MODULE 1

****

SYLLABUS

Module 1 (14 hours)

Introduction: What is AI, The foundations of AI, History and applications, Production systems. Structures and strategies for state space search. Informed and Uninformed searches.

****

Introduction

Artificial intellegence(AI) is the study of how to make computer do things which at the moment people do better.

The AI problems

Some of the task domains of AI are

- Mundane Tasks
  - Perception
    - Vision
    - Speech
  - Natural Language
    - Understanding
    - Generation
    - Translation
  - Commonsense reasoning
  - Robot control
- Formal Taks
  - Games
    - Chess
    - Backgammon
    - Checkers
  - Mathematics
    - Geometry
    - Logic
    - Integral calculus
    - Proving properties of programs
  - Expert Tasks
    - Engineering
      - Design

- Fault Finding
- Manufacturing planning

- Scientific Analysis
- Medical Diagnosis
- Financial Analysis

Perception of the world around us is crucial to our survival. Animals with much less intelligence than people are capable of more sophisticated visual perception than are current machines. Perpetual difficult because they involve analog signals, the signals are typically very noisy and usually a large number of things must be perceived at once.

The ability to use language to communicate a wide variety of ideas is perhaps the most important thing that seperates humans from the other animals. The problem of understanding spoken language is a perceptual problem. This problem, usually referred to as natural language understanding. In order to understand sentences about topic, it is necessary to know not only the language itself but also a good deal about the topic so that unstated assumptions can be recognized.

AI focused on the sort of problem solving that we do every day when we decide how to get to work in the morning, often called commonsense reasoning.

Game Playing and theorm proving show the property that people who do them well are considered to be displaying intelligence. Compters could perform well at those tasks simply by being fast at exploring a large number of solution paths and then selecting the best one.

Now thousands of programs called expert systems in day to day operation throughout all areas of industry and goverment. Each of these system attempts to solve part of a practical, significant problem that previously required scarce expertise.

AI is a system that acts like human beings

For this, a computer would need to possess the following capabilities.

- **Natural language processing** To enable it to communicate successfully in English.
- **Knowledge representation** To store what it knows or hears.
- **Automated reasoning** To use the stored information to answer questions and to draw new conclusions.
- **Machine learning** To adapt to new circumstances and to detect and extrapolate patterns.
- **Computer vision** To perceive objects.
- **Robotics** To manipulate objects and move about.

AI is a system that thinks like human beings.

First we must have some way of determining how humans think. We need to get inside the workings of the human minds. Once we have a sufficiently precise theory of the mind, it becomes possible to express that theory using a computer program.

The field of cognitive science brings together computer models from AI and experimental techniques from psychology to try to construct precise and testable theories of the workings of the human mind.

AI is a system that thinks rationally

For a given set of correct premises, it is possible to yield new conclusions.

For eg.

"Socrates is a man; all men are mortal; therefore, Socrates is mortal."

These laws of thought were supposed to govern the operation of the mind. This resulted in a field called logic. A precise notation for the statements about all kinds of things in the world and about relations among them are developed. Programs exist that could in principle solve any solvable problem described in logical notation.

There are 2 main obstacles to this approach.

- First it is not easy to take informal knowledge and state it in the formal terms required by logical notation.
- Second, there is a big difference between being able to solve a problem "in principle" and doing so in practice.

AI is a system that acts rationally

An agent is something that acts. A rational agent is one that acts so as to achieve the best outcome or, when there is uncertainty, the best expected outcome. We need the ability to represent knowledge and reason with it because this enables us to reach good decisions in a wide variety of situations. We need to be able to generate comprehensive sentences in natural language because saying those sentences helps us get by in a complex society. We need learning because having a better idea of how the world works enables us to generate more effective strategies for dealing with it. We need visual perception to get a better idea of what an action might achieve.

AI Application Areas

The 2 most fundamental concerns of AI researchers are knowledge representation and search.

- **Knowledge representation**

  It addresses the problem of capturing the full range of knowledge required for intelligent behavior in a formal language,

  i.e. One suitable for computer manipulation.

  Eg. predicate calculus, LISP, Prolog

- **Search**

  It is a problem solving technique that systematically explores a space of problem states, ie, successive and alternative stages in the problem solving process.

