For example*, in salesman traveling problem, g(X) may be the actual distance from *Start* to current node *X*.
- During search process there are many incomplete paths contending for further consideration.
- The shortest one is extended one level, creating as many new incomplete paths as there are branches.
- These new paths along with old ones are sorted on the values of function **g**.
- The shortest path is chosen and extended.
- Since the shortest path is always chosen for extension, the path first reaching to the destination is certain to be nearly optimal.
- Termination Condition:
   - Instead of terminating when a path is found, terminate when the shortest incomplete path is longer than the shortest complete path.
- If g(X) = 1, for all operators, then it degenerates to simple Breadth-First search.
- It is as bad as depth first and breadth first, from AI point of view,.



www.Jntufastupdates.com

- This can be improved if we augment it by dynamic programming i.e. delete those paths which are redundant.

Hill Climbing- **(Quality Measurement turns DFS into Hill climbing (Variant of generate and test strategy)**

- Search efficiency may be improved if there is some way of ordering the choices so that the most promising node is explored first.
- Moving through a tree of paths, hill climbing proceeds
  - in depth-first order but the choices are ordered according to some heuristic value (i.e, measure of remaining cost from current to goal state).

**2.5.3. Hill Climbing- Algorithm:**

**Generate and Test** *Algorithm*

***Start***

- Generate a possible solution

- Test to see, if it is goal.

- If not go to start else quit

***End***

**Example of heuristic function**

- Straight line (as the crow flies) distance between two cities may be a heuristic measure of remaining distance in traveling salesman problem .

**Simple Hill climbing : Algorithm**

- Store initially, the root node in a OPEN list (maintained as stack) ; Found = false;
- While ( OPEN ≠ empty and Found = false) Do

{

  - Remove the top element from OPEN list and call it NODE;



www.Jntufastupdates.com

- If NODE is the goal node, then Found = true else find SUCCs, of NODE, if any, and sort SUCCs by estimated cost from NODE to goal state and add them to the front of OPEN list.

} /* end while */

- If Found = true then return Yes otherwise return No
- Stop

**Problems in hill climbing:**

- There might be a position that is not a solution but from there no move improves situations?
- This will happen if we have reached a *Local maximum*, a *plateau* or a *ridge.*
  - **Local maximum**: It is a state that is better than all its neighbors but is not better than some other states farther away. All moves appear to be worse.

*Solution to this is to backtrack to some earlier state and try going in different direction.*

  - **Plateau:** It is a flat area of the search space in which, a whole set of neighboring states have the same value. It is not possible to determine the best direction.
    - *Here make a big jump to some direction and try to get to new section of the search space.*
  - **Ridge:** It is an area of search space that is higher than surrounding areas, but that can not be traversed by single moves in any one direction. (Special kind of local maxima).
    - *Here apply two or more rules before doing the test i.e., moving in several directions at once.*

**2.5.4. Beam Search:**

- Beam Search progresses level by level.
- It moves downward from the best W nodes only at each level. Other nodes are ignored.



www.Jntufastupdates.com

- W is called width of beam search.

- It is like a BFS where also expansion is level wise.
- Best nodes are decided on the heuristic cost associated with the node.
- If B is the branching factor, then there will be only W*B nodes under consideration at any depth but only W nodes will be selected.

**Algorithm – Beam search**

- Found = false;
- NODE = Root_node;
- If NODE is the goal node, then *Found = true* else find SUCCs of NODE, if any with its estimated cost and store in OPEN list;
- While (Found = false and not able to proceed further)

    {

    - Sort OPEN list;
    - Select top W elements from OPEN list and put it in W_OPEN list and empty OPEN list;
    - While (W_OPEN ≠ ϕ and Found = false)

    {

        - Get NODE from W_OPEN;
        - If NODE = Goal state then Found = true else

    {

            - Find SUCCs of NODE, if any with its estimated cost
            - store in OPEN list;

        }

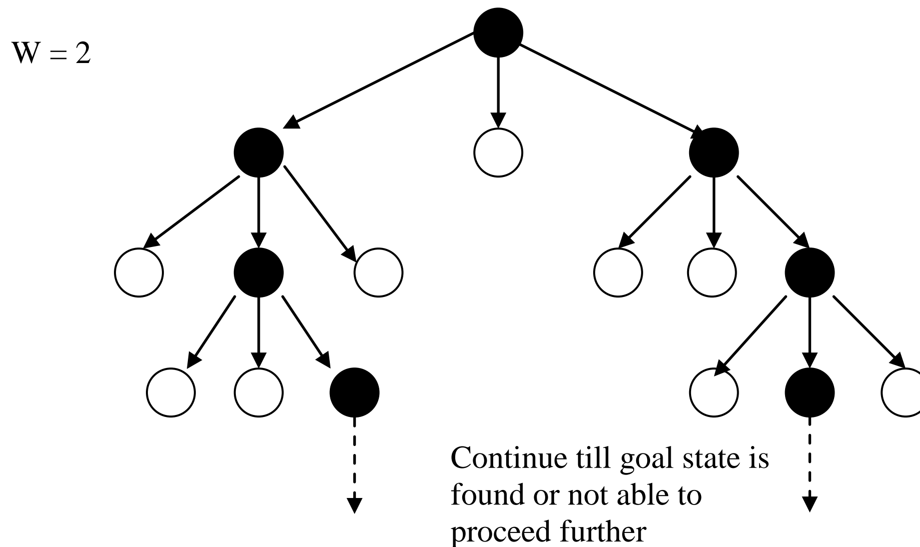
    } // end while

} // end while

- If *Found = true* then return *Yes* otherwise return *No* and Stop



www.Jntufastupdates.com




### 2.5.5. Best First Search

- Expand the best partial path.
- Here forward motion is carried out from the best open node so far in the entire partially developed tree.

**Algorithm (Best First Search)**

- Initialize OPEN list by root node; CLOSED = ϕ;
- Found = false;
- While (OPEN ≠ ϕ and Found = false) Do

  {

  - If the first element is the goal node, then Found = true else remove it from OPEN list and put it in CLOSED list.
  - Add its successor, if any, in OPEN list.
  - Sort the entire list by the value of some heuristic function that assigns to each node, the estimate to reach to the goal node

  } /* end while */

- If the Found = true, then announce the success else announce failure.



www.Jntufastupdates.com

- Stop.

**Observations**

- In hill climbing, sorting is done on the successors nodes whereas in the best first search sorting is done on the entire list.
- It is not guaranteed to find an optimal solution, but normally it finds some solution faster than any other methods.
- The performance varies directly with the accuracy of the heuristic evaluation function.

**Termination Condition**

- Instead of terminating when a path is found, terminate when the shortest incomplete path is longer than the shortest complete path.

**2.5.6.A* Method**

- A* (“Aystar”) (Hart, 1972) method is a combination of branch & bound and best search, combined with the dynamic programming principle.
- The heuristic function (or Evaluation function) for a node N is defined as $f(N) = g(N) + h(N)$
- The function g is a measure of the cost of getting from the Start node (initial state) to the current node.
  - It is sum of costs of applying the rules that were applied along the best path to the current node.
- The function h is an estimate of additional cost of getting from the current node to the Goal node (final state).
  - Here knowledge about the problem domain is exploited.
- A* algorithm is called OR graph / tree search algorithm.

