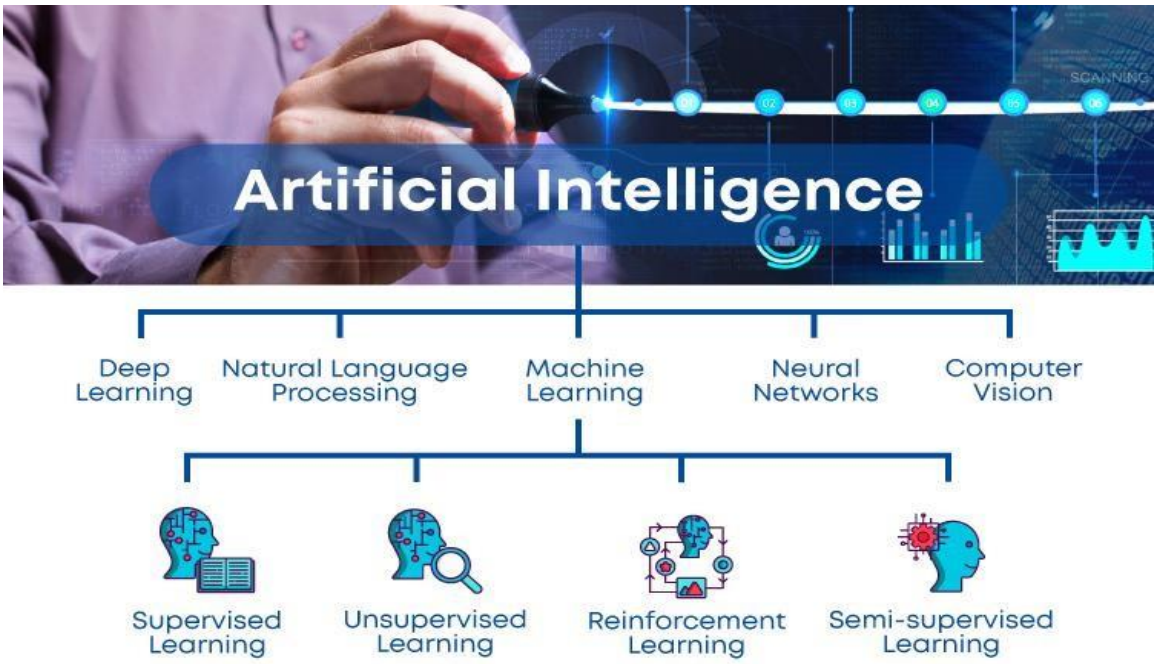


1. What is Artificial Intelligence?

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".
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

DATA : Unformatted information and Raw Facts – Numeric or Categorical

INFORMATION: It is the result of processing and organizing the data in response to a specific field

KNOWLEDGE : Understanding the problem statement and its domain (What to do?) E.g., Finance, Manufacturing, Sales, Forecasting etc.

INTELLIGENCE : The ability to acquire and apply the knowledge and skills to the machines

2. Problems associated with Artificial Intelligence?

Common Sense Reading: AI focuses on reasoning about physical objects and their relationships with each other. It also reasons the actions and other consequences

Game Playing and Theorem Proving: These share the properties that people who do well overall, and are intelligent

General Problem Solver (GPS): It is the problem of applying symbolic manipulators to the logical expressions

3. Task/Application Domains of Artificial Intelligence?

Task Domains of AI

- Mundane Tasks:
  - Perception
    - Vision
    - Speech
  - Natural Languages
    - Understanding
    - Generation
    - Translation
  - Common sense reasoning
  - Robot Control
- Formal Tasks
  - Games : chess, checkers etc
  - Mathematics: Geometry, logic,Proving properties of programs
- Expert Tasks:
  - Engineering (Design, Fault finding, Manufacturing planning)
  - Scientific Analysis
  - Medical Diagnosis
  - Financial Analysis

4. What is Physical System Hypothesis?

Artificial Intelligence practitioners define a physical system as

1. Symbols
2. Expressions
3. Symbol Structures
4. Systems

It contains the process so that it can act upon the symbols and Expressions
Computers provide the perfect medium for the experimentation since they can be programmed to simulate physical system symbol that we like

5. What are Artificial Intelligence Technique (Important)?

There are appropriate techniques to solve multi-domain related problems.

1. The knowledge captures Generalizations
2. It can be understood by the people
3. It can be easily modified to correct the errors and reflect changes in the world
4. It can be used in many situations even if it is not totally accurate or complete
5. It can narrow down the range of possibilities for error

Important AI Techniques

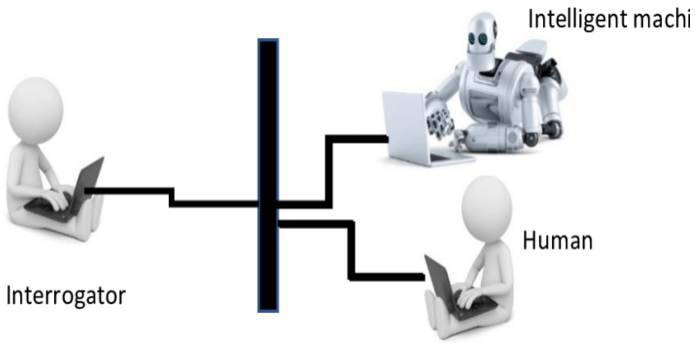
Search: Provides a problem-solving approach where there is no direction or approach available

Use of Knowledge: Provides a way of Solving Complex Problems by exploring the structures of objects that are involved

Abstraction: Provides a way to show the necessary and relevant features by hiding the unimportant ones

6. Explain Turing Test?

In 1950, Alan Turing proposed a method for determining whether a machine can artificially think



One person A plays the role of interrogator. Questions are asked by him to both computer and other human, and simultaneous answers are received from both

The trick is to determine the difference between Human answer and Computer answer

IF THE INTERROGATOR FAILS TO DETERMINE THE DIFFERENCE, IT MEANS THE MACHINE HAS PASSED THE TEST

Less Desirable Properties of Knowledge

1. It is huge
2. It is difficult to characterize correctly
3. It is constantly varying
4. It differs from data to data by being organized in a way that it corresponds to its applications . It is Complicated

7. What are the Steps to Solve Problems in Artificial Intelligence

Steps to solve the problem of building an Artificial Intelligence system:

1. Define the problem accurately including detailed specification and determine the suitable solution
2. Scrutinize(double-check) the problem carefully as some features may have effect on the chosen method of solution(Algorithm)
3. Segregate(separate) the background knowledge needed in the solution of the problem
4. Choose the method having highest accuracy of solution

8. Explain Tic-Tac-Toe problem & How it can be solved using AI Techniques (Important)?

A. Tic-Tac-Toe problem

1	**2**	**3**
4	**5**	**6**
7	**8**	**9**

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 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.

B. 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 centre 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.

C. 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.

9. What is Question Answering?

Let us consider Question Answering systems that accept input in English and provide answers also in English.

For example, consider the following situation:
Text
Rani went shopping for a new Coat. She found a red one she really liked.
When she got home, she found that it went perfectly with her favourite dress.
Question
1. What did Rani go shopping for?
2. What did Rani find that she liked?
3. Did Rani buy anything?
Method 1
Data Structures

A set of templates that match common questions and produce patterns used to match against inputs.

Templates and patterns are used so that a template that matches a given question is associated with the corresponding pattern to find the answer in the input text. For example, the template who did x y generates x y z if a match occurs and z is the answer to the question. The given textand the question are both stored as strings.

Algorithm

Answering a question requires the following four steps to be followed:

Compare the template against the questions and store all successful matches to produce a set of text patterns. Pass these text patterns through a substitution process to change the person or voice and produce an expanded set of text patterns.

Apply each of these patterns to the text; collect all the answers and then print the answers.

Example

In question 1 we use the template WHAT DID X Y which generates Rani go shopping for z and after substitution we get Rani goes shopping for z and Rani went shopping for z giving z [equivalence] a new coat

In question 2 we need a very large number of templates and also a scheme to allow the insertion of „find" before „that she liked"; the insertion of „really" in the text; and the substitution of „she" for „Rani" gives the answer „a red one".

Question 3 cannot be answered.

Comments

This is a very primitive approach basically not matching the criteria we set for intelligence and worse than that, used in the game. Surprisingly this type of technique was actually used in ELIZA which will be considered later in the course.

Method 2

Data Structures

Take, for example sentence:

„She found a red one she really liked".

Event2	**Event2**
instance: finding	instance: liking
tense: past	tense: past
agent: Rani	modifier: much
object: Thing1	object: Thing1

Thing1
instance: coat
colour: red

The question is stored in two forms: as input and in the above form.

Algorithm

*Convert the question to a structured form using English know how, then use a marker to indicate the substring (like „who" or „what") of the structure, that should be returned as an answer. If a slot and filler system is used a special marker can be placed in more than one slot.
*The answer appears by matching this structured form against the structured text.
*The structured form is matched against the text and the requested segments of the question are returned.

Method 3

Data Structures

World model contains knowledge about objects, actions and situations that are described in the input text. This structure is used to create integrated text from input text.

Algorithm

Convert the question to a structured form using both the knowledge contained in Method 2 and the World model, generating even more possible structures, since even more knowledge is being used.
Sometimes filters are introduced to prune the possible answers.
To answer a question, the scheme followed is: Convert the question to a structured form as before but use the world model to resolve any ambiguities that may occur. The structured form is matched against the text and the requested segments of the question are returned.

10. LEVEL OF THE AI MODEL?

1. To test psychological theories of human performance. Ex. PARRY [Colby, 1975] – a program to simulate the conversational behavior of a paranoid person.
2. To enable computers to understand human reasoning – for example, programs that answer questions based upon newspaper articles indicating human behavior.
3. To enable people to understand computer reasoning. Some people are reluctant to accept computer results unless they understand the mechanisms involved in arriving at the

results.

4. To exploit the knowledge gained by people who are best at gathering information. This persuaded the earlier workers to simulate human behavior in the SB part of AISB simulated behavior. Examples of this type of approach led to GPS (General Problem Solver)

Lab Program

1. **Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.**

What is Regression?

Regression analysis is a statistical method that helps us to analyse and understand the relationship between two or more variables of interest. The process that is adapted to perform regression analysis helps to understand which factors are important, which factors can be ignored and how they are influencing each other.

Regression is used for continuous values

● Dependent Variable: This is the variable that we are trying to understand or forecast.

● Independent Variable: These are factors that influence the analysis or target variable and provide us with information regarding the relationship of the variables with the target variable

What is Locally weighted Linear Regression?

● Locally weighted linear regression is a non-parametric algorithm

● The model does not learn a fixed set of parameters as is done in ordinary linear regression.

