CHAPTER 1: WHAT IS ARTIFICIAL INTELLIGENCE?

• It is a branch of Computer Science that pursues creating the computers or machines as intelligent as human beings.

OR

• It is the science and engineering of making intelligent machines, especially intelligent computer programs.

OR

• It is related to the similar task of using computers to understand human intelligence, but AI does not have to confine itself to methods that are biologically observable.

Definition

• Artificial Intelligence is the study of how to make computers do things which, at the moment, people do better.

• According to the father of Artificial Intelligence, John McCarthy, it is :

"The science and engineering of making intelligent machines, especially intelligent computer programs".

• From a **business perspective** AI is a set of very powerful tools, and methodologies for using those tools to solve business problems.

• From a **programming perspective**, AI includes the study of symbolic programming, problem solving, and search.

AI Vocabulary

• **Intelligence** relates to tasks involving higher mental processes,

• e.g. creativity, solving problems, pattern recognition, classification, learning, induction, deduction, building analogies, optimization, language processing, knowledge and many more.

• Intelligence is the computational part of the ability to achieve goals.

• **Intelligent behavior** is depicted by perceiving one's environment, acting in complex environments, learning and understanding from experience, reasoning to solve problems and discover hidden knowledge, applying knowledge successfully in new situations, thinking abstractly, using analogies, communicating with others and more.

• **Science based goals of AI** pertain to developing concepts, mechanisms and understanding biological intelligent behavior. The emphasis is on understanding intelligent behavior.

• **Engineering based goals of AI** relate to developing concepts, theory and practice of building intelligent machines. The emphasis is on system building.

- **AI Techniques** depict how we represent, manipulate and reason with knowledge in order to solve problems. Knowledge is a collection of 'facts'. To manipulate these facts by a program, a suitable representation is required. A good representation facilitates problem solving.

- **Learning** means that programs learn from what facts or behavior can represent. Learning denotes changes in the systems that are adaptive in other words, it enables the system to do the same task(s) more efficiently next time.

- **Applications of AI** refers to problem solving, search and control strategies, speech recognition, natural language understanding, computer vision, expert systems, etc.

- Artificial Intelligence is a way of making a computer, a computer-controlled robot, or a software think intelligently, in the similar manner the intelligent humans think.

- AI is accomplished by studying how human brain thinks and how humans learn, decide, and work while trying to solve a problem, and then using the outcomes of this study as a basis of developing intelligent software and systems.

<u>The AI Problem</u>

- **AI** seeks to understand the computations required from intelligent behavior and to produce computer systems that exhibit intelligence.
- Aspects of intelligence studied by AI include perception, communicational using human languages, reasoning, planning, learning and memory.
- The following questions are to be considered before we proceed:

1. What are the underlying assumptions about intelligence?

2. What kinds of techniques will be useful for solving AI problems?
3. At what level human intelligence can be modelled?
4. When will it be realized when an intelligent program has been built?

1.2 Underlying Assumption

Branches of AI

- Logical AI
- Search
- Pattern Recognition
- Representation
- Inference
- Common sense knowledge and Reasoning
- Learning from experience
- Planning
- Epistemology
- Ontology
- Heuristics

Genetic programming
Applications of AI

AI has applications in all fields of human study, such as finance and economics, environmental engineering, chemistry, computer science, and so on. Some of the applications of AI are listed below:

Perception

- Machine vision
- Speech understanding
- Touch (tactile or haptic) sensation

Robotics

Natural Language Processing
Natural Language Understanding
Speech Understanding
Language Generation
Machine Translation

Planning
Expert Systems
Machine Learning
Theorem Proving
Symbolic Mathematics
Game Playing

What is AI Technique?

Artificial Intelligence research during the last three decades has concluded that Intelligence requires knowledge.

- To compensate overwhelming quality, knowledge possesses less desirable properties.
- It is huge.
- It is difficult to characterize correctly.
- It is constantly varying.
- It differs from data by being organized in a way that corresponds to its application.
- It is complicated

An AI technique is a method that exploits knowledge that is represented so that:

- The knowledge captures generalizations that share properties, are grouped together, rather than being allowed separate representation.

- It can be understood by people who must provide it—even though for many programs bulk of the data comes automatically from readings.

- In many AI domains, how the people understand the same people must supply the knowledge to a program.

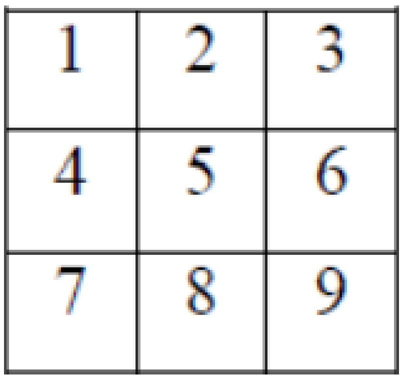
- It can be easily modified to correct errors and reflect changes in real conditions.

- It can be widely used even if it is incomplete or inaccurate.

• It can be used to help overcome its own sheer bulk by helping to narrow the range of possibilities that must be usually considered.

The Level of Model

Example-1: Tic-Tac-Toe



1	2	3
4	5	6
7	8	9

• The Tic-Tac-Toe game consists of a nine element vector called BOARD; it represents the numbers 1 to 9 in three rows.

• An element contains the value 0 for blank, 1 for X and 2 for O.

• A MOVETABLE vector consists of 19,683 elements (3^9) and is needed where each element is a nine element vector. The contents of the vector are especially chosen to help the algorithm.

The algorithm makes moves by pursuing the following:

1. View the vector as a ternary number. Convert it to a decimal number.
2. Use the decimal number as an index in MOVETABLE and access the vector.
3. Set BOARD to this vector indicating how the board looks after the move.

This approach is capable in time but it has several disadvantages.

• It takes more space and requires stunning effort to calculate the decimal numbers.

• This method is specific to this game and cannot be completed.

The second approach

• The structure of the data is as before but we use 2 for a blank, 3 for an X and 5 for an O.

- A variable called TURN indicates 1 for the first move and 9 for the last. The algorithm consists of three actions:
- MAKE2 which returns 5 if the center square is blank; otherwise it returns any blank non corner square, i.e. 2, 4, 6 or 8. POSSWIN (p) returns 0 if player p cannot win on the next move and otherwise returns the number of the square that gives a winning move.
- It checks each line using products 3*3*2 = 18 gives a win for X, 5*5*2=50 gives a win for O, and the winning move is the holder of the blank. GO (n) makes a move to square n setting BOARD[n] to 3 or 5.
- This algorithm is more involved and takes longer but it is more efficient in storage which compensates for its longer time.

It depends on the programmer's skill.

The Final approach

- The structure of the data consists of BOARD which contains a nine element vector, a list of board positions that could result from the next move and a number representing an estimation of how the board position leads to an ultimate win for the player to move.
- This algorithm looks ahead to make a decision on the next move by deciding which the most promising move or the most suitable move at any stage would be and selects the same.
- Consider all possible moves and replies that the program can make. Continue this process for as long as time permits until a winner emerges, and then choose the move that leads to the computer program winning, if possible in the shortest time.
- Actually this is most difficult to program by a good limit but it is as far that the technique can be extended to in any game. This method makes relatively fewer loads on the programmer in terms of the game technique but the overall game strategy must be known to the adviser.

CHAPTER 2

Problems, Problem Spaces, and Search

2.1 Defining the Problem as a State Space Search

- To solve the problem of building a system you should take the following steps:

1. Define the problem accurately including detailed specifications and what constitutes a suitable solution.

2. Scrutinize the problem carefully, for some features may have a central affect on the chosen method of solution.

3. Segregate and represent the background knowledge needed in the solution of the problem.

4. Choose the best solving techniques for the problem to solve a solution.

Problem

- Problem solving is a process of generating solutions from observed data.
- a **'problem'** is characterized by a set of goals,
- a set of objects, and
- a set of operations.

Problem Space

- A '**problem space**' is an abstract space.

A problem space encompasses all valid states that can be generated by the application of any combination of operators on any combination of objects.

The problem space may contain one or more solutions. A solution is a combination of operations and objects that achieve the goals.

Search

- A **'search'** refers to the search for a solution in a problem space.

Search proceeds with different types of 'search control strategies'.