**Algorithm (A*)**

- Initialization OPEN list with initial node; CLOSED= $\phi$; g = 0, f = h, Found = false;



www.Jntufastupdates.com

- While (OPEN ≠ ϕ and Found = false )

  {

  - Remove the node with the lowest value of f from OPEN to CLOSED and call it as a Best_Node.
  - If Best_Node = Goal state then Found = true else

  {

    - Generate the Succ of Best_Node
    - For each Succ do

  {

      - Compute g(Succ) = g(Best_Node) + cost of getting from Best_Node to Succ.
      - If Succ ∈ CLOSED then /* already processed */

{

        - Call the matched node as OLD and add it in the list of Best_Node successors.
        - Ignore the Succ node and change the parent of OLD, if required

- If g(Succ) < g(OLD) then make parent of OLD to be Best_Node and change the values of g and f for OLD.

- Propogate the change to OLD's children using depth first search

- If g(Succ) >= g(OLD) then do nothing

}

      - If Succ ∉ OPEN or CLOSED

  {

        - Add it to the list of Best_Node's successors
        - Compute f(Succ) = g(Succ) + h(Succ)
        - Put Succ on OPEN list with its f value



www.Jntufastupdates.com

```
            }
            } /* for loop*/
      } /* else if */
     } /* End while */
```

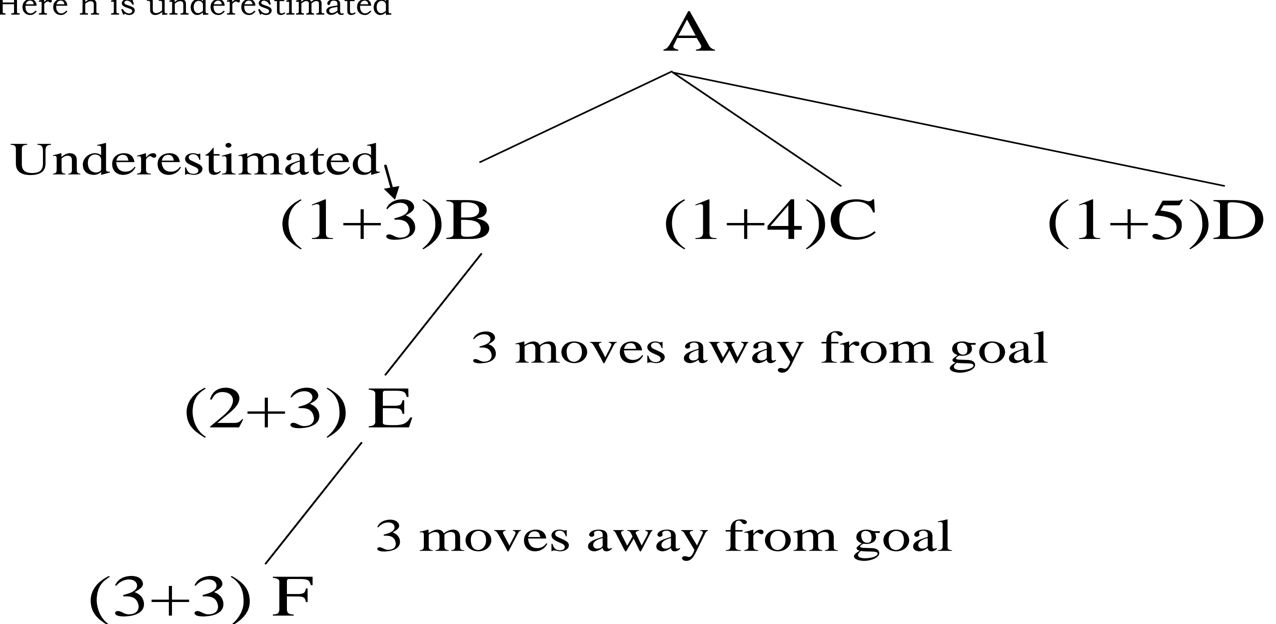
- If Found = true then report the best path else report failure
- Stop

**Behavior of A* Algorithm**

**Underestimation**

If we can guarantee that h never over estimates actual value from current to goal, then A* algorithm is guaranteed to find an optimal path to a goal, if one exists

Example – Underestimation – f=g+h
Here h is underestimated



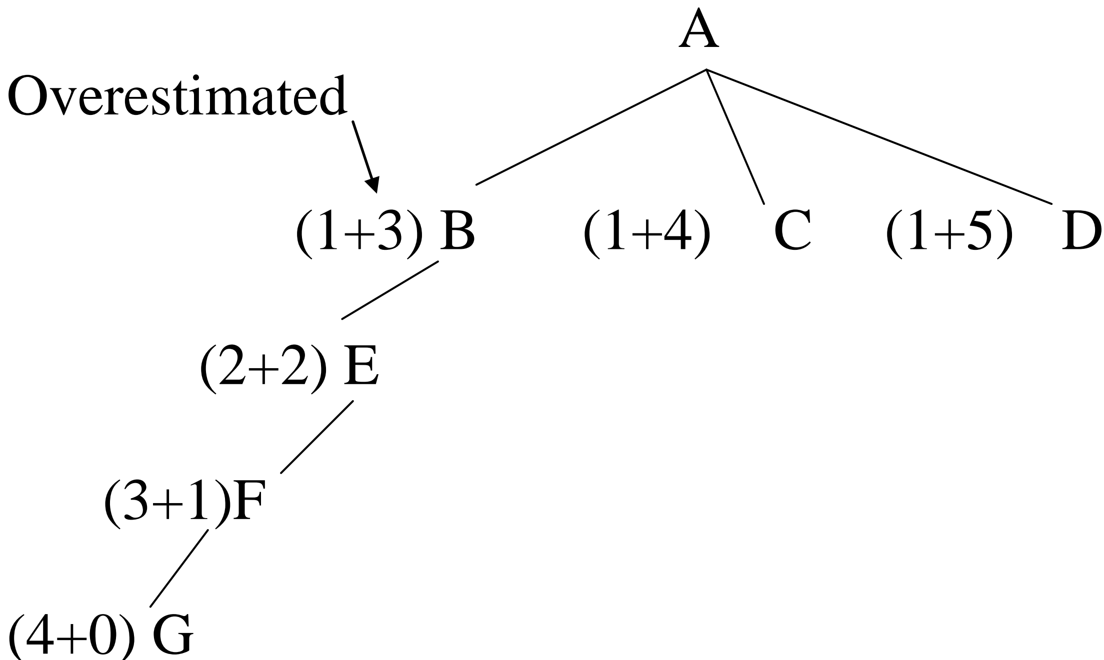

Explanation -E**xample of Underestimation**

- Assume the cost of all arcs to be 1. A is expanded to B, C and D.
- 'f' values for each node is computed.



www.Jntufastupdates.com

- B is chosen to be expanded to E.
- We notice that f(E) = f(C) = 5
- Suppose we resolve in favor of E, the path currently we are expanding. E is expanded to F.
- Clearly expansion of a node F is stopped as f(F)=6 and so we will now expand C.
- Thus we see that by underestimating h(B), we have wasted some effort but eventually discovered that B was farther away than we thought.
- Then we go back and try another path, and will find optimal path.