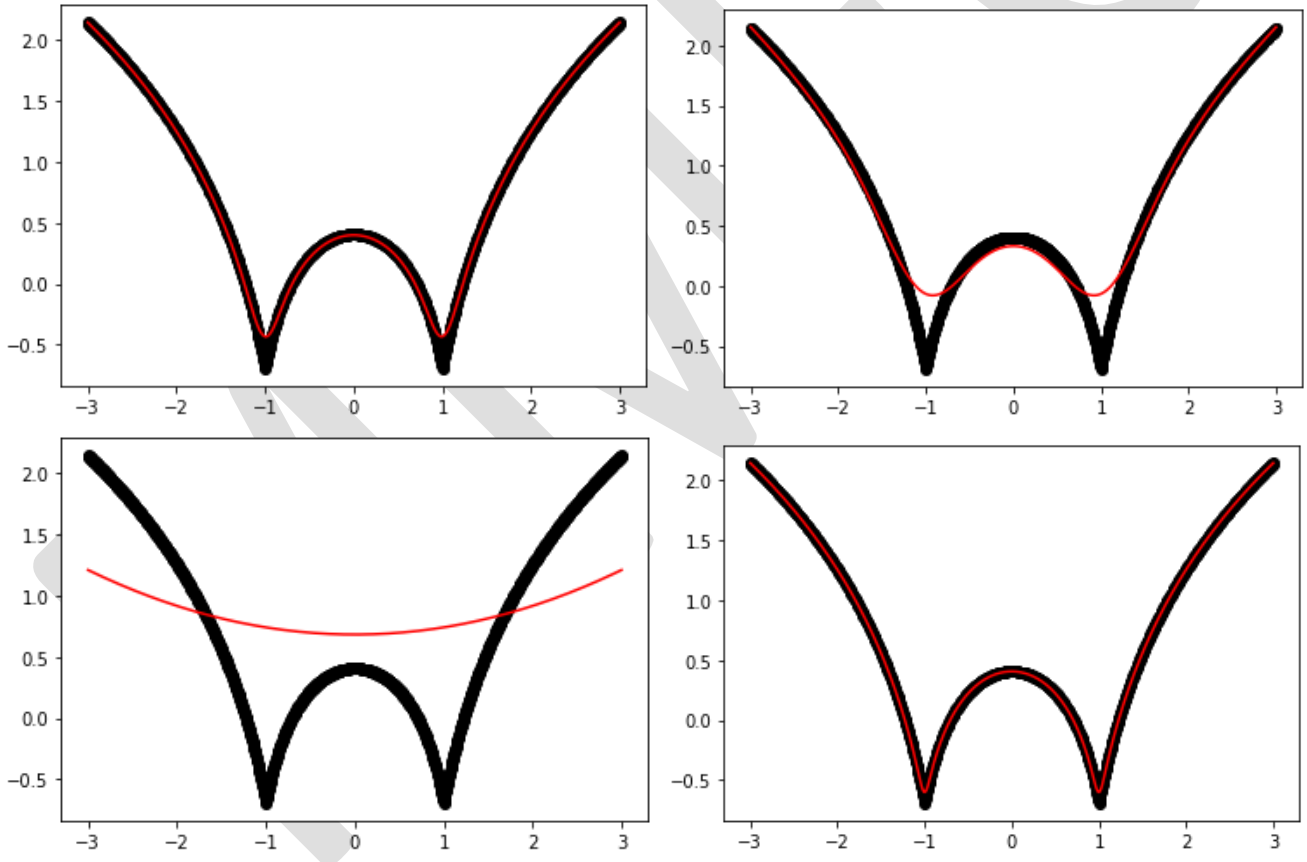
● Rather parameters theta are computed individually for each query point x.

CODE

```
import numpy as np
import matplotlib.pyplot as plt

def local_regression(x0, X, Y, tau):
    x0 = [1, x0]
    X = [[1, i] for i in X]
    X = np.asarray(X)
    xw = (X.T) * np.exp(np.sum((X - x0) ** 2, axis=1) / (-2 * tau))
    beta = np.linalg.pinv(xw @ X) @ xw @ Y @ x0
    return beta

def draw(tau):
    prediction = [local_regression(x0, X, Y, tau) for x0 in domain]
    plt.plot(X, Y, 'o', color='black')
    plt.plot(domain, prediction, color='red')
```

```
    plt.show()

X = np.linspace(-3, 3, num=1000)
domain = X
Y = np.log(np.abs(X ** 2 - 1) + .5)

draw(10)
draw(0.1)
draw(0.01)
draw(0.001)
```

Output



Here's a step-by-step explanation of the code:

1. Import necessary libraries: **numpy** for numerical operations and **matplotlib.pyplot** for data visualization.
2. Define the **local_regression** function:
    - This function takes four parameters:
        - **x0**: The point at which we want to estimate the regression.
        - **X**: The input features (independent variables).
        - **Y**: The corresponding target values (dependent variable).
        - **tau**: The bandwidth parameter that controls the width of the neighborhood for local regression.
    - The function first prepares the input data for regression by appending a constant 1 to each feature (for the intercept term) and exponentiating part of the weighted distance calculation.
    - It calculates a weighted linear regression using the specified bandwidth (**tau**) and returns the coefficients of the linear model.
3. Define the **draw** function:
    - This function takes one parameter, **tau**, and is responsible for drawing the local regression line based on the given **tau**.
    - It calls the **local_regression** function for each data point in the **domain** (X-axis values) to predict the corresponding Y-values.
    - Then, it plots the original data points (**X** and **Y**) as black dots and the predicted values as a red line.
4. Generate the dataset:
    - Generate **X** values (input features) using **numpy.linspace** to create 1000 evenly spaced values from -3 to 3.
    - Define the **domain** as the same as **X**.
    - Generate **Y** values (target values) based on a mathematical function: **np.log(np.abs(X ** 2 - 1) + .5)**. This function creates a curve.
5. Call the **draw** function four times with different **tau** values:
    - **draw(10)**: Draws the local regression line with a bandwidth of 10.
    - **draw(0.1)**: Draws the local regression line with a bandwidth of 0.1.
    - **draw(0.01)**: Draws the local regression line with a bandwidth of 0.01.
    - **draw(0.001)**: Draws the local regression line with a bandwidth of 0.001.

I. ASSIGNMENT QUESTIONS (25-Sept-2023)

1. **What is Artificial Intelligence?**
   a. **Data, Information, Knowledge, Intelligence?**
   b. **Explain Turning Test?**
2. **Explain Task Domain/Application in AI?**
   a. **What are Artificial Intelligence Technique?**
3. **Explain Tic-Tac-Toe problem & How it can be solved using AI Techniques?**
4. **What is Question Answering & explain all three methods?**
5. **<u>Lab Program</u>: Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.**

PROBLEMS, PROBLEM SPACE AND SEARCH

To solve the problem of playing a game, we require the rules of the game and targets for winning as well as representing positions in the game. The opening position can be defined as the initial state and a winning position as a goal state. Moves from initial state to other states leading to the goal state follow legally. However, the rules are far too abundant in most games— especially in chess, where they exceed the number of particles in the universe. Thus, the rules cannot be supplied accurately and computer programs cannot handle easily. The storage also presents another problem but searching can be achieved by hashing.

The number of rules that are used must be minimized and the set can be created by expressing each rule in a form as possible. The representation of games leads to a state space representation and it is common for well-organized games with some structure. This representation allows for the formal definition of a problem that needs the movement from a set of initial positions to one of a set of target positions. It means that the solution involves using known techniques and a systematic search. This is quite a common method in Artificial Intelligence.

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.

Methodology of State Space Approach

1. To represent a problem in structured form using different states
2. Identify the initial state
3. Identify the goal state
4. Determine the operator for the changing state
5. Represent the knowledge present in the problem in a convenient form
6. Start from the initial state and search a path to goal state

To build a program that could “Play Chess”

- we have to first specify the starting position of the chess board

o Each position can be described by an 8-by-8 array.
o Initial position is the game opening position.

- rules that define the legal moves

We must make explicit the preciously implicit goal of not only playing a legal game of

chess but also winning the game, if possible.

- Legal moves can be described by a set of rules:
  - □ Left sides are matched against the current state.
  - □ Right sides describe the new resulting state.
  - □ board positions that represent a win for one side or the other

- Goal position is any position in which the opponent does not have a legal move and his or her king is under attack.
- We must make explicit the preciously implicit goal of not only playing a legal game of chess but also winning the game, if possible.

The 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. The major production rules for solving this problem are shown below:

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

(Fig. 2.1 Production Rules for the Water Jug Problem)

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

(Fig. 2.2One Solution to the Water Jug Problem)

The problem solved by using the production rules in combination with an appropriate control strategy, moving through the problem space until a path from an initial state to a goal state is found. In this problem solving process, search is the fundamental concept. For simple problems it is easier to achieve this goal by hand but there will be cases where this is far too difficult.

2. Problems, Problem Spaces and Search

Problem:

A problem, which can be caused for different reasons, and, if solvable, can usually be solved in a number of different ways, is defined in a number of different ways.

To build a system or to solve a particular problem we need to do four things.

1. Define the problem precisely. This definition must include precise specification of what the initial situation will be as well as what final situations constitute acceptable solutions to the problem
2. Analyze the problem
3. Isolate and represent the task knowledge that is necessary to solve the problem
4. Choose the best solving technique and apply it to the particular problem.

Defining the Problem as a State Space Search

Problem solving = Searching for a goal state

It is a structured method for solving an unstructured problem. This approach consists of number of states. The starting of the problem is "Initial State" of the problem. The last point in the problem is called a "Goal State" or "Final State" of the problem.

State space is a set of legal positions, starting at the initial state, using the set of rules to move from one state to another and attempting to end up in a goal state.

Methodology of State Space Approach

1. To represent a problem in structured form using different states
2. Identify the initial state
3. Identify the goal state
4. Determine the operator for the changing state
5. Represent the knowledge present in the problem in a convenient form
6. Start from the initial state and search a path to goal state

To build a program that could "Play Chess"

- we have to first specify the starting position of the chess board
  - Each position can be described by an 8-by-8 array.
  - Initial position is the game opening position.
- rules that define the legal moves
  - Legal moves can be described by a set of rules:
    - Left sides are matched against the current state.
    - Right sides describe the new resulting state.
- board positions that represent a win for one side or the other
  - Goal position is any position in which the opponent does not have a legal move and his or her king is under attack.

- We must make explicit the preciously implicit goal of not only playing a legal game of chess but also winning the game, if possible.

Production System

The entire procedure for getting a solution for AI problem can be viewed as "Production System". It provides the desired goal. It is a basic building block which describes the AI problem and also describes the method of searching the goal. Its main components are:

- A Set of Rules, each consisting of a left side (a pattern) that determines the applicability of the rule and right side that describes the operation to be performed if the rule is applied.
- Knowledge Base – It contains whatever information is appropriate for a particular task. Some parts of the database may be permanent, while the parts of it may pertain only to the solution of the current problem.
- Control Strategy – It specifies the order in which the rules will be compared to the database and the way of resolving the conflicts that arise when several rules match at one.
    - The first requirement of a goal control strategy is that it is cause motion; a control strategy that does not cause motion will never lead to a solution.
    - The second requirement of a good control strategy is that it should be systematic.
- A rule applier: Production rule is like below

```
if(condition) then
        consequence or action
```

Algorithm for Production System:

1. Represent the initial state of the problem
2. If the present state is the goal state then go to step 5 else go to step 3
3. Choose one of the rules that satisfy the present state, apply it and change the state to new state.
4. Go to Step 2
5. Print "Goal is reached " and indicate the search path from initial state to goal state
6. Stop

Classification of Production System:

Based on the direction they can be

1. Forward Production System
    - Moving from Initial State to Goal State
    - When there are number of goal states and only one initial state, it is advantage to use forward production system.
2. Backward Production System
    - Moving from Goal State to Initial State
    - If there is only one goal state and many initial states, it is advantage to use backward production system.

Production System Characteristics

Production system is a good way to describe the operations that can be performed in a search for solution of the problem.

Two questions we might reasonably ask at this point are:

- *Can production systems, like problems, be described by a set of characteristics that shed some light on how they can easily be implemented?*
- *If so, what relationships are there between problem types and the types of production systems best suited to solving the problems?*

The answer for the first question can be considered with the following *definitions of classes of production systems*:

A monotonic production system is a production system in which the applications of a rule never prevents the later application of another rule that could also have been applied at the time the first rule was selected.

A non-monotonic production system is one which this is not true.

A partially commutative production system is a production system with the property that if the application of a particular sequence of rules transforms state X into state Y, then any permutation of those rules that is allowable also transforms state X into state Y.

A commutative production system is a production system that is both monotonic and partially commutative.

In a formal sense, there is no relationship between kinds of problems and kinds of production of systems, since all problems can be solved by all kinds of systems. But in practical sense, there definitely is such a relationship between kinds of problems and the kinds of systems that led themselves naturally to describing those problems.

The following figure shows the four categories of production systems produced by the two dichotomies, monotonic versus non-monotonic and partially commutative versus non-partially commutative along with some problems that can be naturally be solved by each type of system.

	Monotonic	Non-monotonic
Partially commutative	Theorem proving	Robot Navigation
Not Partially commutative	Chemical Synthesis	Bridge

The four categories of Production Systems

- Partially commutative, monotonic production systems are useful for solving ignorable problems that involves creating new things rather than changing old ones generally ignorable. Theorem proving is one example of such a creative process partially commutative, monotonic production system are important for a implementation stand point because they can be implemented without the ability to backtrack to previous states when it is discovered that an incorrect path has been followed.
- Non-monotonic, partially commutative production systems are useful for problems in which changes occur but can be reversed and in which order of operations is not critical.