The depth-first search and breadth-first search are the two common search strategies.

✓ General Problem Solver (GPS) was a computer program created in 1957 by Simon and Newell to build a universal problem solver machine.

✓ GPS was based on Simon and Newell's theoretical work on logic machines.

✓ GPS solved many simple problems, such as the Towers of Hanoi, that could be sufficiently formalized, but GPS could not solve any real-world problems.

- A problem is defined by its **‘elements’** and their **‘relations’.**
- To provide a formal description of a problem, we need to do the following:

  a. Define a state space that contains all the possible configurations of the relevant objects, including some impossible ones.

  b. Specify one or more states that describe possible situations, from which the problem-solving process may start. These states are called initial states.

  c. Specify one or more states that would be acceptable solution to the problem. These states are called goal states.

The problem can then be solved by using the rules, in combination with an appropriate control strategy, to move through the problem space until a path from an initial state to a goal state is found.

This process is known as ‘search’.

Thus:

Search is fundamental to the problem-solving process.

Search is a general mechanism that can be used when a more direct method is not known.

Search provides the framework into which more direct methods for solving subparts of a problem can be embedded.

A very large number of AI problems are formulated as search problems.

✓ A problem space is represented by a directed graph, where nodes represent search state and paths represent the operators applied to change the state.

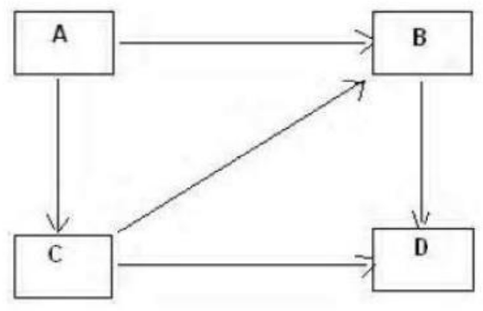
✓ To simplify search algorithms, it is often convenient to logically and programmatically represent a problem space as a tree.

✓ A tree usually decreases the complexity of a search at a cost.

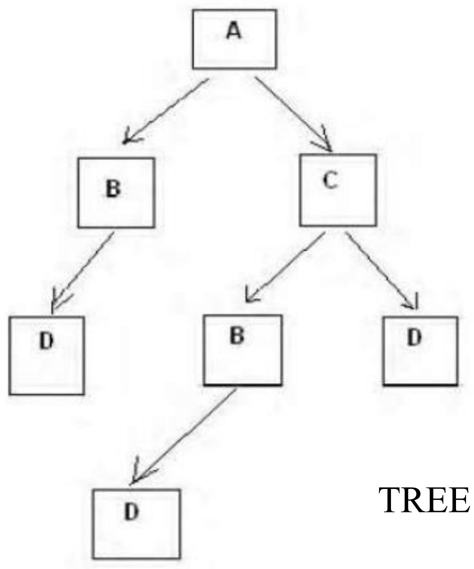
✓ Here, the cost is due to duplicating some nodes on the tree that were linked numerous times in the graph, e.g. node B and node D.

✓ A tree is a graph in which any two vertices are connected by exactly one path.

✓ Alternatively, any connected graph with no cycles is a tree.



GRAPH



TREE

Production Systems

State Space Search

A state space represents a problem in terms of states and operators that change states.

A state space consists of:

- A representation of the states the system can be in.
- For example, in a board game, the board represents the current state of the game.
- A set of operators that can change one state into another state. In a board game, the operators are the legal moves from any given state. Often the operators are represented as programs that change a state representation to represent the new state.
- An initial state.
- A set of final states; some of these may be desirable, others undesirable.

This set is often represented implicitly by a program that detects terminal states.

Water Jug Problem

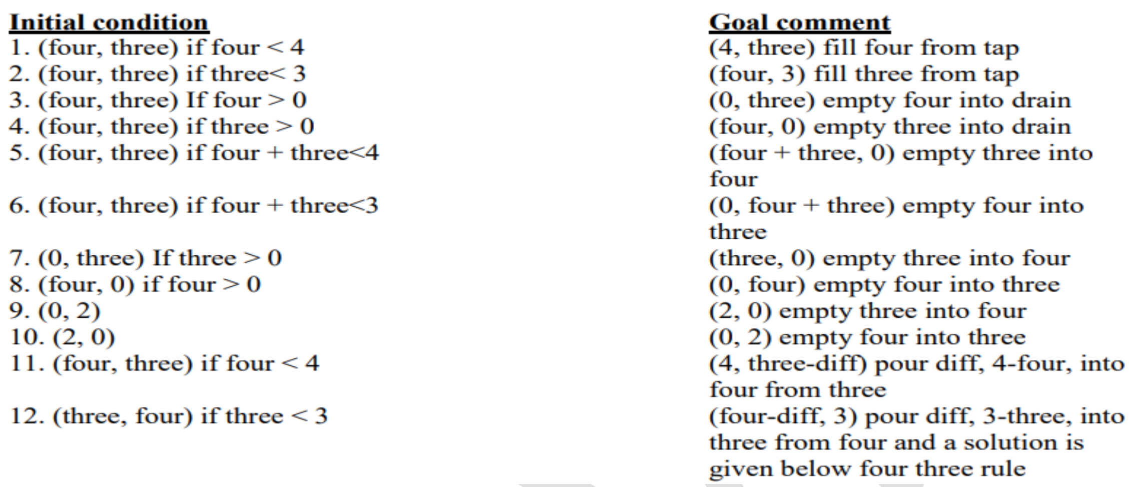
- In this problem, we use two jugs called four and three; four holds a maximum of four gallons of water and three a maximum of three gallons of water.

How can we get two gallons of water in the four jug?

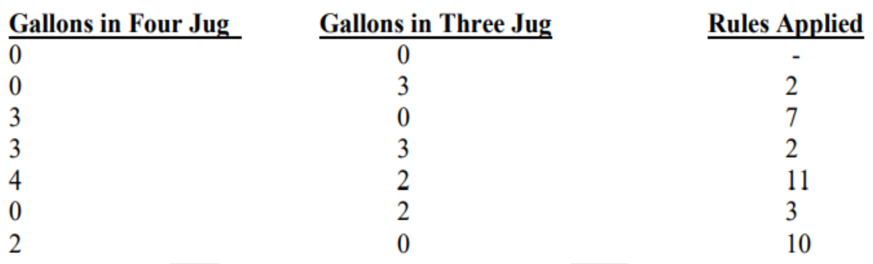
- The state space is a set of prearranged pairs giving the number of gallons of water in the pair of jugs at any time, i.e., (four, three) where four = 0, 1, 2, 3 or 4 and three = 0, 1, 2 or 3.

• The start state is (0, 0) and the goal state is (2, n) where n may be any but it is limited to three holding from 0 to 3 gallons of water or empty.

• Three and four shows the name and numerical number shows the amount of water in jugs for solving the water jug problem.



Initial condition	Goal comment
1. (four, three) if four < 4	(4, three) fill four from tap
2. (four, three) if three< 3	(four, 3) fill three from tap
3. (four, three) If four > 0	(0, three) empty four into drain
4. (four, three) if three > 0	(four, 0) empty three into drain
5. (four, three) if four + three<4	(four + three, 0) empty three into four
6. (four, three) if four + three<3	(0, four + three) empty four into three
7. (0, three) If three > 0	(three, 0) empty three into four
8. (four, 0) if four > 0	(0, four) empty four into three
9. (0, 2)	(2, 0) empty three into four
10. (2, 0)	(0, 2) empty four into three
11. (four, three) if four < 4	(4, three-diff) pour diff, 4-four, into four from three
12. (three, four) if three < 3	(four-diff, 3) pour diff, 3-three, into three from four and a solution is given below four three rule



Gallons in Four Jug	Gallons in Three Jug	Rules Applied
0	0	-
0	3	2
3	0	7
3	3	2
4	2	11
0	2	3
2	0	10

• Production systems provide appropriate structures for performing and describing search processes.

• A production system has four basic components :

• A set of rules each consisting of a left side that determines the applicability of the rule and a right side that describes the operation to be performed if the rule is applied.

• A database of current facts established during the process of inference.

• A control strategy that specifies the order in which the rules will be compared with facts in the database and also specifies how to resolve conflicts in selection of several rules or selection of more facts.