**Explanation –Example of Overestimation**

- A is expanded to B, C and D.
- Now B is expanded to E, E to F and F to G for a solution path of length 4.
- Consider a scenario when there a direct path from D to G with a solution giving a path of length 2.



www.Jntufastupdates.com

- We will never find it because of overestimating h(D).
- Thus, we may find some other worse solution without ever expanding D.
- So by overestimating h, we can not be guaranteed to find the cheaper path solution.

**Admissibility of A*:**

- A search algorithm is admissible, if
  - for any graph, it always terminates in an optimal path from initial state to goal state, if path exists.
- If heuristic function h is underestimate of actual value from current state to goal state, then the it is called admissible function.
- Alternatively we can say that A* always terminates with the optimal path in case
  - h(x) is an admissible heuristic function.

**Monotonicity**

- A heuristic function h is monotone if
  - $\forall$ states Xi and Xj such that Xj is successor of Xi

    $$h(Xi) - h(Xj) \leq cost(Xi, Xj)$$

    where, cost (Xi, Xj) actual cost of going from Xi to Xj
  - h (goal) = 0
- In this case, heuristic is locally admissible i.e., consistently finds the minimal path to each state they encounter in the search.
- The monotone property in other words is that search space which is every where locally consistent with heuristic function employed i.e., reaching each state along the shortest path from its ancestors.
- With monotonic heuristic, if a state is rediscovered, it is not necessary to check whether the new path is shorter.
- Each monotonic heuristic is admissible
- A cost function f(n) is monotone if $f(n) \leq f(succ(n)), \forall n$.



www.Jntufastupdates.com

- For any admissible cost function f, we can construct a monotone admissible function.
- Alternatively, the monotone property:
- that search space which is every where locally consistent with heuristic function employed i.e., reaching each state along the shortest path from its ancestors.
- With monotonic heuristic, if a state is rediscovered, it is not necessary to check whether the new path is shorter.
- Each monotonic heuristic is admissible
- A **cost function** f(n) is monotone. if f(n) ≤f(succ(n)), ∀n.
- For any admissible cost function f, we can construct a monotone admissible function.

**Example:** Solve Eight puzzle problem using A* algorithm

- **Start state**

| | | |
|---|---|---|
| **3** | **7** | **6** |
| **5** | **1** | **2** |
| **4** | □ | **8** |

**Goal state**

| | | |
|---|---|---|
| **5** | **3** | **6** |
| **7** | □ | **2** |
| **4** | **1** | **8** |

- Evaluation function f (X) = g (X) + h(X)

  h (X) = the number of tiles not in their goal position in a given state X

  g(X) = depth of node X in the search tree

- Initial node has f(initial_node) = 4
- Apply A* algorithm to solve it.
- The choice of evaluation function critically determines search results.



www.Jntufastupdates.com

**Start state**

| | | |
|---|---|---|
| 3 | 7 | 6 |
| 5 | 1 | 2 |
| 4 | □ | 8 |

**Goal state**

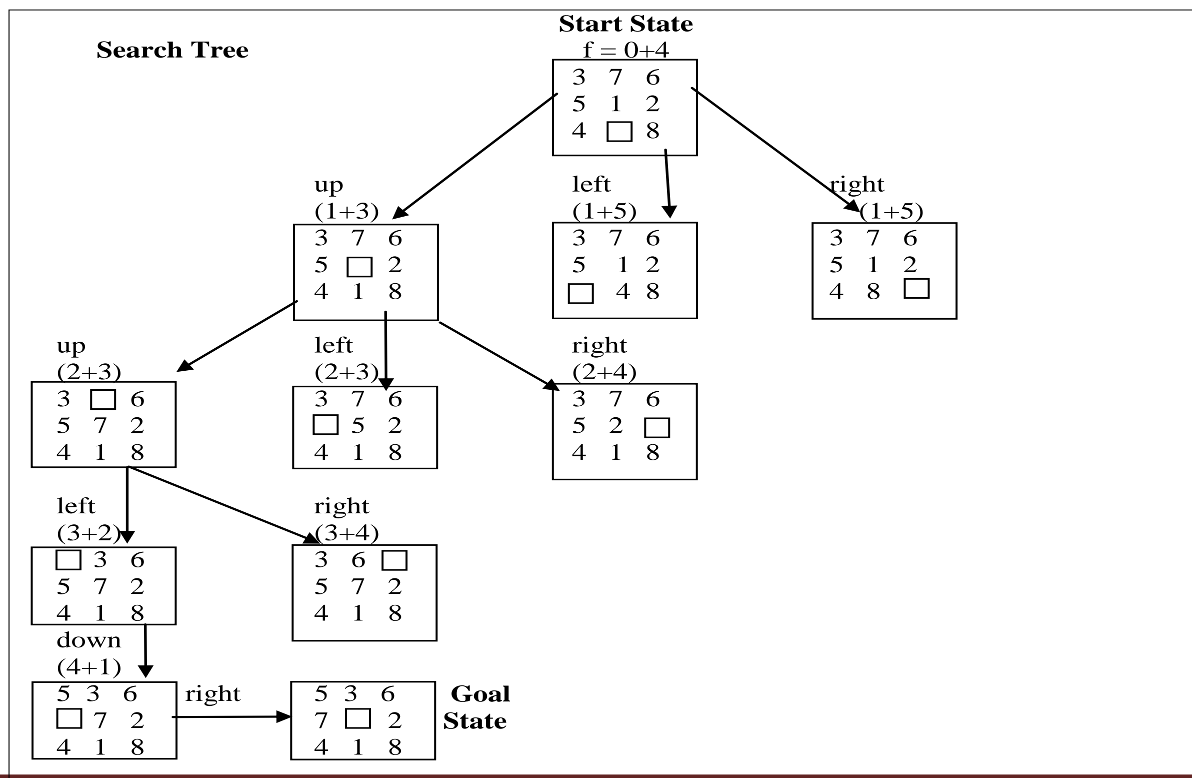
| | | |
|---|---|---|
| 5 | 3 | 6 |
| 7 | □ | 2 |
| 4 | 1 | 8 |

**Evaluation function - f for Eight Puzzle Problem:**

- The choice of evaluation function critically determines search results.
- Consider Evaluation function