This is usually the case in physical manipulation problems such as "Robot navigation on a flat plane". The 8-puzzle and blocks world problem can be considered partially commutative production systems are significant from an implementation point of view because they tend to read too much duplication of individual states during the search process.

- Production systems that are not partially commutative are useful for many problems in which changes occur. For example "Chemical Synthesis"
- Non-partially commutative production system less likely to produce the same node many times in the search process.

Problem Characteristics

In order to choose the most appropriate method (or a combination of methods) for a particular problem, it is necessary to analyze the problem along several key dimensions:

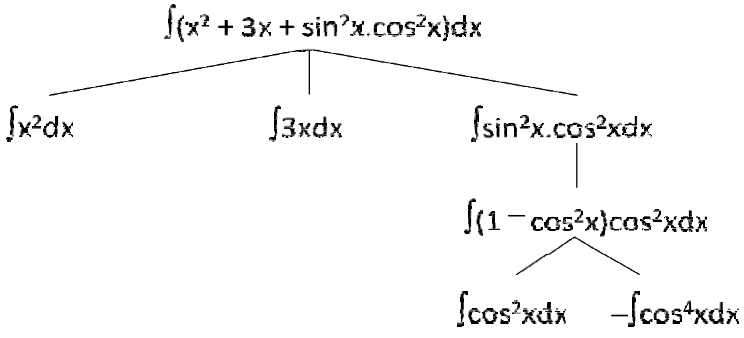
- Is the problem decomposable?
- Can solution steps be ignored or undone?
- Is the universe predictable?
- Is a good solution absolute or relative?
- Is the solution a state or a path?
- What is the role of knowledge?
- Does the task require human-interaction?
- Problem Classification

Is the problem decomposable?

Decomposable problem can be solved easily. Suppose we want to solve the problem of computing the expression.

$$\int (x^2 + 3x + \sin^2 x.\cos^2 x)\, dx$$

We can solve this problem by breaking it down into these smaller problems, each of which we can then solve by using a small collection of specific rules the following figure shows problem tree that as it can be exploited by a simple recursive integration program that works as follows.



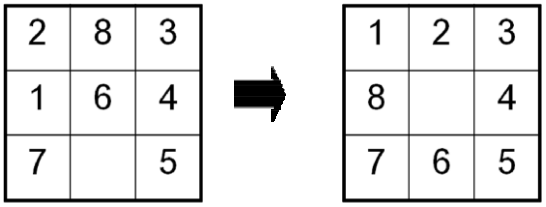
At each step it checks to see whether the problem it is working on is immediately solvable. If so, then the answer is returned directly. If the problem is not easily solvable, the integrator checks to

see whether it can decompose the problem into smaller problems. It can create those problems and calls itself recursively on using this technique of problem decomposition we can often solve very large problem easily.

Can solution steps be ignored or undone?

Suppose we are trying to prove a mathematical theorem. We proceed by first proving a lemma that we think will be useful. A lemma that has been proved can be ignored for next steps as eventually we realize the lemma is no help at all.

Now consider the 8-puzzle game. A sample game using the 8-puzzle is shown below:



In attempting to solve the 8 puzzle, we might make a stupid move for example; we slide the tile 5 into an empty space. We actually want to slide the tile 6 into empty space but we can back track and undo the first move, sliding tile 5 back to where it was then we can know tile 6 so mistake and still recovered from but not quit as easy as in the theorem moving problem. An additional step must be performed to undo each incorrect step.

Now consider the problem of playing chess. Suppose a chess playing problem makes a stupid move and realize a couple of moves later. But here solutions steps cannot be undone.

The above three problems illustrate difference between three important classes of problems:

1) Ignorable: in which solution steps can be ignored.
   Example: Theorem Proving
2) Recoverable: in which solution steps can be undone.
   Example: 8-Puzzle
3) Irrecoverable: in which solution steps cannot be undone.
   Example: Chess

The recoverability of a problem plays an important role in determining the complexity of the control structure necessary for problem solution.

Ignorable problems can be solved using a simple control structure that never backtracks. *Recoverable problems* can be solved by slightly complicated control strategy that does sometimes make mistakes using backtracking. *Irrecoverable problems* can be solved by recoverable style methods via planning that expands a great deal of effort making each decision since the decision is final.

Is the universe predictable?

There are certain outcomes every time we make a move we will know what exactly happen. This means it is possible to plan entire sequence of moves and be confident that we know what the resulting state will be. Example is 8-Puzzle.

In the uncertain problems, this planning process may not be possible. Example: Bridge Game – Playing Bridge. We cannot know exactly where all the cards are or what the other players will do on their turns.

We can do fairly well since we have available accurate estimates of a probabilities of each of the possible outcomes. A few examples of such problems are

- Controlling a robot arm: The outcome is uncertain for a variety of reasons. Someone might move something into the path of the arm. The gears of the arm might stick.
- Helping a lawyer decide how to defend his client against a murder charge. Here we probably cannot even list all the possible outcomes, which leads outcome to be uncertain.

- *For certain-outcome problems, planning can used to generate a sequence of operators that is guaranteed to lead to a solution.*
- *For uncertain-outcome problems, a sequence of generated operators can only have a good probability of leading to a solution.*
- *Plan revision is made as the plan is carried out and the necessary feedback is provided.*

Is a Good Solution Absolute or Relative?

Consider the problem of answering questions based on a database of simple facts, such as the following:

1) Marcus was a man.
2) Marcus was a Pompeian.
3) Marcus was born in 40 A.D.
4) All men are mortal.
5) All Pompeian's died when the volcano erupted in 79 A.D.
6) No mortal lives longer than 150 years.
7) It is now 1991 A.D.

Suppose we ask a question "Is Marcus alive?" By representing each of these facts in a formal language such as predicate logic, and then using formal inference methods we can fairly easily derive an answer to the question.

	Justification
1. Marcus was a man.	axiom 1
4. All men are mortal.	axiom 4
8. Marcus is mortal.	1, 4
3. Marcus was born in 40 A.D.	axiom 3
7. It is now 1991 A.D.	axiom 7
9. Marcus' age is 1951 years.	3, 7
6. No mortal lives longer than 150 years.	axiom 6
10. Marcus is dead.	8, 6, 9
OR	
7. It is now 1991 A.D.	axiom 7
5. All Pompeians died in 79 A.D.	axiom 5
11. All Pompeians are dead now.	7, 5
2. Marcus was a Pompeian.	axiom 2
12. Marcus is dead.	11, 2

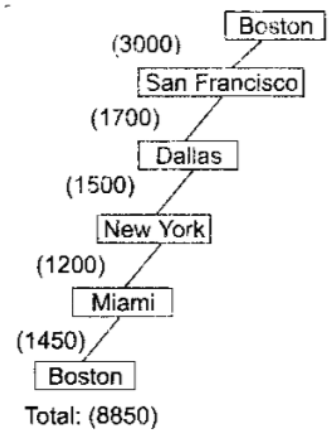
Two Ways of Deciding That Marcus Is Dead

Since we are interested in the answer to the question, it does not matter which path we follow. If we do follow one path successfully to the answer, there is no reason to go back and see if some other path might also lead to a solution. These types of problems are called as "Any path Problems".

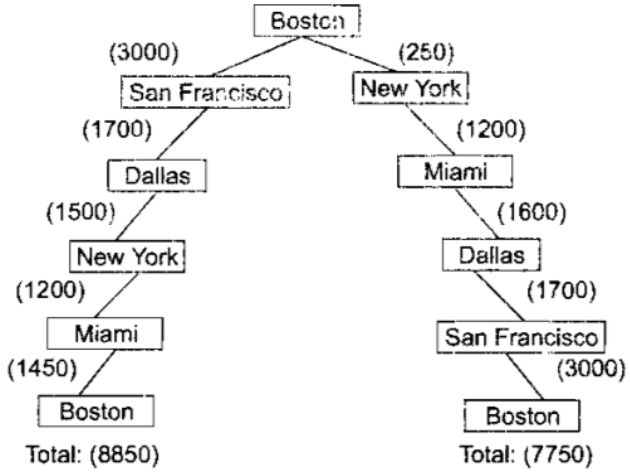
Now consider the Travelling Salesman Problem. Our goal is to find the shortest path route that visits each city exactly once.

	Boston	New York	Miami	Dallas	S.F.
Boston		250	1450	1700	3000
New York	250		1200	1500	2900
Miami	1450	1200		1600	3300
Dallas	1700	1500	1600		1700
S.F.	3000	2900	3300	1700	

An Instance of the Traveling Salesman Problem



One Path among the Cities



Two Paths Among the Cities

Suppose we find a path it may not be a solution to the problem. We also try all other paths. The shortest path (best path) is called as a solution to the problem. These types of problems are known as "Best path" problems. But path problems are computationally harder than any path problems.

Is the solution a state or a path?

Consider the problem of finding a consistent interpretation for the sentence

The bank president ate a dish of pasta salad with the fork

There are several components of this sentence, each of which may have more than one interpretation. Some of the sources of ambiguity in this sentence are the following:

- The word "Bank" may refer either to a financed institution or to a side of river. But only one of these may have a President.
- The word "dish" is the object of the word "eat". It is possible that a dish was eaten.
- But it is more likely that the pasta salad in the dish was eaten.

Because of the interaction among the interpretations of the constituents of the sentence some search may be required to find a complete interpreter for the sentence. But to solve the problem

of finding the interpretation we need to produce only the interpretation itself. No record of the processing by which the interpretation was found is necessary. But with the "water-jug" problem it is not sufficient to report the final state we have to show the "path" also.

So the solution of natural language understanding problem is a state of the world. And the solution of "Water jug" problem is a path to a state.

What is the role of knowledge?

Consider the problem of playing chess. The knowledge required for this problem is the rules for determining legal move and some simple control mechanism that implements an appropriate search procedure.

Now consider the problem of scanning daily newspapers to decide which are supporting 'n' party and which are supporting 'y' party. For this problems are required lot of knowledge.

The above two problems illustrate the difference between the problems for which a lot of knowledge is important only to constrain the search for a solution and those for which a lot of knowledge is required even to be able to recognize a solution.

Does a task require interaction with the person?

Suppose that we are trying to prove some new very difficult theorem. We might demand a prove that follows traditional patterns so that mathematician each read the prove and check to make sure it is correct. Alternatively, finding a proof of the theorem might be sufficiently difficult that the program does not know where to start. At the moment people are still better at doing the highest level strategies required for a proof. So that the computer might like to be able to ask for advice.

For Example:

- Solitary problem, in which there is no intermediate communication and no demand for an explanation of the reasoning process.
- Conversational problem, in which intermediate communication is to provide either additional assistance to the computer or additional information to the user.

Problem Classification

When actual problems are examined from the point of view all of these questions it becomes apparent that there are several broad classes into which the problem fall. The classes can be each associated with a generic control strategy that is approached for solving the problem. There is a variety of problem-solving methods, but there is no one single way of solving all problems. Not all new problems should be considered as totally new. Solutions of similar problems can be exploited.

PROBLEMS

Water-Jug Problem

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

Solution:

The state space for the problem can be described as a set of states, where each state represents the number of gallons in each state. The game start with the initial state described as a set of ordered pairs of integers:

- State: (x, y)
  - x = number of lts in 4 lts jug
  - y = number of lts in 3 lts jug

  x = 0, 1, 2, 3, or 4 y = 0, 1, 2, 3
- Start state: (0, 0) i.e., 4-litre and 3-litre jugs is empty initially.
- Goal state: (2, n) for any n that is 4-litre jug has 2 litres of water and 3-litre jug has any value from 0-3 since it is not specified.
- Attempting to end up in a goal state.

Production Rules: These rules are used as operators to solve the problem. They are represented as rules whose left sides are used to describe new state that result from approaching the rule.