• A rule firing module.

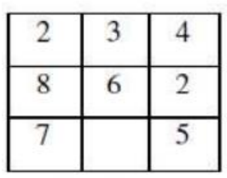
• The production rules operate on the knowledge database.

• Each rule has a precondition—that is, either satisfied or not by the knowledge database. If the precondition is satisfied, the rule can be applied.

- Application of the rule changes the knowledge database.
- The control system chooses which applicable rule should be applied and ceases computation when a termination condition on the knowledge database is satisfied.

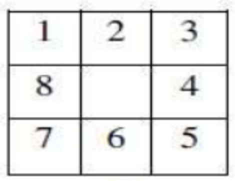
Eight Puzzle Problem

- The 8-puzzle is a 3 × 3 array containing eight square pieces, numbered 1 through 8, and one empty space.
- A piece can be moved horizontally or vertically into the empty space, in effect exchanging the positions of the piece and the empty space.
- There are four possible moves, UP (move the blank space up), DOWN, LEFT and RIGHT.
- The aim of the game is to make a sequence of moves that will convert the board from the start state into the goal state.



2	3	4
8	6	2
7		5

Initial State



1	2	3
8		4
7	6	5

Goal State

<u>Solution:</u>

The puzzle can be solved by moving the tiles one by one in the single empty space and thus achieving the Goal state.

Rules of solving puzzle

Instead of moving the tiles in the empty space we can visualize moving the empty space in place of the tile.

The empty space can only **move in four directions** (Movement of empty space)

1. Up
2. Down
3. Right or
4. Left

The empty space **cannot move diagonally** and can take **only one step at a time**.

Example: Missionaries and Cannibals

The Missionaries and Cannibals problem illustrates the use of state space search for planning under constraints:

Three missionaries and three cannibals wish to cross a river using a two person boat. If at any time the cannibals outnumber the missionaries on either side of the river, they will eat the missionaries.

How can a sequence of boat trips be performed that will get everyone to the other side of the river without any missionaries being eaten?

State representation:

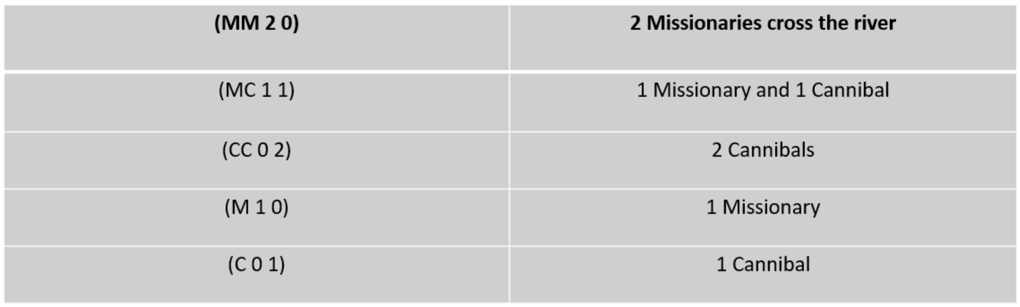
1. BOAT position: original (T) or final (NIL) side

of the river.

2. Number of Missionaries and Cannibals on the original

side of the river.

3. Start is (T 3 3); Goal is (NIL 0 0).



(MM 2 0)	**2 Missionaries cross the river**
(MC 1 1)	1 Missionary and 1 Cannibal
(CC 0 2)	2 Cannibals
(M 1 0)	1 Missionary
(C 0 1)	1 Cannibal

State Representation

- <R,M,C>
- Initial state: <1,3,3> <2,0,0>
- <1,3,1> <2,0,2>
- <1,3,2> <2,0,1>
- <1,3,0> <2,0,3>
- <1,3,1> <2,0,2>
- <1,1,1> <2,2,2>
- <1,2,2> <2,1,1>
- <1,0,2> <2,3,1>
- <1,0,3> <2,3,0>
- <1,0,1> <2,3,2>
- <1,0,2> <2,3,1>

- Goal state: <1,0,0> <2,3,3>

Control strategies

- The word ‘search’ refers to the search for a solution in a problem space.
- Search proceeds with different types of ‘search control strategies’.
- A strategy is defined by picking the order in which the nodes expand.
- The Search strategies are evaluated along the following dimensions:
- Completeness,
- Time complexity,
- Space complexity

Algorithm’s performance and complexity

- Ideally we want a common measure so that we can compare approaches in order to select the most appropriate algorithm for a given situation.
- Performance of an algorithm depends on internal and external factors.
- **Internal factors/ External factors**
- Time required to run
- Size of input to the algorithm
- Space (memory) required to run
- Speed of the computer
- Quality of the compiler

Complexity is a measure of the performance of an algorithm.

Complexity measures the internal factors, usually in time than space.

Computational complexity

It is the measure of resources in terms of **Time and Space**.

- If A is an algorithm that solves a decision problem f, then run-time of A is the number of steps taken on the input of length n.
- Time Complexity T(n) of a decision problem f is the run-time of the ‘best’ algorithm A for f.
- Space Complexity S(n) of a decision problem f is the amount of memory used by the ‘best’ algorithm A for f.

Algorithm BFS: Breadth First Search

1. Create a variable called NODE_LIST and set it to the initial state.
2. Until a goal state is found or NODE_LIST is empty:

a. Remove the first element from NODE_LIST and call it E. If NODE_LIST was empty, quit.

b. For each way that each rule can match the state described in E do:

i. Apply the rule to generate a new state.

ii. If the new state is a goal state, quit and return this state.

iii. Otherwise, add the new state to the end of NODE_LIST.

Algorithm DFS: Depth First Search

1. If the initial state is a goal state, quit and return success.
2. Otherwise, do the following until success or failure is signaled.

a. Generate successor, E of the initial state. If there are no more successors, signal failure.

b. Call Depth-First Search with E as the initial state.

c. If success is returned, signal success. Otherwise continue in this loop.

Advantages of BFS:

3. Used to find the shortest path between states.
4. Always finds optimal solutions.
5. There is nothing like useless path in BFS,since it searches level by level.
6. Finds the closest goal state in less time.

Disadvantages of BFS:

All of the connected vertices must be stored in memory. So consumes more memory

Advantages of DFS:

1. Consumes less memory
2. Finds the larger distant element(from initial state) in less time.

Disadvantages of DFS:

1. May not find optimal solution to the problem.

2. May get trapped in searching useless path.

Heuristic Search

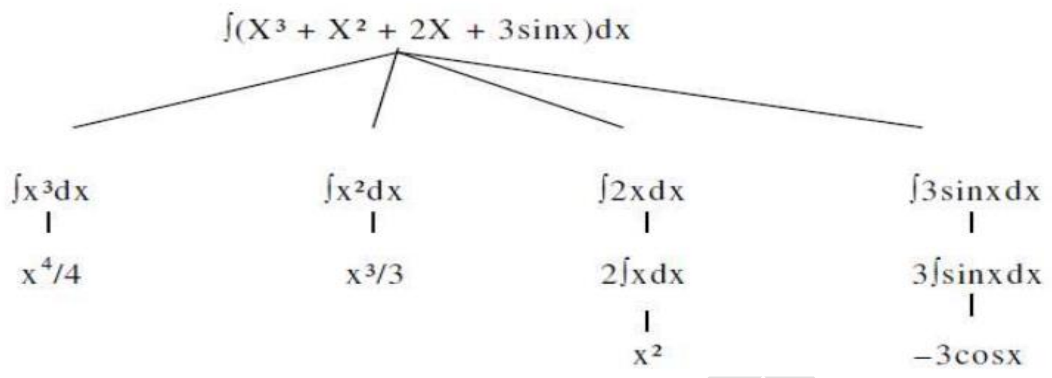
- Heuristic is a technique that improves the efficiency of a search process.
- Eg:Nearest neighbour heuristic in travelling salesman problem.
- A heuristic function is a function that maps from problem state descriptions to measures of desirability.
- Which aspects of problem state are considered,
- How those aspects are evaluated , and
- the weights given to individual aspects are chosen in such a way that the value of the heuristic function at a given node in the search process gives as good an estimate as possible of whether that node is on the desired path to the solution.