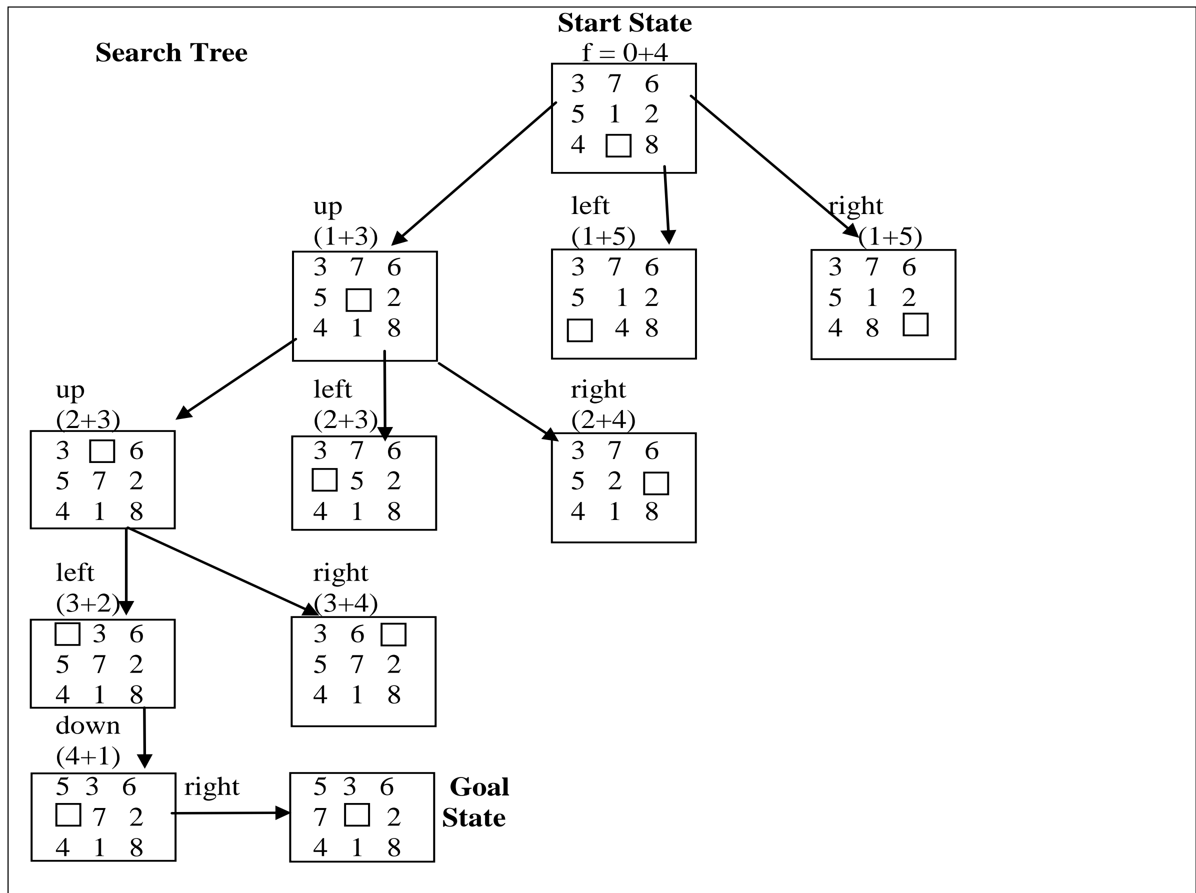
  f (X) = g (X) + h(X)

  - h (X) = the number of tiles not in their goal position in a given state X
  - g(X) = depth of node X in the search tree
- For Initial node
  - f(initial_node) = 4
- Apply A* algorithm to solve it.






www.Jntufastupdates.com




**Harder Problem**

- **Harder problems (8 puzzle) can't be solved by heuristic function defined earlier.**

Initial State
State

Goal

| | | | | | |
|---|---|---|---|---|---|
| 2 | 1 | 6 | 1 | 2 | 3 |
| 4 | | 8 | 8 | | 4 |
| 7 | 5 | 3 | 7 | 6 | 5 |

- A better estimate function is to be thought.

    h (X) = the sum of the distances of the tiles from their goal position in a given state X

- Initial node has h(initial_node) = 1+1+2+2+1+3+0+2=12



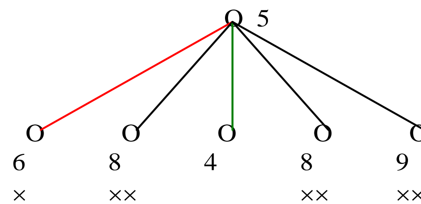
www.Jntufastupdates.com

## 2.5.7.Iterative Deepening A* & Constraint Satisfaction Problems

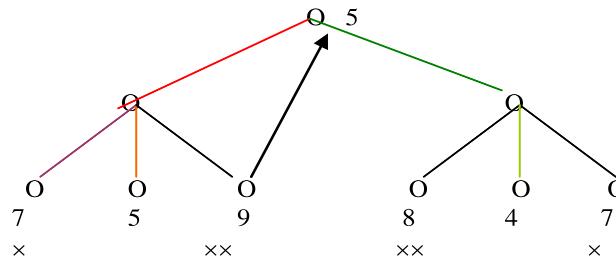
### IDA* Algorithm

- At each iteration, perform a DFS cutting off a branch when its total cost (g+h) exceeds a given threshold.
- This threshold starts at the estimate of the cost of the initial state, and increases for each iteration of the algorithm.
- At each iteration, the threshold used for the next iteration is the minimum cost of all values exceeded the current threshold.

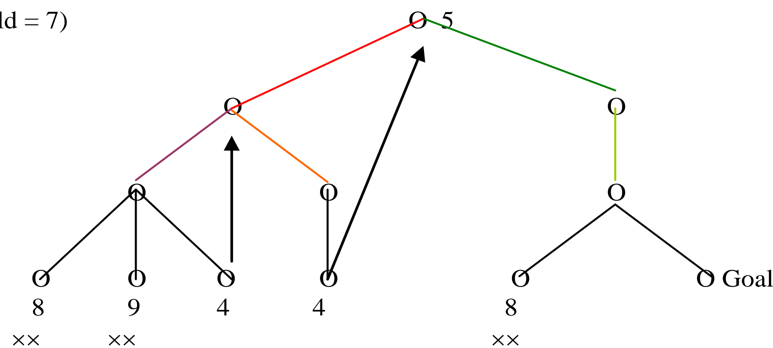
Ist iteration ( Threshhold = 5)




IInd iteration ( Threshhold = 6)




IIIrd iteration ( Threshhold = 7)




- Given an admissible monotone cost function, IDA* will find a solution of least cost or optimal solution if one exists.



www.Jntufastupdates.com

- IDA* not only finds cheapest path to a solution but uses far less space than A* and it expands approximately the same number of nodes as A* in a tree search.
- An additional benefit of IDA* over A* is that it is simpler to implement, as there are no open and closed lists to be maintained.
- A simple recursion performs DFS inside an outer loop to handle iterations.

**2.6. Constrained Satisfaction:**

- Many AI problems can be viewed as problems of constrained satisfaction in which the goal is to solve some problem state that satisfies a given set of constraints.
- Example of such a problem are
  - Crypt-Arithmetic puzzles.
  - Many design tasks can also be viewed as constrained satisfaction problems.
  - N-Queen: Given the condition that no two queens on the same row/column/diagonal attack each other.
  - Map colouring: Given a map, colour three regions in blue, red and black, such that no two neighbouring regions have the same colour.
- Such problems do not require a new search methods.

They can be solved using any of the search strategies which can be augmented with the list of constraints that change as parts of the problem are solved.

**Algorithm:**

- Until a complete solution is found or all paths have lead to dead ends

{

  - Select an unexpanded node of the search graph.
  - Apply the constraint inference rules to the selected node to generate all possible new constraints.



www.Jntufastupdates.com

- If the set of constraints contain a contradiction, then report that this path is a dead end.
- If the set of constraint describes a complete solution, then report success.
- If neither a contradiction nor a complete solution has been found, then
  - apply the problem space rules to generate new partial solutions that are consistent with the current set of constraints.
  - Insert these partial solutions into the search graph.