1	(x, y) if $x < 4$	→	$(4, y)$	Fill the 4-gallon jug
2	(x, y) if $y < 3$	→	$(x, 3)$	Fill the 3-gallon jug
3	(x, y) if $x > 0$	→	$(x - d, y)$	Pour some water out of the 4-gallon jug
4	(x, y) if $y > 0$	→	$(x, y - d)$	Pour some water out of the 3-gallon jug
5	(x, y) if $x > 0$	→	$(0, y)$	Empty the 4-gallon jug on the ground
6	(x, y) if $y > 0$	→	$(x, 0)$	Empty the 3-gallon jug on the ground
7	(x, y) if $x + y \geq 4$ and $y > 0$	→	$(4, y - (4 - x))$	Pour water from the 3-gallon jug into the 4-gallon jug until the 4-gallon jug is full
8	(x, y) if $x + y \geq 3$ and $x > 0$	→	$(x - (3 - y), 3)$	Pour water from the 4-gallon jug into the 3-gallon jug until the 3-gallon jug is full
9	(x, y) if $x + y \leq 4$ and $y > 0$	→	$(x + y, 0)$	Pour all the water from the 3-gallon jug into the 4-gallon jug
10	(x, y) if $x + y \leq 3$ and $x > 0$	→	$(0, x + y)$	Pour all the water from the 4-gallon jug into the 3-gallon jug
11	$(0, 2)$	→	$(2, 0)$	Pour the 2 gallons from the 3-gallon jug into the 4-gallon jug
12	$(2, y)$	→	$(0, y)$	Empty the 2 gallons in the 4-gallon jug on the ground

Fig. 2.3 *Production Rules for the Water Jug Problem*

The solution to the water-jug problem is:

Gallons in the 4-Gallon Jug	Gallons in the 3-Gallon Jug	Rule Applied
0	0	
		2
0	3	
		9
3	0	
		2
3	3	
		7
4	2	
		5 or 12
0	2	
		9 or 11
2	0	

One Solution to the Water Jug Problem

Chess Problem

Problem of playing chess can be defined as a problem of moving around in a state space where each state represents a legal position of the chess board.

The game start with an initial state described as an 8x8 of each position contains symbol standing for the appropriate place in the official chess opening position. A set of rules is used to move from one state to another and attempting to end up on one of a set of final states which is described as any board position in which the opponent does not have a legal move as his/her king is under attacks.

The state space representation is natural for chess. Since each state corresponds to a board position i.e. artificial well organized.

Initial State: Legal chess opening position
Goal State: Opponent does not have any legal move/king under attack

Production Rules:
These rules are used to move around the state space. They can be described easily as a set of rules consisting of two parts:

1. Left side serves as a pattern to be matching against the current board position.
2. Right side that serves decides the chess to be made to the board position to reflect the move.

To describe these rules it is convenient to introduce a notation for pattern and substitutions

E.g.:

1. `White pawn at square (file1,rank2)`
   `Move pawn from square (file i, rank2) AND square (file i, rank2)`
   `AND`

```
      Square (file i,rank3) is empty → To square (file i,rank4)
  AND
      Square (file i,rank4) is empty
```

2. White knight at square (file i,rank1)
   move Square(1,1) to → Square(i-1,3)
   AND
   Empty Square(i-1,3)
3. White knight at square (1,1)
   move Square(1,1) to → Square(i-1,3)
   AND
   Empty Square(i-1,3)

8-Puzzle Problem

The Problem is 8-Puzzle is a square tray in which 8 square tiles are placed. The remaining 9th square is uncovered. Each tile has a number on it. A file that is adjacent to the blank space can be slide into that space. The goal is to transform the starting position into the goal position by sliding the tiles around.

Solution:

State Space: The state space for the problem can be written as a set of states where each state is position of the tiles on the tray.

Initial State: Square tray having 3x3 cells and 8 tiles number on it that are shuffled

2	8	3
1	6	4
7		5

Goal State

1	2	3
8		4
7	6	5

Production Rules: These rules are used to move from initial state to goal state. These are also defined as two parts left side pattern should match with current position and left side will be resulting position after applying the rule.

1. Tile in square (1,1)
   AND — *Move tile from square (1,1) to (2,1)*
   Empty square (2,1)

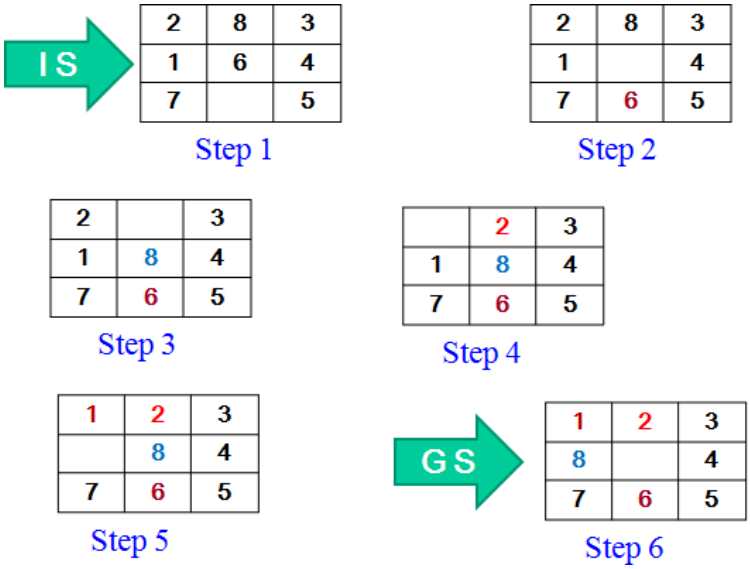
2. Tile in square (1,1)
   AND — *Move tile from square (1,1) to (1,2)*
   Empty square (1,2)

3. Tile in square (2,1)
   AND — *Move tile from square (2,1) to (1,1)*
   Empty square (1,1)

1,1 2	1,2 3	1,3 2
2,1 3	2,2 4	2,3 3
3,1 2	3,2 3	3,3 2

No. of Production Rules: *2 + 3 + 2 + 3 + 4 + 3 + 2 + 3 + 2 = 24*

Solution:



Travelling Salesman Problem

The Problem is the salesman has a list of cities, each of which he must visit exactly once. There are direct roads between each pair of cities on the list. Find the route the salesman should follow for the shortest possible round trip that both states and finishes at any one of the cities.

Solution:

State Space: The state space for this problem represents states in which the cities traversed by salesman and state described as salesman starting at any city in the given list of cities. A set of rules is applied such that the salesman will not traverse a city traversed once. These rules are

resulted to be states with the salesman will complex the round trip and return to his starting position.

Initial State

- Salesman starting at any arbitrary city in the given list of cities

Goal State

- Visiting all cities once and only and reaching his starting state

Production rules:

These rules are used as operators to move from one state to another. Since there is a path between any pair of cities in the city list, we write the production rules for this problem as

- `Visited(city[i]) AND Not Visited(city[j])`
  - `Traverse(city[i],city[j])`
- `Visited(city[i],city[j]) AND Not Visited(city[k])`
  - `Traverse(city[j],city[k])`
- `Visited(city[j],city[i]) AND Not Visited(city[k])`
  - `Traverse(city[i],city[k])`
- `Visited(city[i],city[j],city[k]) AND Not Visited(Nil)`
  - `Traverse(city[k],city[i])`

Towers of Hanoi Problem

Problem is the state space for the problem can be described as each state representing position of the disk on each pole the position can be treated as a stack the length of the stack will be equal to maximum number of disks each post can handle. The initial state of the problem will be any one of the posts will the certain the number of disks and the other two will be empty.

Initial State:

- Full(T1) | Empty(T2) | Empty(T3)

Goal State:

- Empty(T1) | Full(T2) | Empty (T3)

Production Rules:

These are rules used to reach the Goal State. These rules use the following operations:

- POP(x) → Remove top element x from the stack and update top
- PUSH(x,y) → Push an element x into the stack and update top. [Push an element x on to the y]

Now to solve the problem the production rules can be described as follows:

1. Top(T1)<Top(T2) → PUSH(POP(T1),T2)
2. Top(T2)<Top(T1) → PUSH(POP(T2),T1)
3. Top(T1)<Top(T3) → PUSH(POP(T1),T3)
4. Top(T3)<Top(T1) → PUSH(POP(T3),T1)
5. Top(T2)<Top(T3) → PUSH(POP(T2),T3)
6. Top(T3)<Top(T2) → PUSH(POP(T3),T2)
7. Empty(T1) → PUSH(POP(T2),T1)

8. Empty(T1) → PUSH(POP(T3),T1)
9. Empty(T2) → PUSH(POP(T1),T3)
10. Empty(T3) → PUSH(POP(T1),T3)
11. Empty(T2) → PUSH(POP(T3),T2)
12. Empty(T3) → PUSH(POP(T2),T3)

Solution: Example: 3 Disks, 3 Towers

1) T1 → T2
2) T1 → T3
3) T2 → T3
4) T1 → T2
5) T3 → T1
6) T3 → T2
7) T1 → T2

Monkey and Bananas Problem

Problem: A hungry monkey finds himself in a room in which a branch of bananas is hanging from the ceiling. The monkey unfortunately cannot reach the bananas however in the room there are also a chair and a stick. The ceiling is just right high so that a monkey standing on a chair could knock the bananas down with the stick. The monkey knows how to move round, carry other things around reach for the bananas and wave the stick in the air. What is the best sequence of actions for the monkey to acquire lunch?

Solution: The state space for this problem is a set of states representing the position of the monkey, position of chair, position of the stick and two flags whether monkey on the chair & whether monkey holds the stick so there is a 5-tuple representation.
(M, C, S, F1, F2)

- M: position of the monkey
- C: position of the chair
- S: position of the stick
- F1: 0 or 1 depends on the monkey on the chair or not
- F2: 0 or 1 depends on the monkey holding the stick or not

Initial State (M, C, S, 0, 0)

- The objects are at different places and obviously monkey is not on the chair and not holding the stick

Goal State (G, G, G, 1, 1)

- G is the position under bananas and all objects are under it, monkey is on the chair and holding stick

Production Rules:

These are the rules which have a path for searching the goal state here we assume that when monkey hold a stick then it will swing it this assumption is necessary to simplify the representation.
Some of the production rules are:

1) (M,C,S,0,0) → (A,C,S,0,0) {An arbitrary position A}
2) (M,C,S,0,0) → (C,C,S,0,0) {monkey moves to chair position}
3) (M,C,S,0,0) → (S,S,S,0,0) {monkey brings chair to stick position}
4) (C,C,S,0,0) → (A,A,S,0,0) {push the chair to arbitrary position A}
5) (S,C,S,0,0) → (A,C,A,0,1) {Taking the stick to arbitrary position}
6) (S,C,S,0,0) → (C,C,S,0,0) {monkey moves from stick position to chair position}
7) (C,C,C,0,1) → (C,C,C,1,1)
   - {monkey and stick at the chair position, monkey on the chair and holding stick}
8) (S,C,S,0,1) → (C,C,C,0,1)

Solution:

1) (M,C,S,0,0)
2) (C,C,S,0,0)
3) (G,G,S,0,0)
4) (S,G,S,0,0)
5) (G,G,G,0,0)
6) (G,G,G,0,1)
7) (G,G,G,1,1)

Missionaries and Cannibals Problem

Problem is 3 missionaries and 3 cannibals find themselves one side of the river. They have agreed that they would like to get the other side. But the missionaries are not sure what else the cannibals have agreed to. So the missionaries want to manage the trip across the river on either side of the river is never less than the number of cannibals who are on the same side. The only boat available holds only two people at a time. How can everyone get across without missionaries risking hang eager?

Solution:

The state space for the problem contains a set of states which represent the present number of cannibals and missionaries on the either side of the bank of the river.
(C,M,C1,M1,B)

- C and M are number of cannibals and missionaries on the starting bank
- C1 and M1 are number of cannibals and missionaries on the destination bank
- B is the position of the boat wither left bank (L) or right bank (R)

Initial State → C=3,M=3,B=L so (3,3,0,0,L)
Goal State → C1=3, M1=3, B=R so (0,0,3,3,R)

Production System: These are the operations used to move from one state to other state. Since at any bank the number of cannibals must less than or equal to missionaries we can write two production rules for this problem as follows:

- (C,M,C1,M1,L / C=3, M=3) → (C-2,M,C1+2,M1,R)
- (C,M,C1,M1,L / C=3, M=3) → (C-1,M-1,C1+1,M1+1,R)
- (C,M,C1,M1,L / C=3, M=3) → (C-1,M,C1+1,M1,R)
- (C,M,C1,M1,R / C=1, M=3) → (C+1,M,C1-1,M1,L)
- (C,M,C1,M1,R / C=0, M=3,C1=3,M1=0) → (C+1,M,C1-1,M1,L)

The solution path is

LEFT BANK			RIGHT BANK	
C	M	BOAT POSITION	C1	M1
3	3		0	0
1	3	→	2	0
2	3	←	1	0
0	3	→	3	0
1	3	←	2	0
1	1	→	2	2
2	2	←	1	1
2	0	→	1	3
3	0	←	0	3
1	0	→	2	3
2	0	←	1	3
0	0	→	3	3

3. Heuristic Search Techniques

Control Strategy

The question arises

"***How to decide which rule to apply next during the process of searching for a solution to a problem?***"

Requirements of a good search strategy:

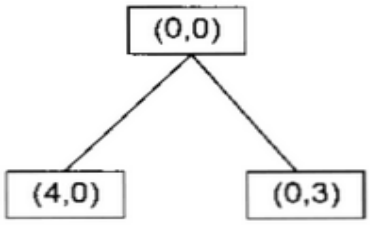
1. It causes motion. It must reduce the difference between current state and goal state. Otherwise, it will never lead to a solution.
2. It is systematic. Otherwise, it may use more steps than necessary.
3. It is efficient. Find a good, but not necessarily the best, answer.