Problem Characteristics

- Heuristics cannot be generalized, as they are domain specific.
- Production systems provide ideal techniques for representing such heuristics in the form of IF-THEN rules.
- Most problems requiring simulation of intelligence use heuristic search extensively.
- Some heuristics are used to define the control structure that guides the search process.
- But heuristics can also be encoded in the rules to represent the domain knowledge.
- Since most AI problems make use of knowledge and guided search through the knowledge, AI can be described as the study of techniques for solving exponentially hard problems in polynomial time by exploiting knowledge about problem domain.
- To use the heuristic search for problem solving, we suggest analysis of the problem for the following considerations:

1. Decomposability of the problem into a set of independent smaller subproblems.
2. Possibility of undoing solution steps, if they are found to be unwise.
3. Predictability of the problem universe.
4. Possibility of obtaining an obvious solution to a problem without comparison of all other possible solutions.
5. Type of the solution: whether it is a state or a path to the goal state.
6. Role of knowledge in problem solving.
7. Nature of solution process: with or without interacting with the user.

1: Problem Decomposition:
Suppose to solve the expression is: $\int(X^3 + X^2 + 2X + 3\sin x)\, dx$



1. Monotonic Production System
2. Non monotonic Production System
3. Partially commutative production system
4. Commutative production system:both monotonic and partially commutative.

The Four categories

Production System	**Monotonic**	**Non monotonic**
Partially Commutative	Theorem Proving	Robot Navigation
Not Partially Commutative	Chemical Synthesis	Bridge

Issues in Design of Search Programs

1. The direction in which to conduct the search.
2. How to select applicable rules.
3. How to represent each node of the search process.

Chapter 3

Heuristic Search Techniques

Generate and Test: Generate and Test Strategy

Generate-And-Test Algorithm

• Generate-and-test search algorithm is a very simple algorithm that guarantees to find a solution if done systematically and there exists a solution.

Algorithm: Generate-And-Test

1.Generate a possible solution.

2.Test to see if this is the expected solution.

3.If the solution has been found quit else go to step 1.

Potential solutions that need to be generated vary depending on the kinds of problems. For some problems the possible solutions may be particular points in the problem space and for some problems, paths from the start state.

• Generate-and-test, like depth-first search, requires that complete solutions be generated for testing.

• In its most systematic form, it is only an exhaustive search of the problem space.

• Solutions can also be generated randomly but solution is not guaranteed.

• This approach is what is known as British Museum algorithm: finding an object in the British Museum by wandering randomly.

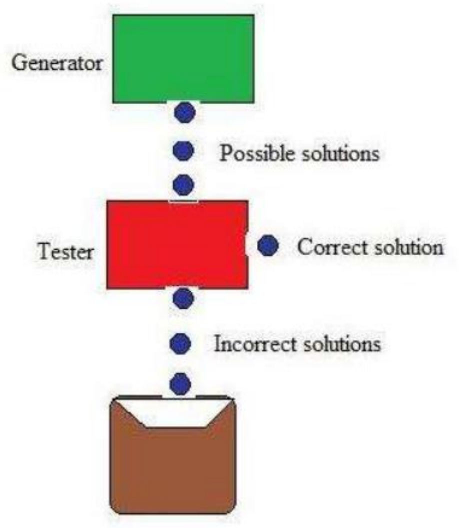


Figure: Generate and Test

Systematic Generate-And-Test

- While generating complete solutions and generating random solutions are the two extremes there exists another approach that lies in between.
- The approach is that the search process proceeds systematically but some paths that unlikely to lead the solution are not considered.
- This evaluation is performed by a heuristic function. Depth-first search tree with backtracking can be used to implement systematic generate-and-test procedure.
- As per this procedure, if some intermediate states are likely to appear often in the tree, it would be better to modify that procedure to traverse a graph rather than a tree.
- Exhaustive generate-and-test is very useful for simple problems.
- But for complex problems even heuristic generate-and-test is not very effective technique.
- But this may be made effective by combining with other techniques in such a way that the space in which to search is restricted.
- An AI program DENDRAL, for example, uses plan-Generate-and-test technique. First, the planning process uses constraint-satisfaction techniques and creates lists of recommended and contraindicated substructures.
- Then the generate-and-test procedure uses the lists generated and required to explore only a limited set of structures.
- Constrained in this way, generate-and-test proved highly effective.
- A major weakness of planning is that it often produces inaccurate solutions as there is no feedback from the world.
- But if it is used to produce only pieces of solutions then lack of detailed accuracy becomes unimportant

Hill Climbing

- Hill Climbing is 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.
- In the above definition, mathematical optimization problems implies that hill climbing solves the problems where we need to maximize or minimize a given real function by choosing values from the given inputs.

- Example-Travelling salesman problem where we need to minimize the distance traveled by salesman.
- 'Heuristic search' means that this search algorithm may not find the optimal solution to the problem.
- However, it will give a good solution in reasonable time.
- A heuristic function is a function that will rank all the possible alternatives at any branching step in search algorithm based on the available information.
- It helps the algorithm to select the best route out of possible routes.

Features of Hill Climbing

1. **Variant of generate and test algorithm** : It is a variant of generate and test algorithm.

- The generate and test algorithm is as follows :

1. Generate a possible solutions.
2. Test to see if this is the expected solution.
3. If the solution has been found quit else go to step 1

- Hence we call Hill climbing as a variant of generate and test algorithm as it takes the feedback from test procedure.
- Then this feedback is utilized by the generator in deciding the next move in search space.

2. Uses the Greedy approach : At any point in state space, the search moves in that direction only which optimizes the cost of function with the hope of finding the optimal solution at the end.

3. No Backtracking: A hill-climbing algorithm only works on the current state and succeeding states (future). It does not look at the previous states.

4. Feedback mechanism: The algorithm has a feedback mechanism that helps it decide on the direction of movement (whether up or down the hill).

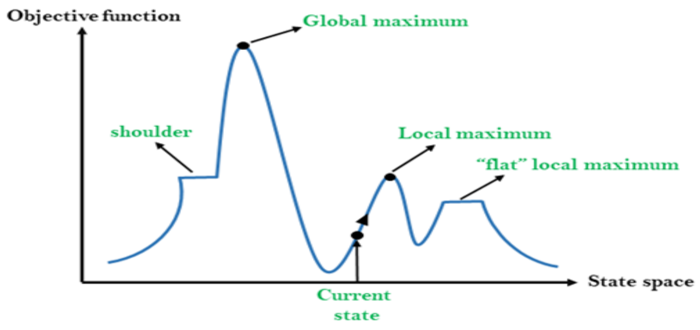
The feedback mechanism is enhanced through the generate-and-test technique.

5. Incremental change: The algorithm improves the current solution by incremental changes.

State Space diagram for Hill Climbing

- State space diagram is a graphical representation of the set of states our search algorithm can reach VS(versus) the value of our objective function(the function which we wish to maximize).
- X-axis : denotes the state space i.e., states or configuration our algorithm may reach.
- Y-axis : denotes the values of objective function corresponding to a particular state.

• The best solution will be that state space where objective function has maximum value(global maximum).



Different regions in the State Space Diagram

1. **Local maximum**

• It is a state which is better than its neighboring state however there exists a state which is better than it(global maximum).

• This state is better because here value of objective function is higher than its neighbors.

2. Global maximum

• It is the best possible state in the state space diagram. This because at this state, objective function has highest value.

3. Plateau/flat local maximum

• It is a flat region of state space where neighboring states have the same value.

4. Ridge :

• It is region which is higher than its neighbors but itself has a slope. It is a special kind of local maximum.

5. Current state

• The region of state space diagram where we are currently present during the search.

6. Shoulder

• It is a plateau that has an uphill edge.

• It examines the neighboring nodes one by one and selects the first neighboring node which optimizes the current cost as next node.

Algorithm for Simple Hill climbing :

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

b) Perform these to evaluate new state

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

ii. If 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 until a solution is found.

Step 3 : Exit.

PROBLEMS OF HILL CLIMBING

• A major problem of hill climbing strategies is their tendency to become stuck at foothills, a plateau or a ridge.

• If the algorithm reaches any of the above mentioned states, then the algorithm fails to find a solution.

• Foothills or **local maxima** is a state that is better than all its neighbors but is not better than some other states farther away.

• At a local maximum, all moves appear to make things worse.