The following explains the major application areas of AI.

1. Game Playing

Much of the early research in AI was done using common board games such as

- checkers,
- chess, and
- 15 puzzle.

Board games have certain properties that made them ideal for AI research. Most games are played using a well defined set of rules. This makes it easy to generate the search space. The board configuration used in playing these games can be easily represented on a computer. As games can be easily played, testing a game playing program presents no financial or ethical burden.

2. Heuristics

Games can generate extremely large search spaces. So we use powerful techniques called heuristics to explore the problem space.

A heuristic is a useful but potentially fallible problem strategy, such as checking to make sure that an unresponsive appliance is plugged in before assuming that it is broken. Since most of us have some experience with these simple games, we do not need to find and consult an expert. For these reasons games provide a rich domain for the study of heuristic search.

3. Automated Reasoning and Theorem Proving

Examples for automatic theorem provers are

Newell and Simon's Logic Theorist,

General Problem Solver (GPS).

Theorem proving research is responsible for the development of languages such as predicate calculus and prolog. The attraction of automated theorem proving lies in the rigor and generality of logic. A wide variety of problems can be attacked by representing the problem description as logical axioms and treating problem instances as theorems to be proved. Reasoning based on formal mathematical logic is also attractive. Many important problems such as design and verification of logic circuits, verification of the correctness of computer programs and control of complex systems come in this category.

4. Expert Systems

Here comes the importance of domain specific knowledge. A doctor, for example, is effective at diagnosing illness because she possesses some innate general problem solving skill; she is effective because she knows a lot about medicine.

A geologist is effective at discovering mineral deposits. Expert knowledge is a combination of theoretical understanding of the problem and a collection of heuristic problem solving rules that experience has shown to be effective in the domain. Expert systems are constructed by obtaining this knowledge from a human expert and coding it into a form that a computer may apply to similar problems. To develop such a system, we must obtain knowledge from a human domain expert. Examples for domain experts are doctor, chemist, geologist, engineer etc..

The domain expert provides the necessary knowledge of the problem domain. The AI specialist is responsible for implementing this knowledge in a program.

Once such a program has been written, it is necessary to refine its expertise through a process of giving it example problems to solve and making any required changes or modifications to the program's knowledge.

Dendral is an expert system designed to infer the structure of organic molecules from their chemical formulas and

mass spectrographic information about the chemical bonds present in the molecules.

Mycin is an expert system which uses expert medical knowledge to diagnose and prescribe treatment for spinal meningitis

and bacterial infections of the blood.

Prospector is an expert system for determining the probable location and type of ore deposits based on geological

information about a site.

Internist is an expert system for performing diagnosis in the area of internal medicine.

The dipmeter advisor is an expert system for interpreting the results of oil well drilling logs.

Xcon is an expert system for configuring VAX computers.

5. Natural Language Understanding and Semantic Modeling

One goal of AI is the creation of programs that are capable of understanding and generating human language. Systems that can use natural language with the flexibility and generality that characterize human speech are beyond current methodologies. Understanding natural language involves much more than parsing sentences into their individual parts of speech and looking those words up in a dictionary. Real understanding depends on extensive background knowledge.

Consider for example, the difficulties in carrying out a conversation about baseball with an individual who understands English but knows nothing about the rules of the game. This person will not be able to understand the meaning of the sentence.

"With none down in the top of the ninth and the go ahead run at second, the manager called his relief from the bull pen".

Even though hall of the words in the sentence may be individually understood, this sentence would be difficult to even the most intelligent non base ball fan. The task of collecting and organizing this background knowledge in such a way that it may be applied to language comprehension forms the major problem in automating natural language understanding.

6. Modeling Human Performance

We saw that human intelligence is a reference point in considering artificial intelligence. It does not mean that programs should pattern themselves after the organization of the human mind. Programs that take non human approaches to solving problems are often more successful than their human counterparts. Still,

the design of systems that explicitly model some aspect of human performance has been a fertile area of research in both AI and psychology.

7. Planning and Robotics

Research in planning began as an effort to design robots that could perform their tasks with some degree of flexibility and responsiveness to outside world. Planning assumes a robot that is capable of performing certain atomic actions.