Breadth First Search

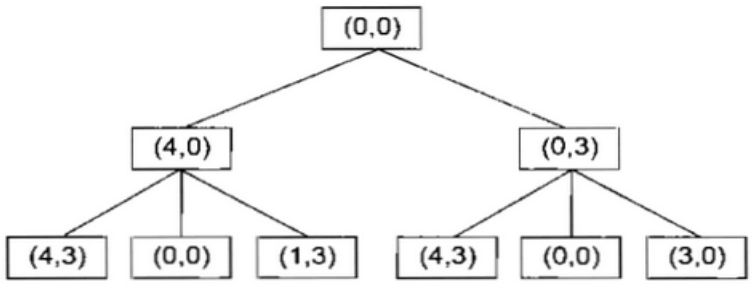
To solve the water jug problem systemically construct a tree with limited states as its root. Generate all the offspring and their successors from the root according to the rules until some rule produces a goal state. This process is called Breadth-First Search.

Algorithm:

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 do:
    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 goal state, quit and return this state
        iii. Otherwise add the new state to the end of NODE_LIST



One Level of a Breadth-First Search Tree



Two Levels of a Breadth-First Search Tree

The data structure used in this algorithm is QUEUE.

Explanation of Algorithm:

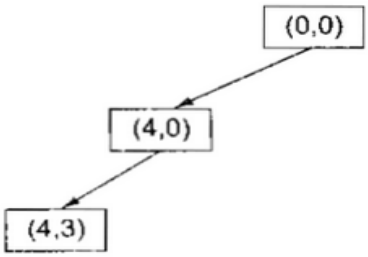
- Initially put (0,0) state in the queue
- Apply the production rules and generate new state
- If the new states are not the goal state, (not generated before and not expanded) then only add these states to queue.

Depth First Search

There is another way of dealing the Water Jug Problem. One should construct a single branched tree utility yields a solution or until a decision terminate when the path is reaching a dead end to the previous state. If the branch is larger than the pre-specified unit then backtracking occurs to the previous state so as to create another path. This is called Chronological Backtracking because the order in which steps are undone depends only on the temporal sequence in which the steps were originally made. This procedure is called Depth-First Search.

Algorithm:

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



A Depth-First Search Tree

The data structure used in this algorithm is STACK.
Explanation of Algorithm:

- Initially put the (0,0) state in the stack.
- Apply production rules and generate the new state.
- If the new states are not a goal state, (not generated before and no expanded) then only add the state to top of the Stack.
- If already generated state is encountered then POP the top of stack elements and search in another direction.

Advantages of Breadth-First Search

- BFS will not get trapped exploring a blind alley.
    - In case of DFS, it may follow a single path for a very long time until it has no successor.
- If there is a solution for particular problem, the BFS is generated to find it. We can find minimal path if there are multiple solutions for the problem.

Advantages of Depth –First Search

- DFS requires less memory since only the nodes on the current path are stored.
- Sometimes we may find the solution without examining much.

Example: Travelling Salesman Problem

To solve the TSM problem we should construct a tree which is simple, motion causing and systematic. It would explore all possible paths in the tree and return the one with the shortest length. If there are N cities, then the number of different paths among them is 1.2...(N-1) or (N-1)!

The time to examine a single path is proportional to N. So the total time required to perform this search is proportional to N!

Another strategy is, begin generating complete paths, keeping track of the shorter path so far and neglecting the paths where partial length is greater than the shortest found. This method is better than the first but it is inadequate.

To solve this efficiently we have a search called HEURISTIC SEARCH.

HEURISTIC SEARCH

Heuristic:

- It is a "rule of thumb" used to help guide search
- It is a technique that improves the efficiency of search process, possibly by sacrificing claims of completeness.
- It is involving or serving as an aid to learning, discovery, or problem-solving by experimental and especially trial-and-error methods.

Heuristic Function:

- It is a function applied to a state in a search space to indicate a likelihood of success if that state is selected
- It is a function that maps from problem state descriptions to measures of desirability usually represented by numbers
- Heuristic function is problem specific.

The purpose of heuristic function is to guide the search process in the most profitable direction by suggesting which path to follow first when more than one is available (best promising way).

We can find the TSM problem in less exponential items. On the average Heuristic improve the quality of the paths that are explored. Following procedure is to solve TRS problem

- Select a Arbitrary City as a starting city
- To select the next city, look at all cities not yet visited, and select one closest to the current city
- Repeat steps until all cities have been visited

HEURISTIC SEARCH TECHNIQUES:

Search Algorithms
Many traditional search algorithms are used in AI applications. For complex problems, thetraditional algorithms are unable to find the solutions within some practical time and spacelimits. Consequently, many special techniques are developed, using ***heuristic functions.***

The algorithms that use *heuristic functions* are called ***heuristic algorithms***.

• Heuristic algorithms are not really intelligent; they appear to be intelligent because theyachieve better performance.
• Heuristic algorithms are more efficient because they take advantage of feedback from the datato direct the search path.
• ***Uninformed search algorithms*** or *Brute-force algorithms*, search through the search space allpossible candidates for the solution checking whether each candidate satisfies the problem's statement.
• ***Informed search algorithms*** use heuristic functions that are specific to the problem, apply them to guide the search through the search space to try to reduce the amount of time spent insearching.

A good heuristic will make an informed search dramatically outperform any uninformed search:for example, the Traveling Salesman Problem (TSP), where the goal is to find is a good solutioninstead of finding the best solution.

In such problems, the search proceeds using current information about the problem to predict which path is closer to the goal and follow it, although it does not always guarantee to find thebest possible solution. Such techniques help in finding a solution within reasonable time and space (memory). Some prominent intelligent search algorithms are stated below:

1. ***Generate and Test Search***
2. ***Best-first Search***
3. ***Greedy Search***
4. ***A* Search***
5. ***Constraint Search***
6. ***Means-ends analysis***

There are some more algorithms. They are either improvements or combinations of these.
• **Hierarchical Representation of Search Algorithms:** A Hierarchical representation of mostsearch algorithms is illustrated below. The representation begins with two types of search:
• **Uninformed Search:** Also called blind, exhaustive or brute-force search, it uses no information about the problem to guide the search and therefore may not be very efficient.
• **Informed Search:** Also called heuristic or intelligent search, this uses information about theproblem to guide the search—usually guesses the distance to a goal state and is therefore efficient, but the search may not be always possible.

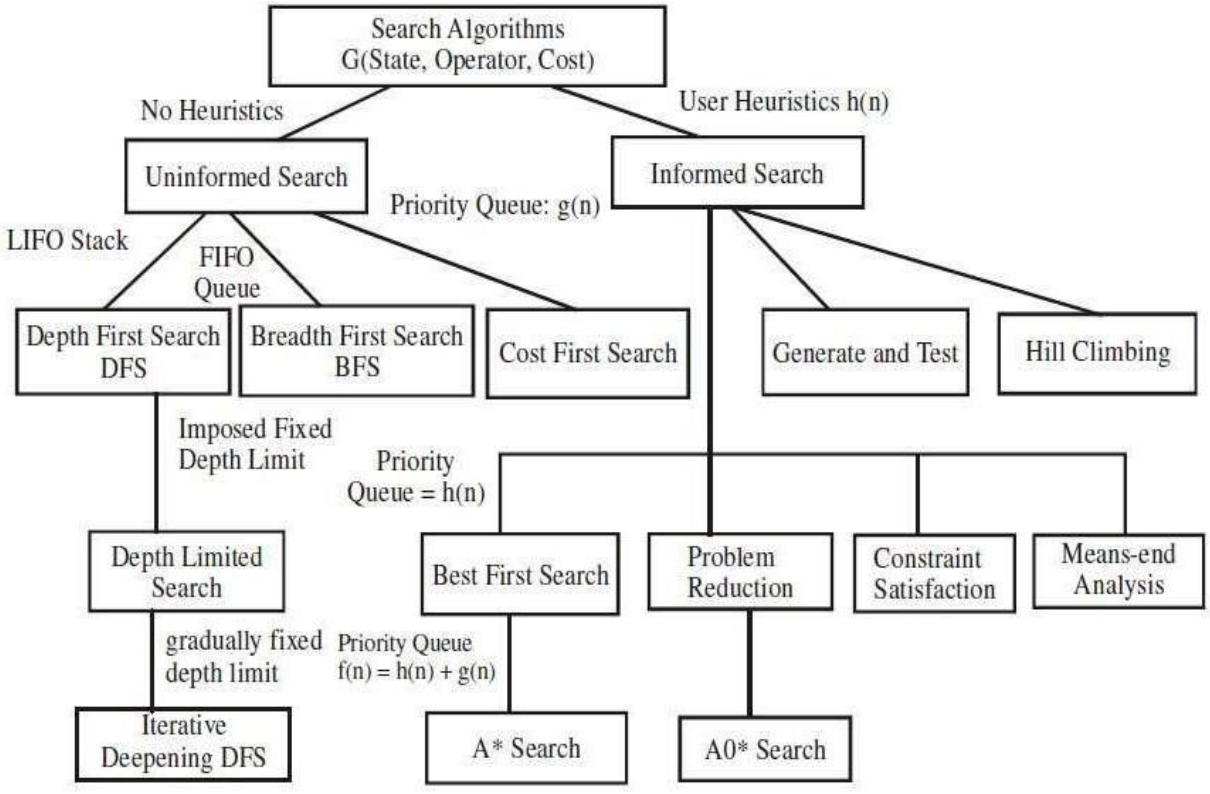
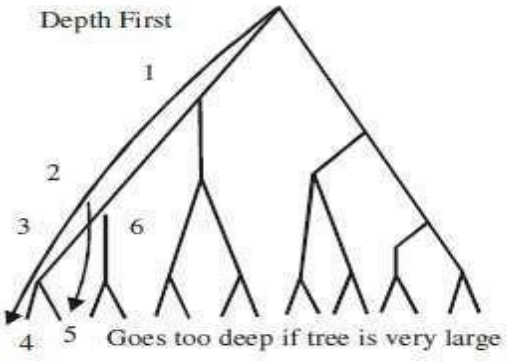
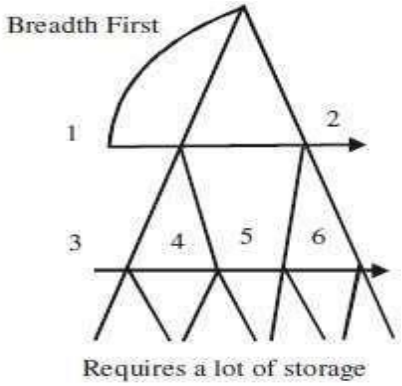


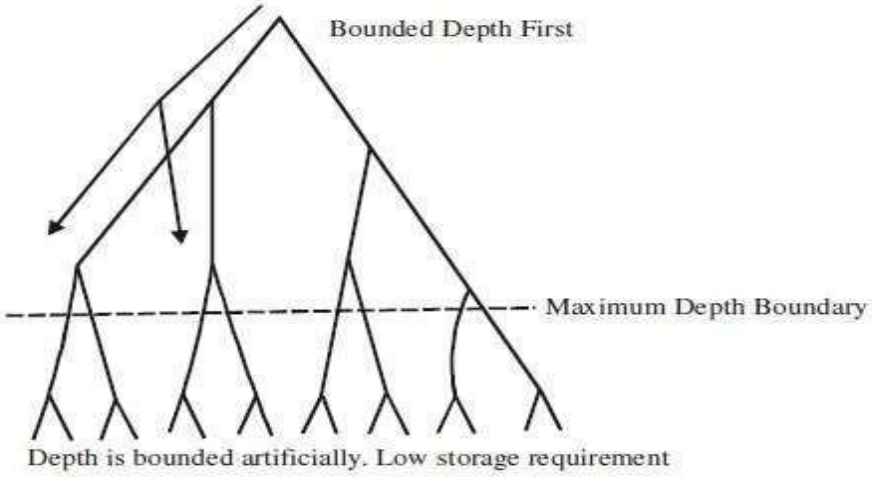
Fig. *Different Search Algorithms*

The first requirement is that it causes motion, in a game playing program, it moves on the boardand in the water jug problem, filling water is used to fill jugs. It means the control strategies without the motion will never lead to the solution.