• Foothills are potential traps for the algorithm.

It can be overcome by:

• Utilize backtracking technique.

• Maintain a list of visited states.

• If the search reaches an undesirable state, it can backtrack to the previous configuration and explore a new path.

• A **plateau** is a flat area of the search space in which a whole set of neighboring states have the same value.

• On a plateau, it is not possible to determine the best direction in which to move by making local comparisons.

How to Overcome Plateaus

• Make a big jump.

- Randomly select a state far away from current state.
- Chances are that we will land at a non-plateau region.
- A **ridge** is a special kind of local maximum.
- It is an area of the search space that is higher that the surrounding areas and that itself has a slope.
- But the orientation of the high region, compared to the set of available moves and the directions in which they move, makes it impossible to traverse a ridge by single moves.
- Any point on a ridge can look like peak because movement in all probe directions is downward.

It can be overcome by:

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

Types of Hill Climbing :2. Steepest-Ascent Hill climbing

- It first examines all the neighboring nodes and then selects the node closest to the solution state as next node.
- **Step 1** : Evaluate the initial state. If it is goal state then exit else make the current state as initial state
- **Step 2** : Repeat these steps until a solution is found or current state does not change

```
    i. Let 'target' be a state such that any successor of the current state will be better than it;
    ii. for each operator that applies to the current state
  a. apply the new operator and create a new state
  b. evaluate the new state
  c. if this state is goal state then quit else compare with 'target'
  d. if this state is better than 'target', set this state as 'target'
  e. if target is better than current state set current state to Target
```

Step 3 : Exit

Stochastic Hill Climbing Algorithm:

- It does not examine all the neighboring nodes before deciding which node to select .
- It just selects a neighboring node at random, and decides (based on the amount of improvement in that neighbor) whether to move to that neighbor or to examine another.

Best First Search (Informed Search)

In BFS and DFS, when we are at a node, we can consider any of the adjacent as next node. So both BFS and DFS blindly explore paths without considering any cost function. The idea of Best First Search is to use an evaluation function to decide which adjacent is most promising and then explore. Best First Search falls under the category of Heuristic Search or Informed Search. We use a priority queue to store costs of nodes. So the implementation is a variation of BFS, we just need to change Queue to PriorityQueue.

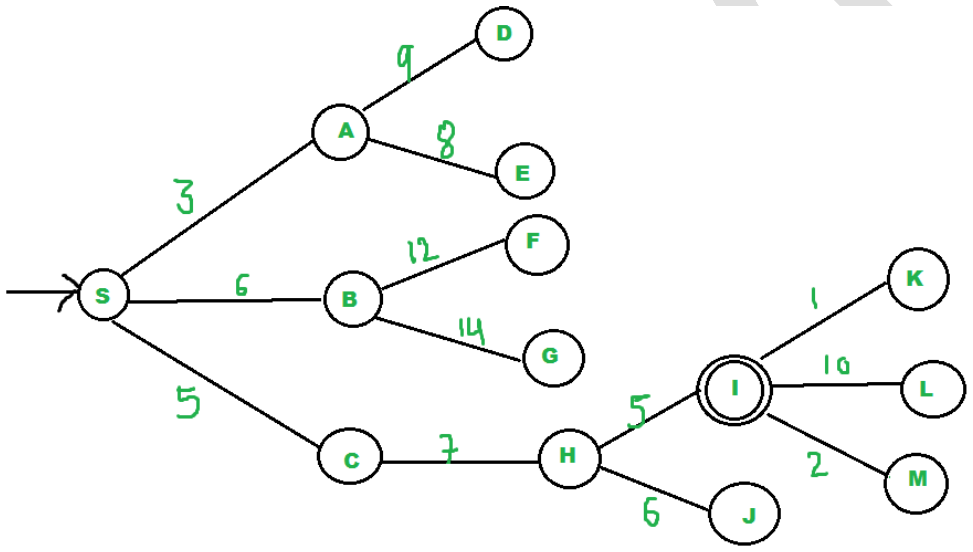
Algorithm: Best-First-Search(Grah g, Node start)

1) Create an empty PriorityQueue PriorityQueue pq;

2) Insert "start" in pq. pq.insert(start)

3) Until PriorityQueue is empty u = PriorityQueue.DeleteMin

If u is the goal Exit Else Foreach neighbor v of u If v "Unvisited" Mark v "Visited" pq.insert(v) Mark v "Examined" End procedure Let us consider below example.



We start from source "S" and search for goal "I" using given costs and Best First search. pq initially contains S We remove s from and process unvisited neighbors of S to pq. pq now contains {A, C, B} (C is put before B because C has lesser cost) We remove A from pq and process unvisited neighbors of A to pq. pq now contains {C, B, E, D} 43 We remove C from pq and process unvisited neighbors of C to pq. pq now contains {B, H, E, D} We remove B from pq and process unvisited neighbors of B to pq. pq now contains {H, E, D, F, G} We remove H from pq. Since our goal "I" is a neighbor of H, we return. Analysis : ♣ The worst case time complexity for Best First Search is O(n * Log n) where n is number of nodes. In worst case, we may have to visit all nodes before we reach goal. Note that priority queue is implemented using Min(or Max) Heap, and insert and remove operations take O(log n) time. ♣ Performance of the algorithm depends on how well the cost or evaluation function is designed.

<u>A* Search Algorithm</u>

A* is a type of search algorithm. Some problems can be solved by representing the world in the initial state, and then for each action we can perform on the world we generate states for what the world would be like if we did so. If you do this until the world is in the state that we specified as a solution, then the route from the start to this goal state is the solution to your problem. Let's look at some of the terms used in Artificial Intelligence when describing this state space search. Some terminology

A node is a state that the problem's world can be in. In pathfinding a node would be just a 2d coordinate of where we are at the present time. In the 8-puzzle it is the positions of all the tiles. Next all the nodes are arranged in a graph where links between nodes represent valid steps in solving the problem. These links are known as edges. State space search, then, is solving a problem by beginning with the start state, and then for each node we expand all the nodes beneath it in the graph by applying all the possible moves that can be made at each point.

Heuristics and Algorithms

At this point we introduce an important concept, the heuristic. This is like an algorithm, but with a key difference. An algorithm is a set of steps which you can follow to solve a problem, which always works for valid input.

For example you could probably write an algorithm yourself for 44 multiplying two numbers together on paper. A heuristic is not guaranteed to work but is useful in that it may solve a problem for which there is no algorithm. We need a heuristic to help us cut down on this huge search problem. What we need is to use our heuristic at each node to make an estimate of how far we are from the goal. In pathfinding we know exactly how far we are, because we know how far we can move each step, and we can calculate the exact distance to the goal. But the 8-puzzle is more difficult. There is no known algorithm for calculating from a given position how many moves it will take to get to the goal state. So various heuristics have been devised. The best one is known as the Nilsson score which leads fairly directly to the goal most of the time, as we shall see.

Cost

When looking at each node in the graph, we now have an idea of a heuristic, which can estimate how close the state is to the goal. Another important consideration is the cost of getting to where we are. In the case of pathfinding we often assign a movement cost to each square. The cost is the same then the cost of each square is one. If we wanted to differentiate between terrain types we may give higher costs to grass and mud than to newly made road. When looking at a node we want to add up the cost of what it took to get here, and this is simply the sum of the cost of this node and all those that are above it in the graph.

AO* Search: (And-Or) Graph

The Depth first search and Breadth first search given earlier for OR trees or graphs can be easilyadopted by AND-OR graph. The main difference lies in the way termination conditions are determined, since all goals following an AND nodes must be realized; where as a single goal node following an OR node will do. So for this purpose we are using AO* algorithm.

Like A* algorithm here we will use two arrays and one heuristic function.

OPEN:

It contains the nodes that has been traversed but yet not been marked solvable or unsolvable.

CLOSE:

It contains the nodes that have already been processed.

Algorithm:

Step 1: Place the starting node into OPEN.

Step 2: Compute the most promising solution tree say T0.

Step 3: Select a node n that is both on OPEN and a member of T0. Remove it from OPEN andplace it in CLOSE.

Step 4: If n is the terminal goal node then leveled n as solved and leveled all the ancestors of nas solved. If the starting node is marked as solved then success and exit.