}

**2.6.1.Crypt-Arithmetic puzzle**

- **Problem Statement:**
  - Solve the following puzzle by assigning numeral (0-9) in such a way that each letter is assigned unique digit which satisfy the following addition.
  - Constraints : No two letters have the same value. (The constraints of arithmetic).

| | S | E | N | D |
|---|---|---|---|---|
| + | M | O | R | E |
| M | O | N | E | Y |

- Initial Problem State
  - S = ? ; E = ? ;N = ? ; D = ? ; M = ? ;O = ? ; R = ? ;Y = ?
- Carries :
- $C_4$ = ? ; $C_3$ = ? ; $C_2$ = ? ; $C_1$ = ?



www.Jntufastupdates.com

| $C_4$ | $C_3$ | $C_2$ | $C_1$ | ← | Carry |
|---|---|---|---|---|---|
| | **S** | **E** | **N** | **D** | |
| + | **M** | **O** | **R** | **E** | |
| **M** | **O** | **N** | **E** | **Y** | |

Constraint equations:

$Y = D + E \longrightarrow C_1$

$E = N + R + C_1 \longrightarrow C_2$

$N = E + O + C_2 \longrightarrow C_3$

$O = S + M + C_3 \longrightarrow C_4$

$M = C_4$

- We can easily see that M has to be non zero digit, so the value of C4 =1

1. $M = C4 \Rightarrow$ **$M = 1$**
2. $O = S + M + C3 \rightarrow C4$

   For C4 =1, $S + M + C3 > 9 \Rightarrow$

   $S + 1 + C3 > 9 \Rightarrow S+C3 > 8.$

   If C3 = 0, then S = 9 else if C3 = 1,

   then S = 8 or 9.

- We see that for S = 9
  - C3 = 0 or 1



www.Jntufastupdates.com

- ❑ It can be easily seen that C3 = 1 is not possible as O = S + M + C3 ⇒ O = 11 ⇒ O has to be assigned digit 1 but 1 is already assigned to M, so not possible.
- ❑ Therefore, only choice for C3 = 0, and thus O = 10. This implies that O is assigned 0 (zero) digit.

- **Therefore, O = 0**

**M = 1, O = 0**

| $C_4$ | $C_3$ | $C_2$ | $C_1$ | ← | Carry |
|---|---|---|---|---|---|
| | S | E | N | D | |
| + | M | O | R | E | |
| M | O | N | E | Y | |

Y = D + E → C1

E = N + R + C1 → C2

N = E + O + C2 → C3

O = S + M + C3 → C4

M = C4

Since C3 = 0; N = E + O + C2 produces no carry.

- As O = 0, N = E + C2 .
- Since N ≠ E, therefore, C2 = 1.

**Hence N = E + 1**

- Now E can take value from 2 to 8 {0,1,9 already assigned so far }
  - ❑ If E = 2, then N = 3.
  - ❑ Since C2 = 1, from E = N + R + C1 , we get 12 = N + R + C1
    - If C1 = 0 then R = 9, which is not possible as we are on the path with S = 9
    - If C1 = 1 then R = 8, then



www.Jntufastupdates.com

- From Y = D + E , we get 10 + Y= D + 2 .
- For no value of D, we can get Y.

- Try similarly for E = 3, 4. We fail in each case.

| $C_4$ | $C_3$ | $C_2$ | $C_1$ | ← | Carry |
|---|---|---|---|---|---|
| | S | E | N | D | |
| + | M | O | R | E | |
| M | O | N | E | Y | |

Y = D + E → C1

E = N + R + C1 → C2

N = E + O + C2 → C3

O = S + M + C3 → C4

M = C4

- If E = 5, then N = 6
  - Since C2 = 1, from E = N + R + C1 , we get 15 = N + R + C1 ,
  - If C1 = 0 then R = 9, which is not possible as we are on the path with S = 9.
  - If C1 = 1 then R = 8, then
    - From Y = D + E , we get 10 + Y= D + 5 i.e., 5 + Y = D.
    - If Y = 2 then D = 7. These values are possible.
      - Hence we get the final solution as given below and on backtracking, we may find more solutions.
- **S = 9 ; E = 5 ; N = 6 ; D = 7 ;**
- **M = 1 ; O = 0 ; R = 8 ;Y = 2**

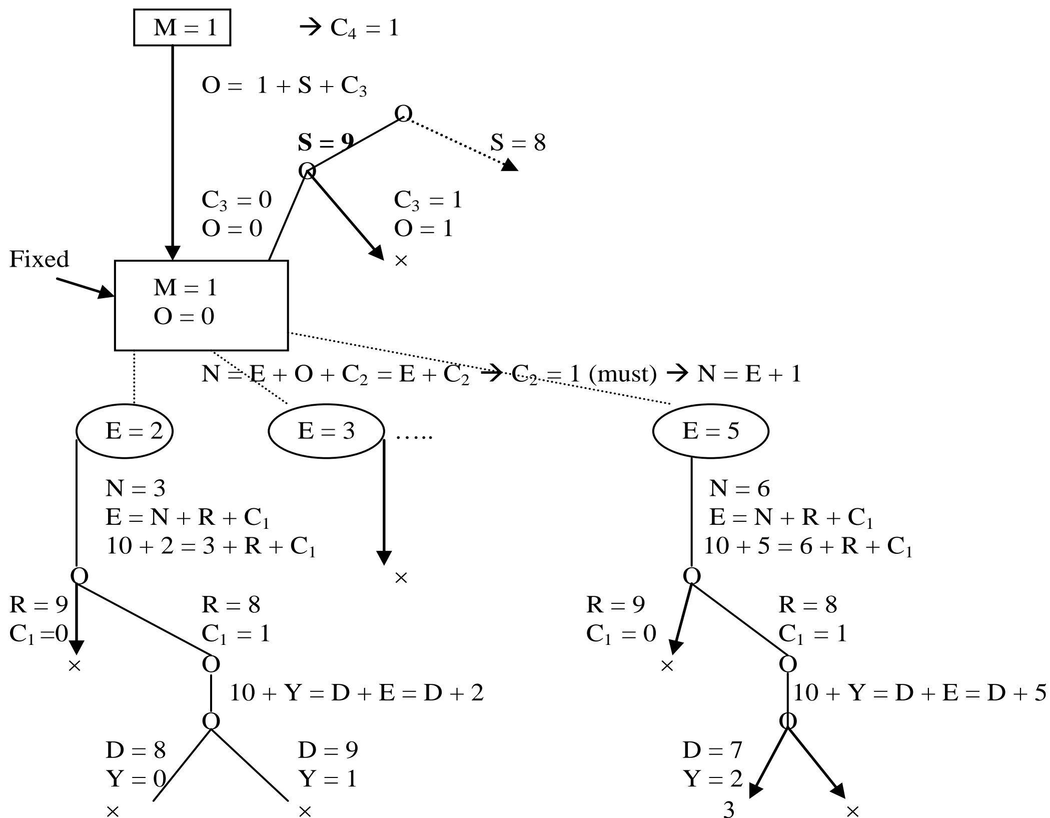


www.Jntufastupdates.com

Constraints:

$Y = D + E \longrightarrow C_1$
$E = N + R + C_1 \longrightarrow C_2$
$N = E + O + C_2 \longrightarrow C_3$
$O = S + M + C_3 \longrightarrow C_4$
$M = C_4$