The second requirement is that it is systematic, that is, it corresponds to the need for global motion as well as for local motion. This is a clear condition that neither would it be rational to fill a jug and empty it repeatedly, nor it would be worthwhile to move a piece round and round on the board in a cyclic way in a game. We shall initially consider two systematic approaches forsearching. Searches can be classified by the order in which operators are tried: depth-first, breadth-first, bounded depth-first.







Breadth-first search

A Search strategy, in which the highest layer of a decision tree is searched completely beforeproceeding to the next layer is called *Breadth-first search (BFS).*

• In this strategy, no viable solutions are omitted and therefore it is guaranteed that an optimalsolution is found.

• This strategy is often not feasible when the search space is large.

Algorithm

1. Create a variable called LIST and set it to be the starting state.
2. Loop until a goal state is found or LIST is empty, Do

a. Remove the first element from the LIST and call it E. If the LIST is empty, quit.

b. For every path each rule can match the state 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 LIST.

Advantages

1. Guaranteed to find an optimal solution (in terms of shortest number of stepsto reach the goal).
2. Can always find a goal node if one exists (complete).

Disadvantages

1. High storage requirement: *exponential* with tree depth.

Depth-first search

A search strategy that extends the current path as far as possible before backtracking to the lastchoice point and trying the next alternative path is called *Depth-first search (DFS).*

• This strategy does not guarantee that the optimal solution has been found.

• In this strategy, search reaches a satisfactory solution more rapidly than breadth first, anadvantage when the search space is large.

Algorithm

Depth-first search applies operators to each newly generated state, trying to drive directly towardthe goal.

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

a. Generate a successor E to the starting state. If there are no more successors, then signal failure.

b. Call Depth-first Search with E as the starting state.

c. If success is returned signal success; otherwise, continue in the loop.

Advantages

1. Low storage requirement: *linear* with tree depth.
2. Easily programmed: function call stack does most of the work of maintaining state of thesearch.

Disadvantages

1. May find a sub-optimal solution (one that is deeper or more costly than the best solution).
2. Incomplete: without a depth bound, may not find a solution even if one exists.

2.4.2.3 Bounded depth-first search

Depth-first search can spend much time (perhaps infinite time) exploring a very deep path that does not contain a solution, when a shallow solution exists. An easy way to solve this problem isto put a maximum depth bound on the search. Beyond the depth bound, a failure is generated automatically without exploring any deeper.

Problems:

1. It's hard to guess how deep the solution lies.
2. If the estimated depth is too deep (even by 1) the computer time used is dramaticallyincreased, by a factor of *bextra*.
3. If the estimated depth is too shallow, the search fails to find a solution; all that computer timeis wasted.

Heuristics

A heuristic is a method that improves the efficiency of the search process. These are like tour guides. There are good to the level that they may neglect the points in general interesting directions; they are bad to the level that they may neglect points of interest to particular individuals. Some heuristics help in the search process without sacrificing any claims to entiretythat the process might previously had. Others may occasionally cause an excellent path to be overlooked. By sacrificing entirety it increases efficiency. Heuristics may not find the best

solution every time but guarantee that they find a good solution in a reasonable time. These are particularly useful in solving tough and complex problems, solutions of which would require infinite time, i.e. far longer than a lifetime for the problems which are not solved in any other way.

Heuristic search

To find a solution in proper time rather than a complete solution in unlimited time we use heuristics. 'A heuristic function is a function that maps from problem state descriptions to measures of desirability, usually represented as numbers'. Heuristic search methods use knowledge about the problem domain and choose promising operators first. These heuristic search methods use heuristic functions to evaluate the next state towards the goal state. For finding a solution, by using the heuristic technique, one should carry out the following steps:

1. Add domain—specific information to select what is the best path to continue searching along.
2. Define a heuristic function h(n) that estimates the 'goodness' of a node n.

Specifically, h(n) = estimated cost(or distance) of minimal cost path from n to a goal state.

3. The term, heuristic means 'serving to aid discovery' and is an estimate, based on domain specific information that is computable from the current state description of how close we are toa goal.

Finding a route from one city to another city is an example of a search problem in whichdifferent search orders and the use of heuristic knowledge are easily understood.

1. State: The current city in which the traveller is located.
2. Operators: Roads linking the current city to other cities.
3. Cost Metric: The cost of taking a given road between cities.

4. Heuristic information: The search could be guided by the direction of the goal city from thecurrent city, or we could use airline distance as an estimate of the distance to the goal. **Heuristic search techniques**

For complex problems, the traditional algorithms, presented above, are unable to find the solution within some practical time and space limits. Consequently, many special techniques aredeveloped, using ***heuristic functions.***

• Blind search is not always possible, because it requires too much time or Space (memory).

Heuristics are ***rules of thumb***; they do not guarantee a solution to a problem.

• Heuristic Search is a weak technique but can be effective if applied correctly; it requiresdomain specific information.

Characteristics of heuristic search

• Heuristics are knowledge about domain, which help search and reasoning in its domain.
• Heuristic search incorporates domain knowledge to improve efficiency over blind search.
• Heuristic is a function that, when applied to a state, returns value as estimated merit of state,with respect to goal.

- ✓ Heuristics might (for reasons) *underestimate* or *overestimate* the merit of a state withrespect to goal.
- ✓ Heuristics that underestimate are desirable and called admissible.

• Heuristic evaluation function estimates likelihood of given state leading to goal state.
• Heuristic search function estimates cost from current state to goal, presuming function isefficient.

Heuristic search compared with other search
The Heuristic search is compared with Brute force or Blind search techniques below:

Comparison of Algorithms

Brute force / Blind search	**Heuristic search**
Can only search what it has knowledge	Estimates 'distance' to goal state about already through explored nodes
No knowledge about how far a node state	Guides search process toward goalnode from goal
	Prefers states (nodes) that lead close to and not away from goalstate

Example: Travelling salesman
A salesman has to visit a list of cities and he must visit each city only once. There are different routes between the cities. The problem is to find the shortest route between the cities so that thesalesman visits all the cities at once.

Suppose there are N cities, then a solution would be to take N! possible combinations to find theshortest distance to decide the required route. This is not efficient as with N=10 there are 36,28,800 possible routes. This is an example of *combinatorial explosion*.

There are better methods for the solution of such problems: one is called *branch* and *bound*. First, generate all the complete paths and find the distance of the first complete path. If the nextpath is shorter, then save it and proceed this way avoiding the path when its length exceeds the saved shortest path length, although it is better than the previous method.

Generate and Test Strategy

Generate-And-Test Algorithm
Generate-and-test search algorithm is a very simple algorithm that guarantees to find a solution ifdone 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 someproblems the possible solutions may be particular points in the problem space and for some problems, paths from the start state.

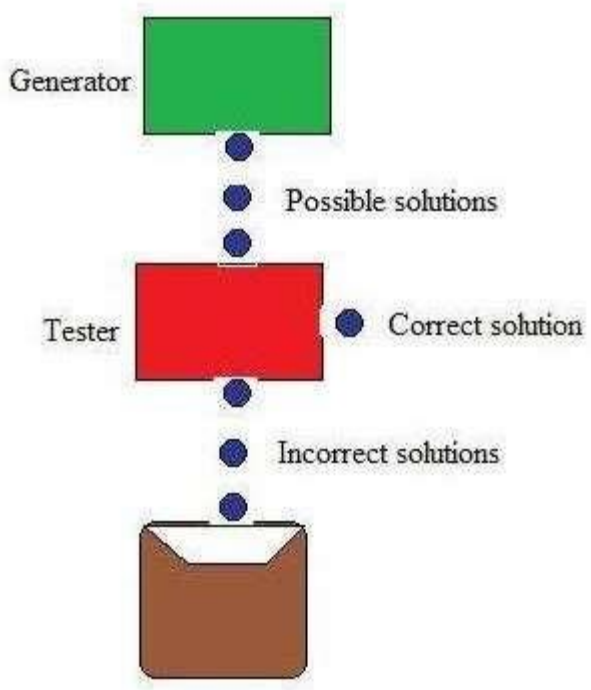


Figure: Generate And Test

Generate-and-test, like depth-first search, requires that complete solutions be generated fortesting. 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 isknown as British Museum algorithm: finding an object in the British Museum by wandering randomly.

Systematic Generate-And-Test

While generating complete solutions and generating random solutions are the two extremes thereexists 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.

Generate-And-Test And Planning

Exhaustive generate-and-test is very useful for simple problems. But for complex problems evenheuristic 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. AnAI 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 ofArtificial Intelligence .

Given a large set of inputs and a good heuristic function, it tries to find a sufficiently goodsolution 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 weneed to minimize the distance traveled by salesman.
- 'Heuristic search' means that this search algorithm may not find the optimal solution tothe 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 thealgorithm 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. Thegenerate 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 insearch space.

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

Types of Hill Climbing

1. Simple Hill climbing : 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 beapplied to current state.
a) Select a state that has not been yet applied to the current state and apply it to produce a newstate.
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.

2. Steepest-Ascent Hill climbing : It first examines all the neighboring nodes and thenselects 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 initialstate
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 TargetStep 3 : Exit

3. Stochastic hill climbing : 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 onthe amount of improvement in that neighbor) whether to move to that neighbor or to examine another.

State Space diagram for Hill Climbing
State space diagram is a graphical representation of the set of states our search algorithm canreach vs the value of our objective function(the function which we wish to maximize).
X- axis : denotes the state space ie states or configuration our algorithm may reach.

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 herevalue of objective function is higher than its neighbors.

2. Global maximum : It is the best possible state in the state space diagram. This because atthis state, objective function has highest value.
3. Plateua/flat local maximum : It is a flat region of state space where neighboring stateshave the same value.
4. Ridge : It is region which is higher than its neighbours but itself has a slope. It is a specialkind of local maximum.
5. Current state : The region of state space diagram where we are currently present duringthe search.
6. Shoulder : It is a plateau that has an uphill edge.Problems in different regions in Hill climbing

Hill climbing cannot reach the optimal/best state(global maximum) if it enters any of thefollowing regions :

1. Local maximum : At a local maximum all neighboring states have a values which is worse than than the current state. Since hill climbing uses greedy approach, it will not move to the worse state and terminate itself. The process will end even though a bettersolution may exist.
   To overcome local maximum problem : Utilize backtracking technique. Maintain a list ofvisited states. If the search reaches an undesirable state, it can backtrack to the previous configuration and explore a new path.
2. Plateau : On plateau all neighbors have same value . Hence, it is not possible to select thebest direction.

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

3. Ridge : Any point on a ridge can look like peak because movement in all possibledirections is downward. Hence the algorithm stops when it reaches this state.

To overcome Ridge : In this kind of obstacle, use two or more rules before testing. Itimplies moving in several directions at once.