Step 5: If n is not a solvable node, then mark n as unsolvable. If starting node is marked asunsolvable, then return failure and exit.

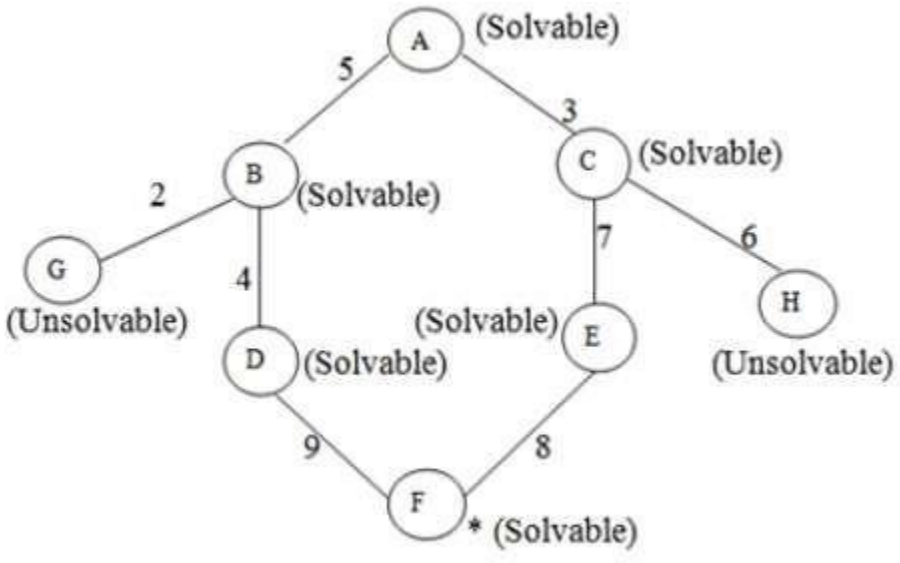
Step 6: Expand n. Find all its successors and find their h (n) value, push them into OPEN.

Step 7: Return to Step 2.

Step 8: Exit

Implementation:

Let us take the following example to implement the AO* algorithm.



Figure

Step 1:

In the above graph, the solvable nodes are A, B, C, D, E, F and the unsolvable nodes are G, H.Take A as the starting node. So place A into OPEN.

i.e. OPEN = $\boxed{A}$ CLOSE = (NULL) $\boxed{\phi}$

Step 2:

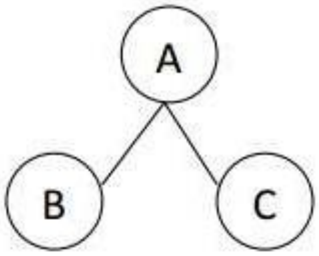
The children of A are B and C which are solvable. So place them into OPEN and place A into the CLOSE.

i.e. OPEN =

B	C

CLOSE =

A



Step 3:

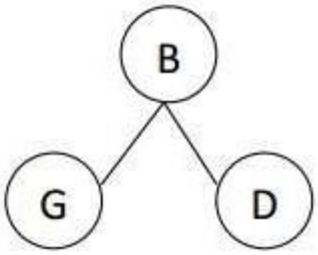
Now process the nodes B and C. The children of B and C are to be placed into OPEN. Also remove B and C from OPEN and place them into CLOSE.

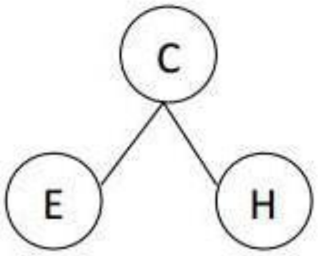
So OPEN =

G	D	E	

C

A	B	C





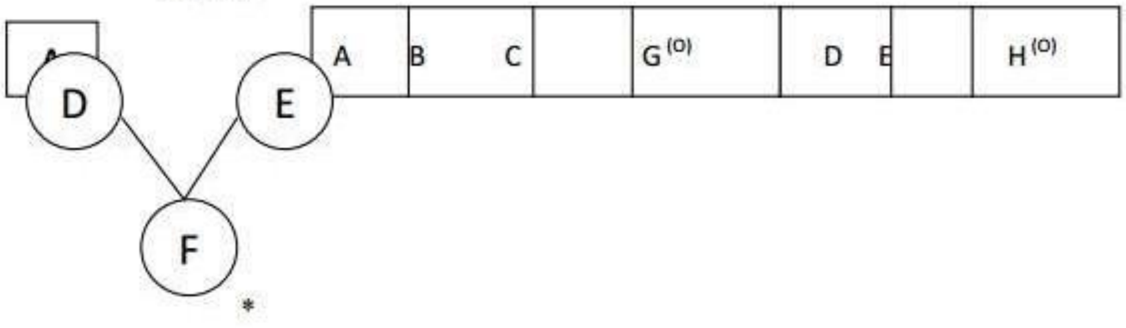
(O)

‘O’ indicated that the nodes G and H are unsolvable.

Step 4:

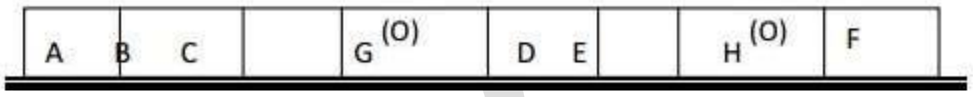
As the nodes G and H are unsolvable, so place them into CLOSE directly and process the nodes D and E.

i.e. OPEN = CLOSE =



Step 5:

Now we have been reached at our goal state. So place F into CLOSE.

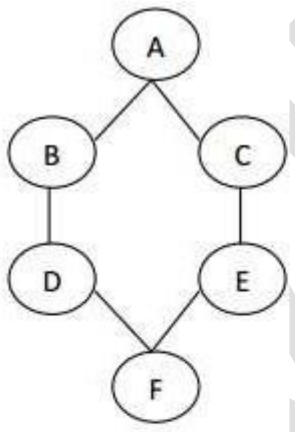


i.e. CLOSE =

Step 6:

Success and Exit

AO* Graph:



Figure

Advantages:

It is an optimal algorithm.

If traverse according to the ordering of nodes. It can be used for both OR and AND graph.

Disadvantages:

Sometimes for unsolvable nodes, it can't find the optimal path. Its complexity is than other algorithms.

PROBLEM REDUCTION

Problem Reduction with AO* Algorithm

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 are may point to any number of successor nodes. All these must be solved so that the

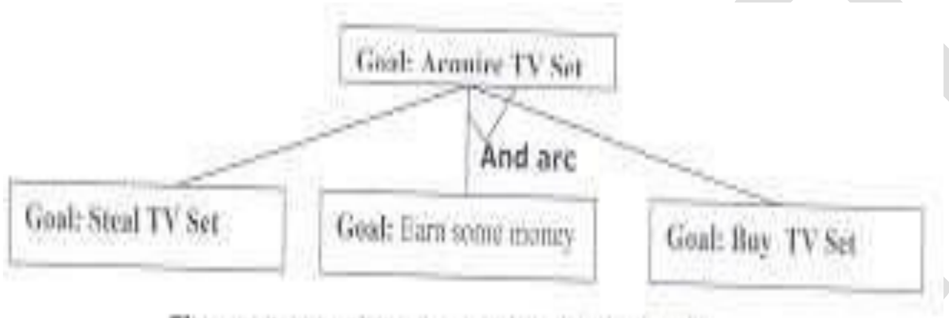


Figure shows AND - Or graph - an example.

arc will rise to many arcs, indicating several possible solutions.Hence the graph is known as AND - OR instead of AND. Figure shows an AND - OR graph.

An algorithm to find a solution in an AND - OR graph must handle AND area appropriately. A* algorithm cannot search AND - OR graphs efficiently. This can be understand from the give figure.