Initial State




The first solution obtained is:

M = 1, O = 0, S = 9, E = 5, N = 6, R = 8, D = 7, Y = 2

## Solve the following given problem using Constraint Satisfaction

| C4 | C3 | C2 | C1 | | ← Carries |
|---|---|---|---|---|---|
| | B | A | S | E | |
| + | B | A | L | L | |
| G | A | M | E | S | |



www.Jntufastupdates.com

Constraints equations are:

E + L = S → C1

S + L + C1= E → C2

2A + C2 = M → C3

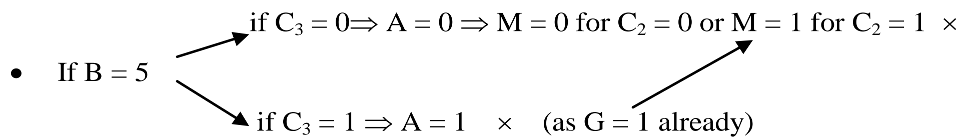
2B + C3 = A → C4

G = C4

Initial Problem State

G = ?; A = ?;M = ?; E = ?; S = ?; B = ?; L = ?

1. $G = C_4 \Rightarrow$ **G = 1**
2. $2B + C_3 = A \quad \rightarrow C_4$
   2.1 Since $C_4 = 1$, therefore, $2B + C_3 > 9 \Rightarrow$ B can take values from 5 to 9.
   2.2 Try the following steps for each value of B from 5 to 9 till we get a possible value of B.




- For B = 6 we get similar contradiction while generating the search tree.
- If **B = 7** , then for $C_3 = 0$, we get **A = 4** $\Rightarrow M = 8$ if $C_2 = 0$ that leads to contradiction, so this path is pruned. If $C_2 = 1$, then **M = 9**.

3. Let us solve $S + L + C_1 = E$ and $E + L = S$
   - Using both equations, we get $2L + C_1 = 0 \Rightarrow$ **L = 5** and $C_1 = 0$
   - Using L = 5, we get $S + 5 = E$ that should generate carry $C_2 = 1$ as shown above
   - So $S+5 > 9 \Rightarrow$ Possible values for E are {2, 3, 6, 8} (with carry bit $C_2 = 1$ )
   - If E = 2 then S + 5 = 12 $\Rightarrow$ S = 7 × (as B = 7 already)
   - If E = 3 then S + 5 = 13 $\Rightarrow$ S = 8.
   - Therefore **E = 3** and **S = 8** are fixed up.
4. Hence we get the final solution as given below and on backtracking, we may find more solutions. In this case we get only one solution.

   **G = 1; A = 4; M = 9; E = 3; S = 8;B = 7; L = 5**



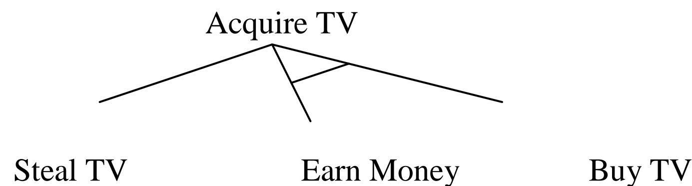
www.Jntufastupdates.com

## 2.7.Problem Reduction & Game Playing

### 2..7.1.Problem Reduction

- So far search strategies discussed were for OR graphs.
  - Here several arcs indicate a different ways of solving problem.
- Another kind of structure is AND-OR graph (tree).
- Useful for representing the solution of problem by decomposing it into smaller sub-problems.
- Each sub-problem is solved and final solution is obtained by combining solutions of each sub-problem.
- Decomposition generates arcs that we will call AND arc.
- One AND arc may point to any number of successors, all of which must be solved.
- Such structure is called AND–OR graph rather than simply AND graph.

**Example of AND-OR Tree**




**AND–OR Graph**

- To find a solution in AND–OR graph, we need an algorithm similar to A*
  - with the ability to handle AND arc appropriately.
- In search for AND-OR graph, we will also use the value of heuristic function f for each node.

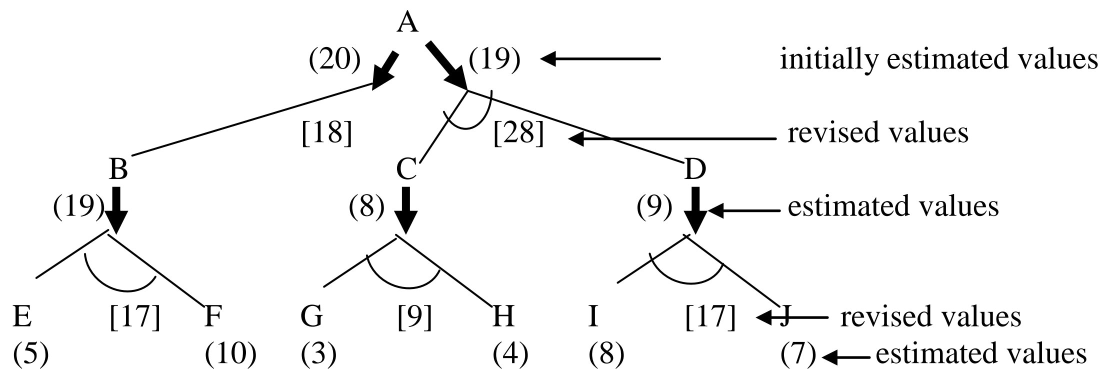
**AND–OR Graph Search**

- Traverse AND-OR graph, starting from the initial node and follow the current best path.



www.Jntufastupdates.com

- Accumulate the set of nodes that are on the best path which have not yet been expanded.
- Pick up one of these unexpanded nodes and expand it.
- Add its successors to the graph and compute f (using only h) for each of them.
- Change the f estimate of newly expanded node to reflect the new information provided by its successors.
    - Propagate this change backward through the graph to the start.
- Mark the best path which could be different from the current best path.
- Propagation of revised cost in AND-OR graph was not there in A*.
- Consider AND-OR graph given on next slide.
    - Let us assume that each arc with single successor will have a cost of 1 and each AND arc with multiple successor will have a cost of 1 for each of its components for the sake of simplicity.
    - Here the numbers listed in the circular brackets ( ) are estimated cost and the revised costs are enclosed in square brackets [ ].
    - Thick lines indicate paths from a given node.




- Initially we start from start node A and compute heuristic values for each of its successors, say {B, (C and D)} as {19, (8, 9)}.
- The estimated cost of paths from A to B is 20 (19 + cost of one arc from A to B) and from A to (C and D) path is 19 ( 8+9 + cost of two arcs A to C and A to D).



www.Jntufastupdates.com

- The path from A to (C and D) seems to be better. So expend this AND path by expending C to {(G and H)} and D to {(I and J)}.
- Now heuristic values of G, H, I and J are 3, 4, 8 and 7 respectively.
- This leads to revised cost of C and D as 9 and 17 respectively.
- These values are propagated up and the revised costs of path from A through (C and D) is calculated as 28 (9 + 17 + cost of arcs A to C and A to D).
- Now the revised cost of this path is 28 instead of earlier estimation of 19 and this path is no longer a best path.
- Then choose path from A to B for expansion.
- After expansion we see that heuristic value of node B is 17 thus making cost of path from A to B to be 18.
- This path is still best path so far, so further explore path from A to B.
- The process continues until either a solution is found or all paths have lead to dead ends, indicating that there is no solution.