Best First Search (Informed Search)

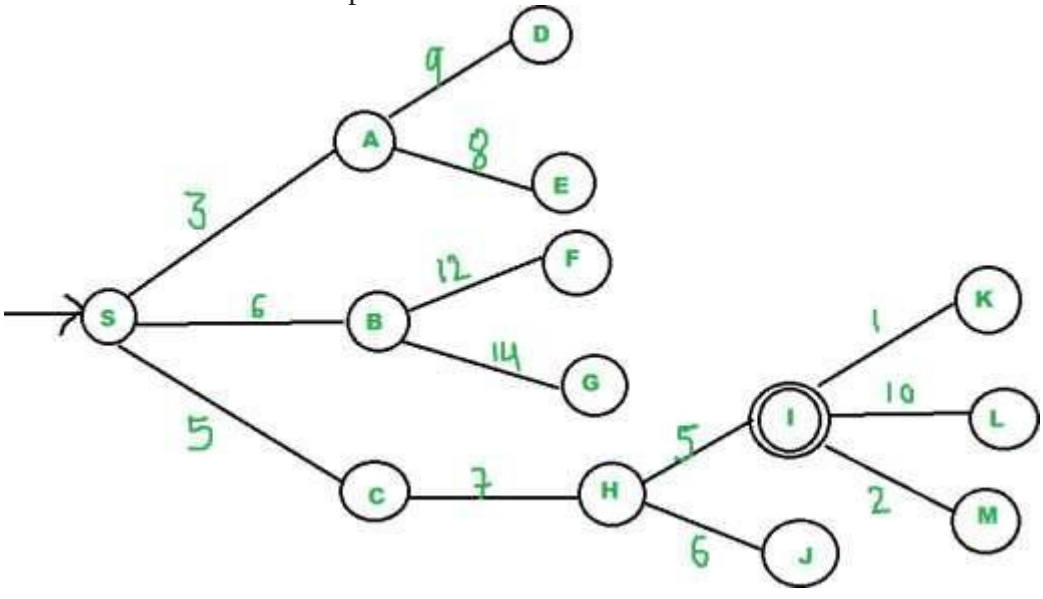
In BFS and DFS, when we are at a node, we can consider any of the adjacent as next node. So both BFS and DFS blindly explore paths without considering any cost function. Theidea of Best First Search is to use an evaluation function to decide which adjacent is most promising and then explore. Best First 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, wejust need to change Queue to PriorityQueue.

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

1) Create an empty PriorityQueuePriorityQueue pq;
2) Insert "start" in pq.pq.insert(start)
3) Until PriorityQueue is empty u = PriorityQueue.DeleteMin

If u is the goalExit
Else
Foreach neighbor v of uIf 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 forgoal "I" using given costs and Best First search.

pq initially contains S
We remove s from and process unvisitedneighbors of S to pq.
pq now contains {A, C, B} (C is putbefore B because C has lesser cost)

We remove A from pq and process unvisitedneighbors of A to pq.
pq now contains {C, B, E, D}

We remove C from pq and process unvisitedneighbors of C to pq.
pq now contains {B, H, E, D}

We remove B from pq and process unvisitedneighbors 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 numberof 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 isdesigned.

A* Search Algorithm

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 theworld 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.

In this tutorial I will look at the use of state space search to find the shortest path between twopoints (pathfinding), and also to solve a simple sliding tile puzzle (the 8-puzzle). Let's look atsome 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*. In the 8-puzzle diagram the edges are shown as blue lines. See figure 1 below. *State space search*, then, is solving a problem by beginning with the start state, and then for eachnode 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 witha 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

multiplying two numbers together on paper. A heuristic is not guaranteed to work but is useful inthat 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 ourheuristic 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 that I know of is known as the Nilsson score which leads fairly directly tothe 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 wherewe 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 wewant 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.

8 Puzzle
Let's look at the 8 puzzle in more detail. This is a simple sliding tile puzzle on a 3*3 grid whereone tile is missing and you can move the other tiles into the gap until you get the puzzle into thegoal position. See figure 1.

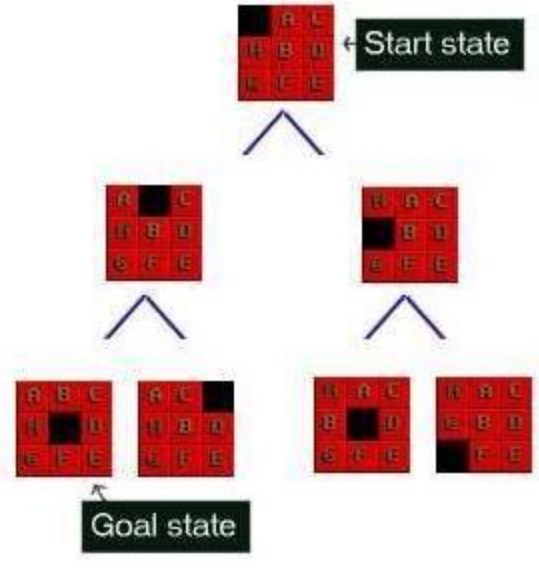


Figure 1 : The 8-Puzzle state space for a very simple example

There are 362,880 different states that the puzzle can be in, and to find a solution the search hasto find a route through them. From most positions of the search the number of edges (that's the

blue lines) is two. That means that the number of nodes you have in each level of the search is2^d where d is the depth. If the number of steps to solve a particular state is 18, then that's 262,144 nodes just at that level.

The 8 puzzle game state is as simple as representing a list of the 9 squares and what's in them. Here are two states for example; the last one is the GOAL state, at which point we've found thesolution. The first is a jumbled up example that you may start from.

Start state SPACE, A, C, H, B, D, G, F, EGoal state A, B, C, H, SPACE, D, G, F, E

The rules that you can apply to the puzzle are also simple. If there is a blank tile above, below, tothe left or to the right of a given tile, then you can move that tile into the space. To solve the puzzle you need to find the path from the start state, through the graph down to the goal state.

There is example code to to solve the 8-puzzle on the github site.

Pathfinding

In a video game, or some other pathfinding scenario, you want to search a state space and find out how to get from somewhere you are to somewhere you want to be, without bumping into walls or going too far. For reasons we will see later, the A* algorithm will not only find a path, ifthere is one, but it will find the shortest path. A state in pathfinding is simply a position in the world. In the example of a maze game like Pacman you can represent where everything is using a simple 2d grid. The start state for a ghost say, would be the 2d coordinate of where the ghost isat the start of the search. The goal state would be where pacman is so we can go and eat him.

There is also example code to do pathfinding on the github site.

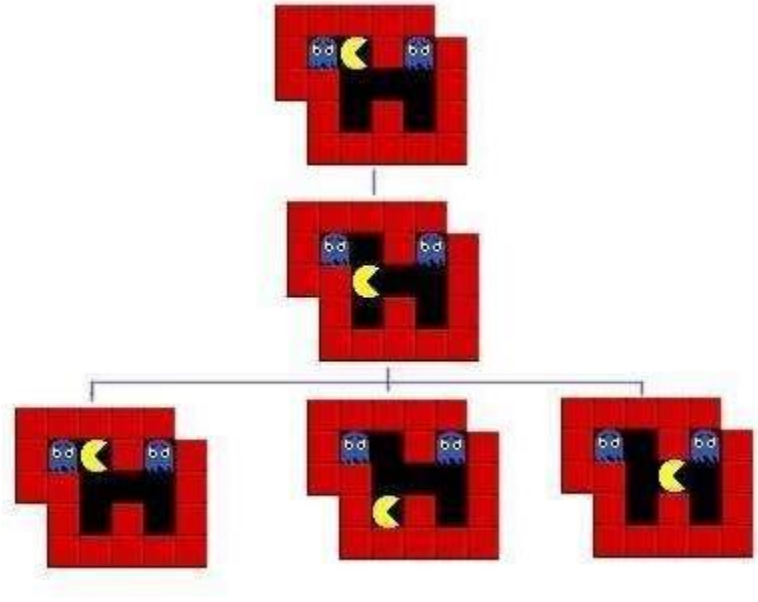


Figure 2 : The first three steps of a pathfinding state space

Implementing A*

We are now ready to look at the operation of the A* algorithm. What we need to do is start withthe goal state and then generate the graph downwards from there. Let's take the 8-puzzle in figure 1. We ask how many moves can we make from the start state? The answer is 2, there are two directions we can move the blank tile, and so our graph expands.

If we were just to continue blindly generating successors to each node, we could potentially fill the computer's memory before we found the goal node. Obviously we need to remember the bestnodes and search those first. We also need to remember the nodes that we have expanded already, so that we don't expand the same state repeatedly.

Let's start with the OPEN list. This is where we will remember which nodes we haven't yet expanded. When the algorithm begins the start state is placed on the open list, it is the only statewe know about and we have not expanded it. So we will expand the nodes from the start and putthose on the OPEN list too. Now we are done with the start node and we will put that on the CLOSED list. The CLOSED list is a list of nodes that we have expanded.

$f = g + h$

Using the OPEN and CLOSED list lets us be more selective about what we look at next in the search. We want to look at the best nodes first. We will give each node a score on how good wethink it is. This score should be thought of as the cost of getting from the node to the goal plus the cost of getting to where we are. Traditionally this has been represented by the letters f, g and
h. 'g' is the sum of all the costs it took to get here, 'h' is our heuristic function, the estimate of what it will take to get to the goal. 'f' is the sum of these two. We will store each of these in ournodes.

Using the f, g and h values the A* algorithm will be directed, subject to conditions we will lookat further on, towards the goal and will find it in the shortest route possible.

So far we have looked at the components of the A*, let's see how they all fit together to make the algorithm :

Pseudocode

Hopefully the ideas we looked at in the preceding paragraphs will now click into place as welook at the A* algorithm pseudocode. You may find it helpful to print this out or leave the window open while we discuss it.

To help make the operation of the algorithm clear we will look again at the 8-puzzle problem infigure 1 above. Figure 3 below shows the f,g and h scores for each of the tiles.

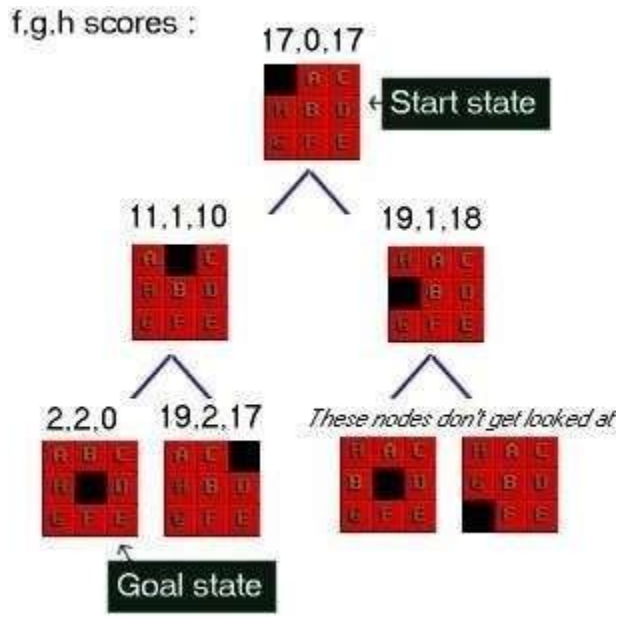


Figure 3 : 8-Puzzle state space showing f,g,h scores

First of all look at the g score for each node. This is the cost of what it took to get from the startto that node. So in the picture the center number is g. As you can see it increases by one at eachlevel. In some problems the cost may vary for different state changes. For example in pathfinding there is sometimes a type of terrain that costs more than other types.
Next look at the last number in each triple. This is h, the heuristic score. As I mentioned above I am using a heuristic known as Nilsson's Sequence, which converges quickly to a correct solutionin many cases. Here is how you calculate this score for a given 8-puzzle state :

Advantages:

It is complete and optimal.

It is the best one from other techniques. It is used to solve very complex problems.

It is optimally efficient, i.e. there is no other optimal algorithm guaranteed to expand fewer nodesthan A*.

Disadvantages:

This algorithm is complete if the branching factor is finite and every action has fixed cost.

The speed execution of A* search is highly dependant on the accuracy of the heuristic algorithmthat is used to compute h (n).

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.

6 7: The distance from current node to goal node.

ALGORITH
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.