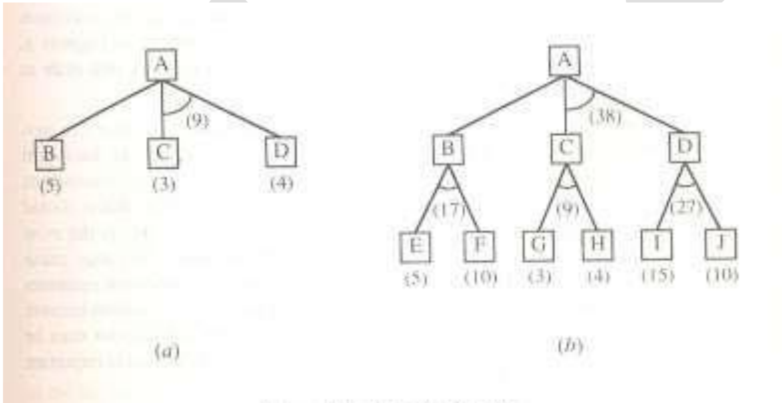


Figure 3.7: AND-OR Graphs

FIGURE : AND - OR graph

In figure (a) the top node A has been expanded producing two area one leading to B and leadingto C-D . the numbers at each node represent the value of f ' at that node (cost of getting to the goal state from current state). For simplicity, it is assumed that every operation(i.e. applying a rule) has unit cost, i.e., each are with single successor will have a cost of 1 and each of its components. With the available information till now , it appears that C is the most promising node to expand since its f ' = 3 , the lowest but going through B would be better since to use C we must also use D' and the cost would be 9(3+4+1+1). Through B it would be 6(5+1).

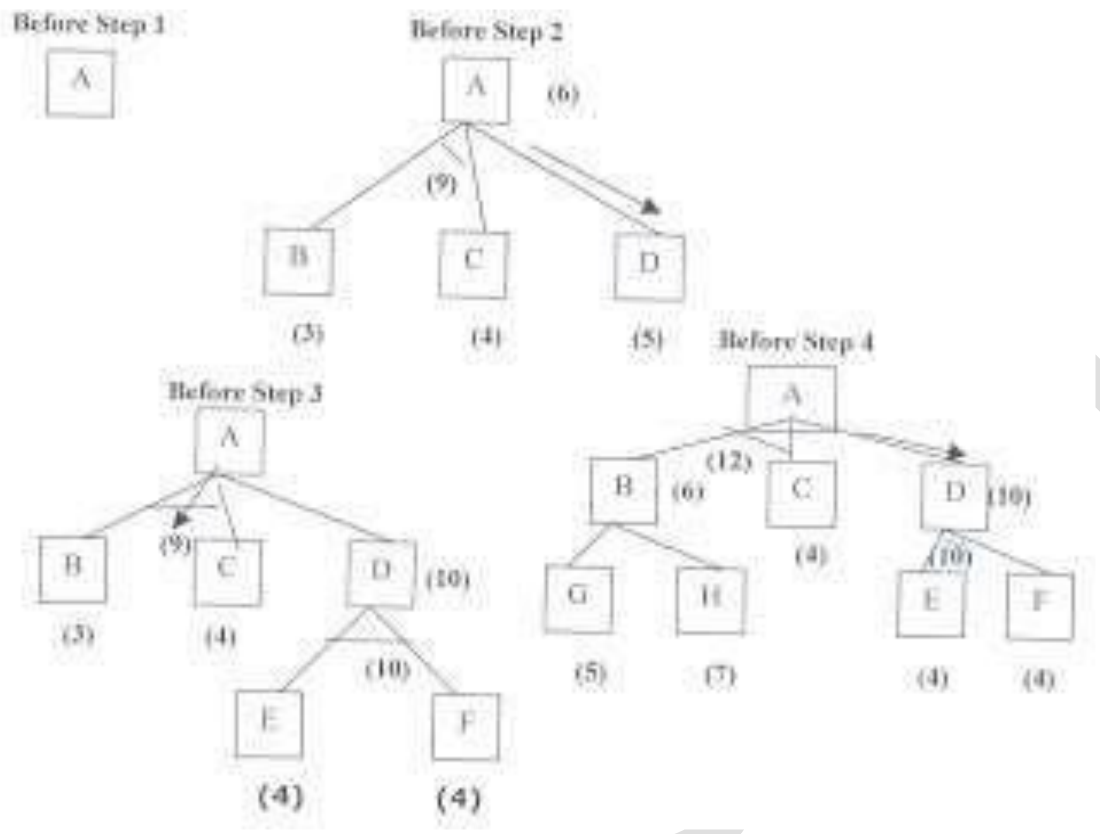
Thus the choice of the next node to expand depends not only n a value but also on whether that node is part of the current best path form the initial mode. Figure (b) makes this clearer. In figurethe node G appears to be the most promising node, with the least f ' value. But G is not on the current beat path, since to use G we must use GH with a cost of 9 and again this demands that arcs be used (with a cost of 27). The path from A through B, E-F is better with a total cost of (17+1=18). Thus we can see that to search an AND-OR graph, the following three things must be done.

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

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

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

The propagation of revised cost estimation backward is in the tree is not necessary in A* algorithm. This is because in AO* algorithm expanded nodes are re-examined so that the currentbest path can be selected. The working of AO* algorithm is illustrated in figure as follows:



Referring the figure. The initial node is expanded and D is Marked initially as promising node. Dis expanded producing an AND arc E-F. f ' value of D is updated to 10. Going backwards we cansee that the AND arc B-C is better . it is now marked as current best path. B and C have to be expanded next. This process continues until a solution is found or all paths have led to dead ends,indicating that there is no solution. An A* algorithm the path from one node to the other is always that of the lowest cost and it is independent of the paths through other nodes.

The algorithm for performing a heuristic search of an AND - OR graph is given below. 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. In AO*algorithm serves as the estimate of goodness of a node. Also a there should value called FUTILITY is used. The estimated cost of a solution is greater than FUTILITY then the search isabandoned as too expansive to be practical.
For representing above graphs AO* algorithm is as follows

AO* ALGORITHM:

1. Let G consists only to the node representing the initial state call this node INTT. Computeh' (INIT).

2. Until INIT is labeled SOLVED or hi (INIT) becomes greater than FUTILITY, repeat thefollowing procedure.

(I) Trace the marked arcs from INIT and select an unbounded node NODE.

(II) Generate the successors of NODE . if there are no successors then assign FUTILITY ash' (NODE). This means that NODE is not solvable. If there are successors then for each
on
e called SUCCESSOR, that is not also an ancester of NODE do the following

(a) add SUCCESSOR to graph G

(b) if successor is not a terminal node, mark it solved and assign zero to its h ' value.

(c) If successor is not a terminal node, compute it h' value.

(III) propagate the newly discovered information up the graph by doing the following . let S be aset of nodes that have been marked SOLVED. Initialize S to NODE. Until S is empty repeat the following procedure;

(a) select a node from S call if CURRENT and remove it from S.

(b) compute h' of each of the arcs emerging from CURRENT , Assign minimum h' toCURRENT.

(c) Mark the minimum cost path a s the best out of CURRENT.

(d) Mark CURRENT SOLVED if all of the nodes connected to it through the new markedare have been labeled SOLVED.

(e) If CURRENT has been marked SOLVED or its h ' has just changed, its new status must be propagate backwards up the graph . hence all the ancestors of CURRENT are addedto S.

(Refered From Artificial Intelligence TMH)AO* Search Procedure.

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 isthe 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, exitwith 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)

Note: AO* will always find minimum cost solution.

CONSTRAINT SATISFACTION:-

Many problems in AI can be considered as problems of constraint satisfaction, in which the goalstate satisfies a given set of constraint. constraint satisfaction problems can be solved by using any of the search strategies. The general form of the constraint satisfaction procedure is as follows:

Until a complete solution is found or until all paths have led to lead ends, do

1. select an unexpanded node of the search graph.

2. Apply the constraint inference rules to the selected node to generate all possible newconstraints.

3. If the set of constraints contains a contradiction, then report that this path is a dead end.

4. If the set of constraints describes a complete solution then report success.

5. If neither a constraint nor a complete solution has been found then apply the rules to generatenew partial solutions. Insert these partial solutions into the search graph.

Example: consider the crypt arithmetic problems.

```
SEND
+ MORE
 -------
 MONEY
 -------
```

Assign decimal digit to each of the letters in such a way that the answer to the problem is correctto the same letter occurs more than once , it must be assign the same digit each time . no two different letters may be assigned the same digit. Consider the crypt arithmetic problem.

```
SEND
+ MORE
 -------
 MONEY
 -------
```

CONSTRAINTS:-

1. no two digit can be assigned to same letter.