**Cyclic Graph**

- If a graph is cyclic (containing cycle) then the algorithm discussed earlier does not operate unless modified as follows:
  - If successor is generated and found to be already in the graph, then
    - we must check that the node in the graph is not an ancestor of the node being expanded.
    - If not, then newly discovered path to the node be entered in the graph.
- We can now state precisely the steps taken for performing heuristic search of an AND-OR graph.
- Algorithm for searching AND-OR graph is called AO*
  - Here we maintain single structure G, representing the part of the search graph explicitly generated so far rather than two lists, OPEN and CLOSED as in previous algorithms.



www.Jntufastupdates.com

- Each node in the graph will
  - point both down to its immediate successors and up to its immediate predecessor.
  - have an h value (an estimate of the cost of a path from current node to a set of solution nodes) associated with it.
  - We will not store **g** (cost from start to current node) as it is not possible to compute a single such value since there may be many paths to the same state.
  - The value **g** is also not necessary because of the top-down traversing of the best-known path which guarantees that only nodes on the best path will ever be considered for expansion.
  - So **h** will be good estimate for AND/OR graph search.

**The "Solve" labeling Procedure**

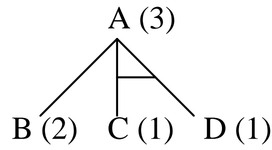
- A terminal node is labeled as
  - "solved" if it is a goal node (representing a solution of sub-problem)
  - "unsolved" otherwise (as we can not further reduce it)
- A non-terminal AND node labeled as
  - "solved" if all of its successors are "solved".
  - "unsolved" as soon as one of its successors is labeled "unsolved".
- A non-terminal OR node is labeled as
  - "solved" as soon as one of its successors is labeled "solved".
  - "unsolved" if all its successors are "unsolved".



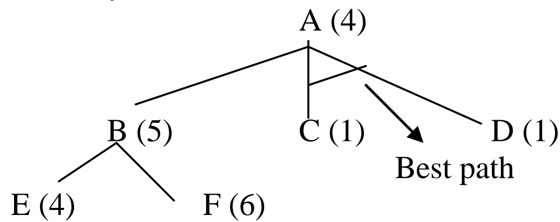
www.Jntufastupdates.com

**Example**

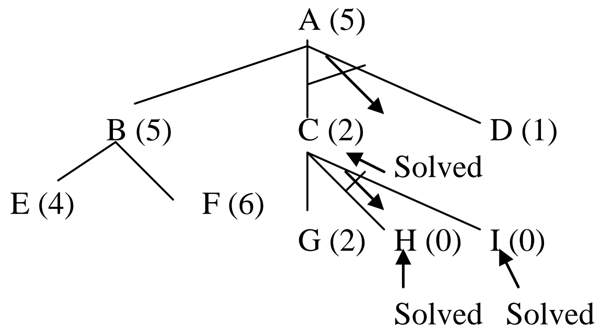
1. After one cycle



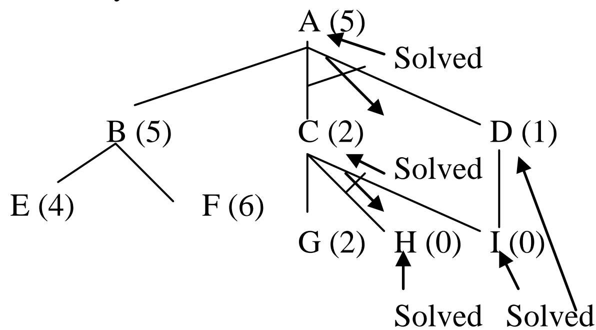

2. After two cycle




3. After three cycle




4. After four cycle






www.Jntufastupdates.com

**2.7.2.AO* Algorithm**

- Let graph G consists initially the start node. Call it INIT.
- Compute h(INIT).
- Until INIT is SOLVED or h(INIT) > Threshold or **Un_Sol**

{1

- Traverse the graph starting from INIT and follow the current best path.
- Accumulate the set of nodes that are on the path which have not yet been expanded or labeled as SOLVED.
- Select one of these unexpanded nodes. Call it NODE and expand it.
- Generate the successors of NODE. If there are none, then assign Threshold as the value of this NODE else for each SUCC that is also not ancestor of NODE do the following

{2

- Add SUCC to the graph G and compute **h** for each.
- If h(SUCC) = 0 then it is a solution node and label it as SOLVED. Propagate the newly discovered information up the graph as follows:
- Initialize S with NODE.
- Until S is empty

{3

- Select from S, a node such that the selected node has no ancestor in G occurring in S /* to avoid cycle */.
- Call it CURRENT and remove it from S.
- Compute the cost of each arcs emerging from CURRENT.
- **Cost of AND arc** = (h of each of the nodes at the end of the arc) + (cost of arc itself)



www.Jntufastupdates.com

- Assign the minimum of the costs as new h value of CURRENT.
- Mark the best path out of CURRENT (with minimum cost).
- Mark CURRENT node as SOLVED if all of the nodes connected to it through the new marked arc have been labeled SOLVED.
- If CURRENT has been marked SOLVED or if the cost of CURRENT was just changed, then new status must be propagated back up the graph. So add to S all of the ancestors of CURRENT.

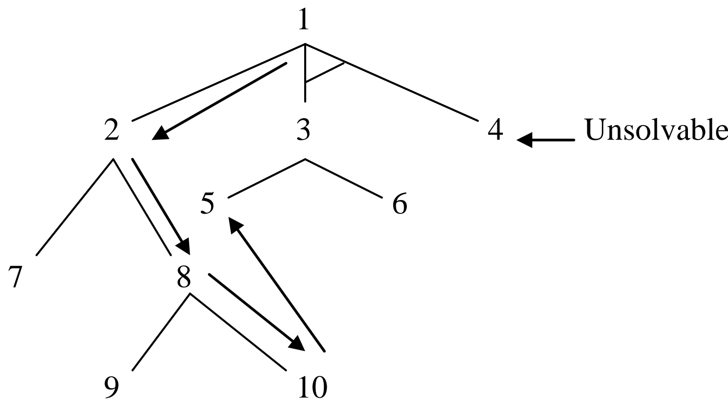
3}

2}

1}

**Longer Path May be Better**

- Consider another example




**Explanation**

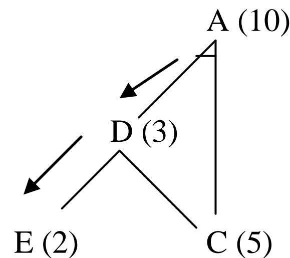
- Nodes are numbered in order of their generation.
- Now node 10 is expanded at the next step and one of its successors is node 5.
- This new path to 5 is longer than the previous path to 5 going through 3.



www.Jntufastupdates.com

- But since the path through 3 will only lead to a solution as there is no solution to 4, so the path through 10 is better.