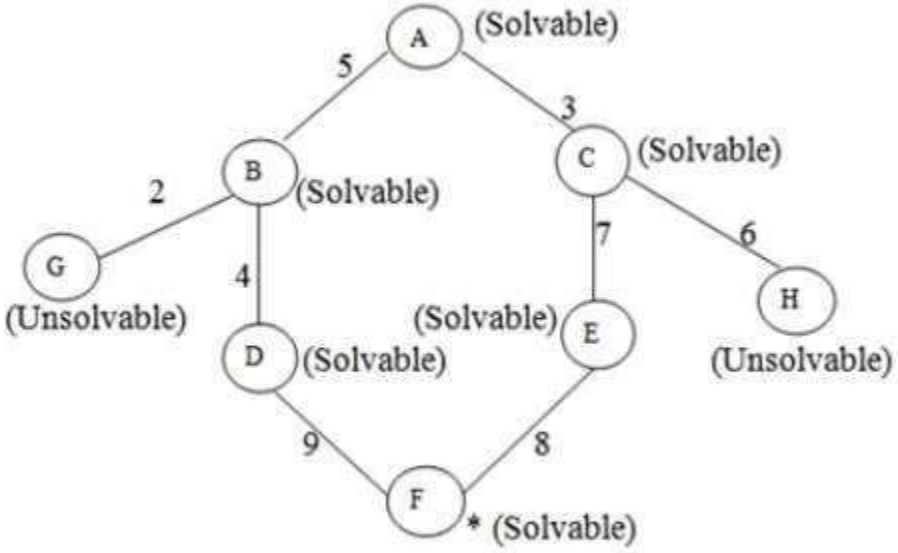
Step 6: Expand n. Find all its successors and find their h (n) value, push them into OPEN.

Step 7: Return to Step 2.

Step 8: Exit.

Implementation:

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



Figure

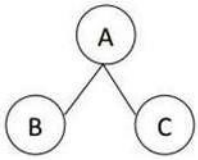
Step 1:

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.

Step 2:

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

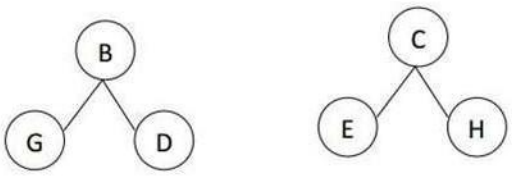
i.e. OPEN = | B | C | CLOSE = | A |



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 | | CLOSE = | A | B | C |



(O)

'O' indicated that the nodes G and H are unsolvable.

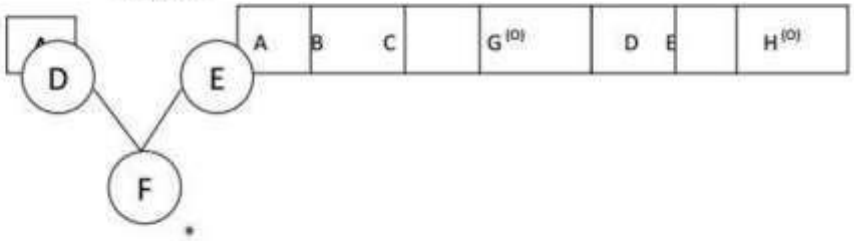
i.e. C

Step 4:

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

Succ

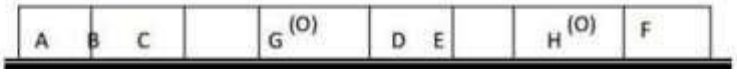
i.e. OPEN = CLOSE =



AO*

Step 5:

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



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 otheralgorithms.

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 - ORtrees 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 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.

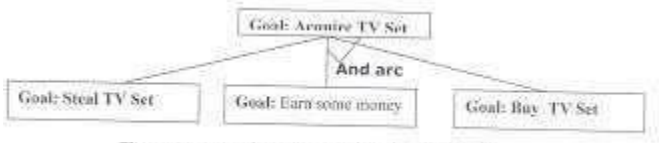


Figure shows AND - Or graph - an example.

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

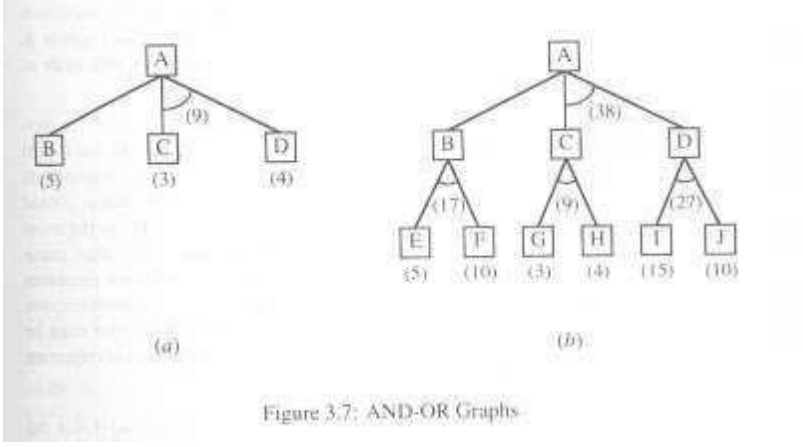


Figure 3.7: AND-OR Graphs

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).

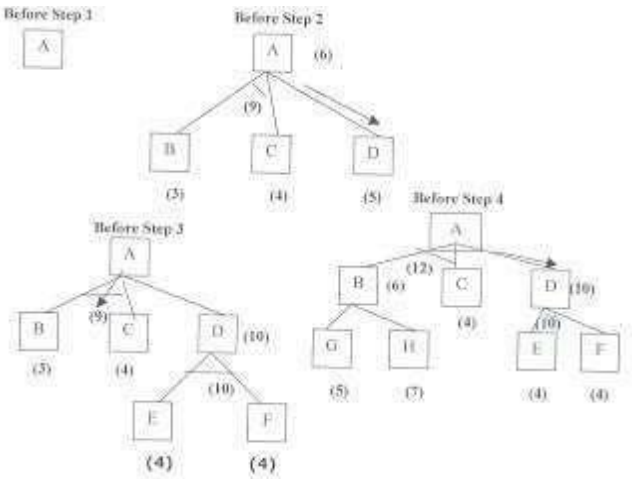
Thus the choice of the next node to expand depends not only n a value but also on whether that node is part of the current best path form the initial mode. Figure (b) makes this clearer. In figurethe node G appears to be the most promising node, with the least f ' value. But G is not on the current beat 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
o
n called SUCCESSOR, that is not also an ancester of NODE do the following
e

(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
m
u be propagate backwards up the graph . hence all the ancestors of CURRENT are addedto S.
s
t

(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 newlygenerated 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:

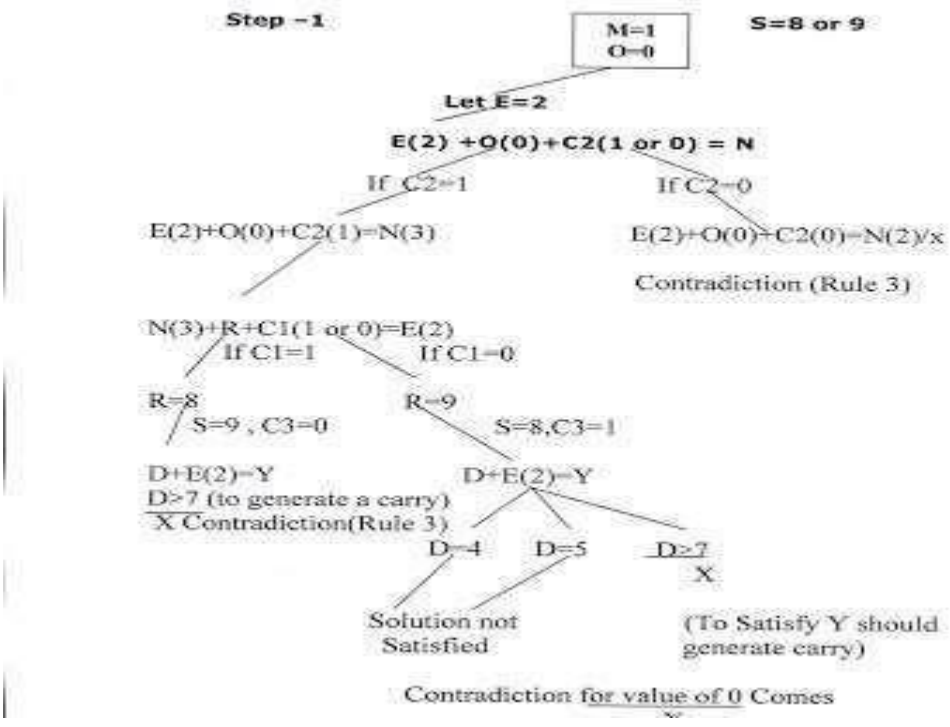
We are following the depth-first method to solve the problem.

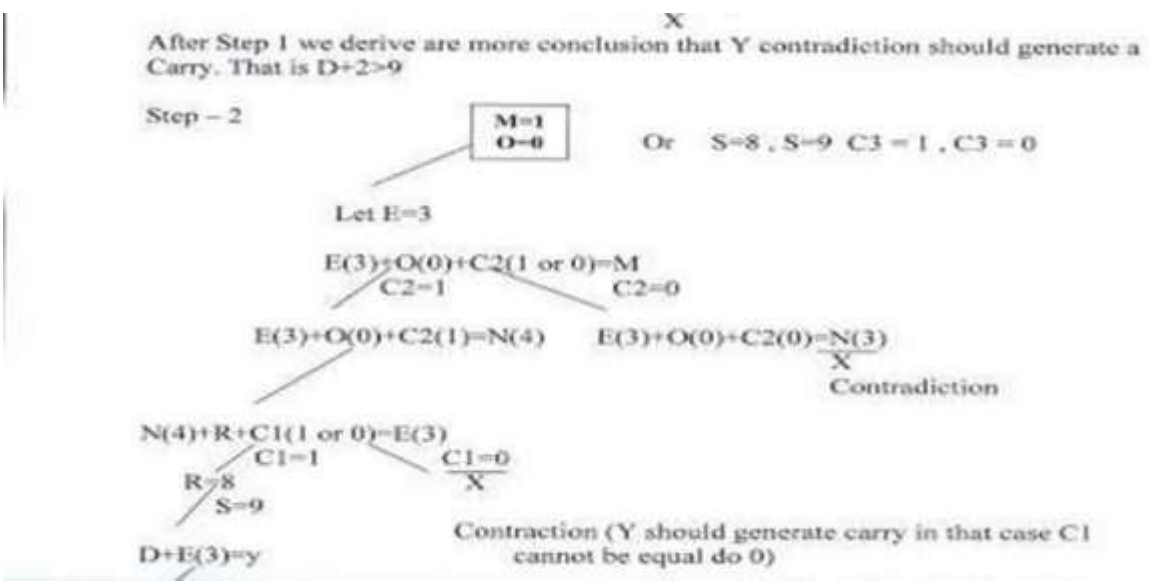
1. initial guess m=1 because the sum of two single digits can generate at most a carry '1'.

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.

nstraints. And
umed that the
, using depth-

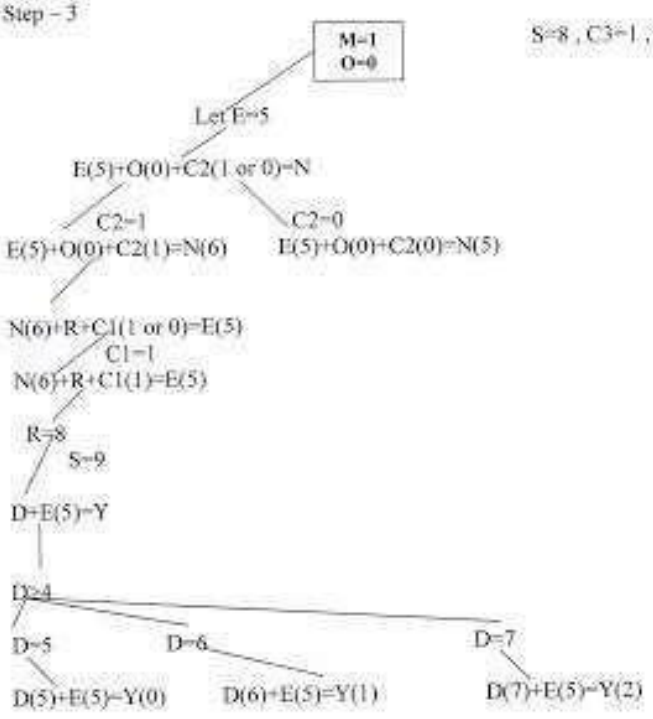




D>6(Controduction)

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)