2. only single digit number can be assign to a letter.

1. no two letters can be assigned same digit.

2. Assumption can be made at various levels such that they do not contradict each other.

3. The problem can be decomposed into secured constraints. A constraint satisfaction approachmay be used.

4. Any of search techniques may be used.

5. Backtracking may be performed as applicable us applied search techniques.

6. Rule of arithmetic may be followed.

Initial state of problem.
D=?
E=?
Y=?
N=?
R=?
O=?
S=?
M=? C1=?C2=?
C1 ,C 2, C3 stands for the carry variables respectively.

Goal State: the digits to the letters must be assigned in such a manner so that the sum is satisfied.

Solution Process:

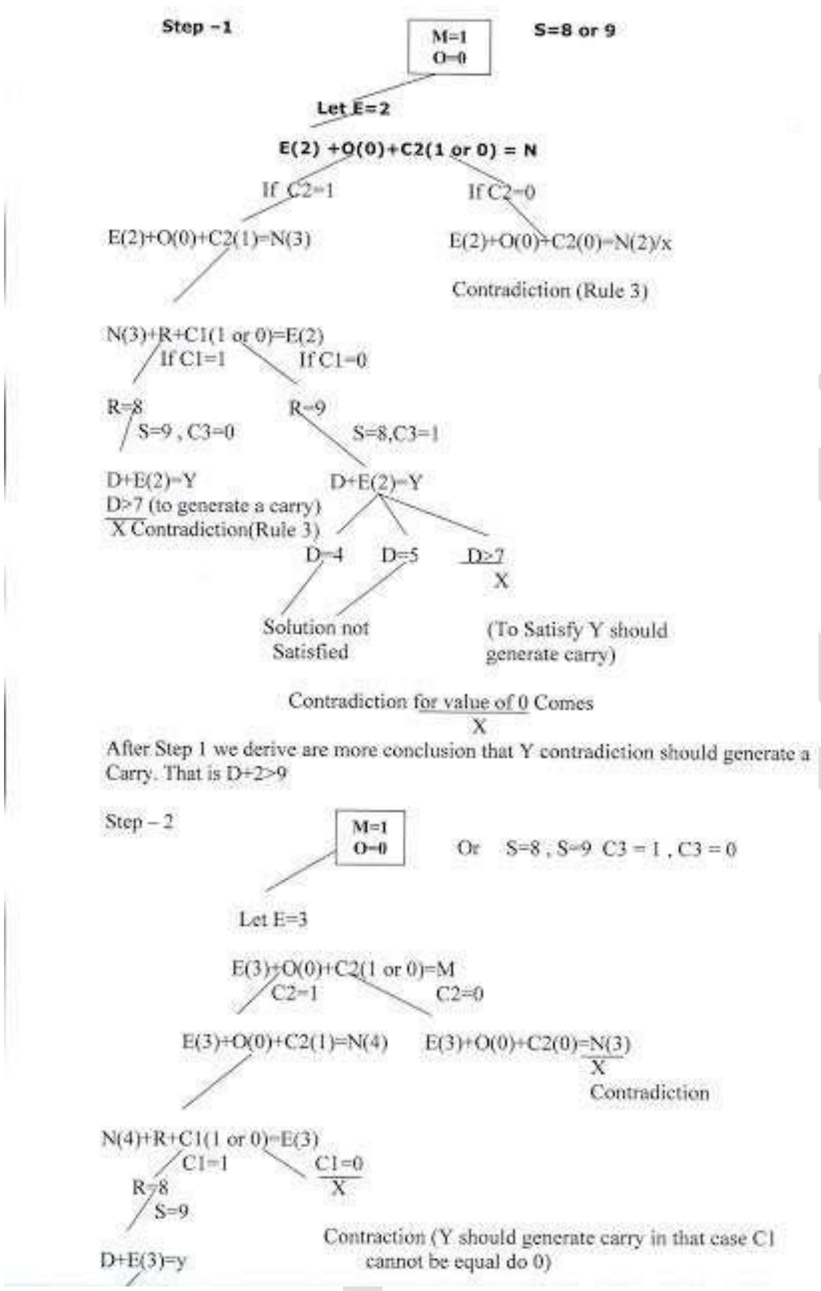
We are following the depth-first method to solve the problem.

1. initial guess m=1 because the sum of two single digits can generate at most a carry '1'.

2. When n=1 o=0 or 1 because the largest single digit number added to m=1 can generate the sum of either 0 or 1 depend on the carry received from the carry sum. By this we conclude thato=0 because m is already 1 hence we cannot assign same digit another letter(rule no.)

3. We have m=1 and o=0 to get o=0 we have s=8 or 9, again depending on the carry receivedfrom the earlier sum.

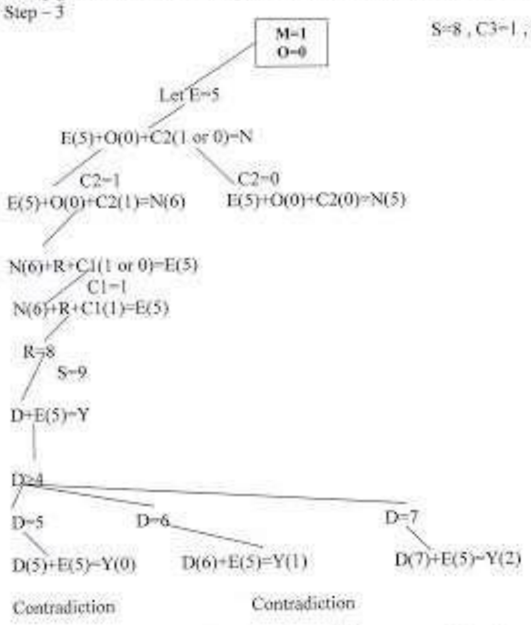
The same process can be repeated further. The problem has to be composed into various constraints. And each constraints is to be satisfied by guessing the possible digits that the letterscan be assumed that the initial guess has been already made . rest of the process is being shownin the form of a tree, using depth-first search for the clear understandability of the solution process.



D>6(Controduction)

After Step 2 , we found that C1 cannot be Zero, Since Y has to generate a carry to satisfy goal state. From this step onwards, no need to branch for C1=0.

Step – 3



At Step (4) we have assigned a single digit to every letter in accordance with the constraints & production rules.

Now by backtracking , we find the different digits assigned to different letters and hence reach the solution state.

Solution State:-

Y = 2
D = 7
S = 9
R = 8
N = 6
E = 5
O = 0
M = 1
C1 = 1
C2 = 0
C3 = 0

	C3(0)	C2(1)	C1(1)	
	S(9)	E(5)	N(6)	D(7)
+	M(1)	O(0)	R(8)	E(5)
M(1)	O(0)	N(6)	E(5)	Y(2)

MEANS - ENDS ANALYSIS:-

Most of the search strategies either reason forward of backward however, often a mixture o the two directions is appropriate. Such mixed strategy would make it possible to solve the major parts of problem first and solve the smaller problems the arise when combining them together. Such a technique is called "Means - Ends Analysis".

The means -ends analysis process centers around finding the difference between current state and goal state. The problem space of means - ends analysis has an initial state and one or more goal state, a set of operate with a set of preconditions their application and difference functions that computes the difference between two state a(i) and s(j). A problem is solved using means - ends analysis by

1. Computing the current state s1 to a goal state s2 and computing their difference D12.

2. Satisfy the preconditions for some recommended operator op is selected, then to reduce the difference D12.

3. The operator OP is applied if possible. If not the current state is solved a goal is created and means- ends analysis is applied recursively to reduce the sub goal.

4. If the sub goal is solved state is restored and work resumed on the original problem.

(the first AI program to use means - ends analysis was the GPS General problem solver)

means- ends analysis I useful for many human planning activities. Consider the example of planing for an office worker. Suppose we have a different table of three rules:

1. If in out current state we are hungry , and in our goal state we are not hungry , then either the"visit hotel" or "visit Canteen " operator is recommended.

2. If our current state we do not have money , and if in your goal state we have money, then the "Visit our bank" operator or the "Visit secretary" operator is recommended.

3. If our current state we do not know where something is , need in our goal state we do know, then either the "visit office enquiry" , "visit secretary" or "visit co worker " operator is recommended.