**AO* may fail to take into account an interaction between sub-goals.**




- Assume that both C and E ultimately lead to a solution.
- According to AO* algorithm, both C and D must be solved to solve A.
- Algorithm considers the solution of D as a completely separate process from the solution of C.
  - As there is no interaction between these two sub-goals).
- Looking just at the alternative from D, the path from node E is the best path but it turns out that C is must anyways, so it is better also to use it to satisfy D.
- But to solve D, the path from node E is the best path and will try to solve E.

AO* algorithm does not consider such interactions, so it will find a non-optimal path.

## 2.8. Game Playing

- Games require different search procedures.
- Basically they are based on generate and test philosophy.
- The generator generates individual move in the search space, each of which is then evaluated by the tester and the most promising one is chosen.



www.Jntufastupdates.com

- Game playing is most practical and direct application of the heuristic search problem solving paradigm.
- It is clear that to improve the effectiveness of a search for problem solving programs, there are two things that can be done:
  - Improve the generate procedure so that only good moves (paths) are generated.
  - Improve the test procedure so that the best moves (paths) will be recognized and explored first.
- Let us consider only two player discrete, perfect information games, such as tic-tac-toe, chess, checkers etc.
  - **Discrete** because they contain finite number of states or configurations.
  - **Perfect-information** because both players have access to the same information about the game in progress (card games are not perfect - information games).
- Two-player games are easier to imagine & think and more common to play.
- Typical characteristic of the games is to 'look ahead' at future positions in order to succeed.
- There is a natural correspondence between such games and state space problems.
- For example,

| **State Space** | | **Game Problem** |
|---|---|---|
| states | - | legal board positions |
| operators | - | legal moves |
| goal | - | winning positions |

- The game begins from a specified initial state and ends in position that can be declared **win** for one, **loss** for other or possibly a **draw**.
- Game tree is an explicit representation of all possible plays of the game.



www.Jntufastupdates.com

  - The root node is an initial position of the game.
  - Its successors are the positions that the first player can reach in one move, and
  - Their successors are the positions resulting from the second player's replies and so on.
  - Terminal or leaf nodes are represented by WIN, LOSS or DRAW.
  - Each path from the root to a terminal node represents a different complete play of the game.

**Correspondence with AND/OR graph**

- The correspondence between **game tree** and **AND/OR** tree is obvious.
  - The moves available to one player from a given position can be represented by OR links.
  - Whereas the moves available to his opponent are AND links.
- The trees representing games contain two types of nodes:
  - MAX- nodes (nodes with OR links, maximizing its gain)
  - MIN - nodes (nodes with AND links, minimizing opponent’s its gain)
- The leaf nodes are leveled WIN, LOSS or DRAW depending on
  - whether they represent a win, loss or draw position from MAX's view point.
- Each non-terminal nodes in the game tree can be labeled WIN, LOSS or DRAW by a bottom up process similar to the "Solve" labeling procedure in AND/OR graph.
- If j is a non-terminal MAX node, then

STATUS (j) =
- WIN , if any of j's successor is a WIN
- LOSS , if all j's successor are LOSS
- DRAW, if any of j's successor is a DRAW and none is WIN

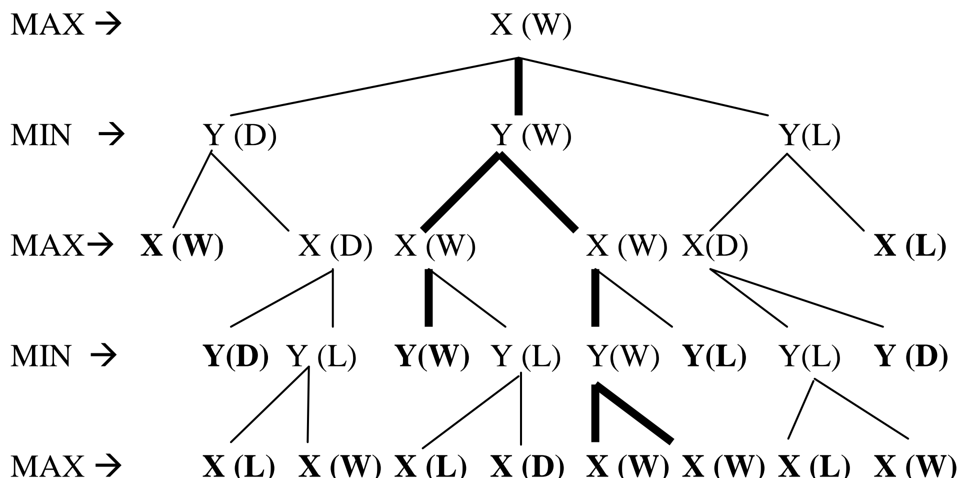


www.Jntufastupdates.com

- If j is a non-terminal MIN node, then

STATUS (j) =
- WIN , if all j's successor is a WIN
- LOSS , if any of j's successor are LOSS
- DRAW, if any of j's successor is a DRAW and none is LOSS

- The function STATUS(j) should be interpreted as the best terminal status MAX can achieve from position j, if MAX plays optimally against a perfect opponent.
- Example: Consider a game tree on next slide.
  - Let us denote
    - MAX → X
    - MIN → Y,
    - WIN → W,
    - DRAW → D and
    - LOSS → L.
  - The status of the leaf nodes is assigned by the rules of the game whereas, those of non-terminal nodes are determined by the labeling procedure.
  - Solving a game tree means labeling the root node by WIN, LOSS, or DRAW from Max player point of view.






www.Jntufastupdates.com

- Labeling is done from MAX point of view.
- Associated with each root label, there is an optimal playing strategy which prescribes how that label can be guaranteed regardless of MIN.
- Highlighted paths are optimal paths for MAX to play.
- An optimal strategy for MAX is a sub-tree whose all nodes are WIN.(See fig on the previous slides)

### 2.9. Look-ahead Strategy

- The status labeling procedure described earlier requires that a complete game tree or at least sizable portion of it be generated.
- For most of the games, tree of possibilities is far too large to be generated and evaluated backward from the terminal nodes in order to determine the optimal first move.

**Examples:**

- Checkers : Non-terminal nodes are $10^{40}$ and $10^{21}$ centuries required if 3 billion nodes could be generated each second.
- Chess : $10^{120}$ nodes and $10^{101}$ centuries.
- So this approach is not practical

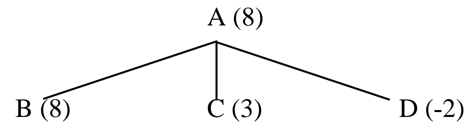
**Evaluation Function**

- Having no practical way of evaluating the exact status of successor game positions, one may naturally use heuristic approximation.
- Experience shows that certain features in a game position contribute to its strength, whereas others tend to weaken it.
- The static evaluation function converts all judgments about board situations into a single, overall quality number.

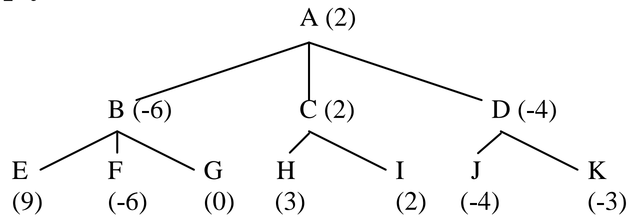


www.Jntufastupdates.com

**One - ply search**




**Two - ply search**






www.Jntufastupdates.com