Planning is a difficult problem because of the size of the space of possible sequences of moves. Even an extremely simple robot is capable of generating a vast number of potential move sequences. One method that human beings use in planning is hierarchical problem decomposition. If we plan a trip to London, we will generally treat the problems of arranging a flight, getting to the air port, making airline connections and finding ground transportation in London separately. Each of these may be further decomposed into smaller sub problems. Creating a computer program that can do the same is a difficult challenge.

A robot that blindly performs a sequence of actions without responding to changes in its environment cannot be considered intelligent. Often, a robot will have to formulate a plan based on the incomplete information and correct its behavior. A robot may not have adequate sensors to locate all obstacles in the way of a projected path. Organizing plans in a fashion that allows response to changing environmental conditions is a major problem for planning.

8. Machine Learning

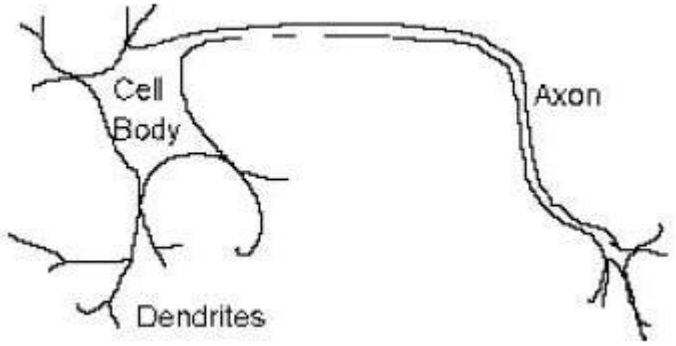
An expert system may perform extensive and costly computations to solve a problem. But if it is given the same or similar problem a second time, it usually does not remember the solution. It performs the same sequence of computations again.

This is not the behavior of an intelligent problem solver. The programs must learn on their own. Learning is a difficult area. But there are several programs that suggest that it is possible. One program is AM, the automated mathematician which was designed to discover mathematical laws. Initially given the concepts and axioms of set theory, AM was able to induce important mathematical concepts such as cardinality, integer arithmetic and many of the results of number theory. AM conjectured new theorems by modifying its current knowledge base.

Early work includes Winston's research on the induction of structural concepts such as "arch" from a set of examples in the blocks world. The ID3 algorithm has proved successful in learning general patterns from examples. Meta dendral learns rules for interpreting mass spectrographic data in organic chemistry from examples of data on compounds of known structure.Teiresias, an intelligent front end for expert systems, converts high level advice into new rules for its knowledge base. There are also now many important biological and sociological models of learning.

9. Neural Nets and Genetic Algorithms

An approach to build intelligent programs is to use models that parallel the structure of neurons in the human brain. A neuron consists of a cell body that has a number of branched protrusions called dendrites and a single branch called the axon. Dendrites receive signals from other neurons. When these combined impulses exceed a certain threshold, the neuron fires and an impulse or spike passes down the axon.



This description of the neuron captures features that are relevant to neural models of computation. Each computational unit computes some function of its inputs and passes the result along to connected units in the network; the final results are produced by the parallel and distributed processing of this network of neural connection and threshold weights.

Languages and Environments for AI

Programming environments include knowledge structuring techniques such as object oriented programming and expert systems frameworks.

High level languages such as Lisp, and Prolog support modular development.

Problems, Problem Space And Search

To build a system to solve a particular problem

1. Define the problem precisely: this definition must include precise specification of what the initial situation will be as well as what final situation constitute acceptable solution to the problem

2. Analyse the problem: a few very important feature can have an immense impact on appropriateness of various possible technique for solving the problem

3. Isolate and represent the task knowledge: that is necessary to solve the problem

4. Choose the best problem solving technique and apply it to particular problem.

Defining problem as a state space search

Suppose we start with the problem statement “PLAY CHESS”

To build a program

- Specify the starting position of the chess board
- The rules that define the legal moves
- The board positions that represent a win for one side or other

In addition we must make explicit the implicit goal of not only playing a legal game of chess but also winning the game if possible.

The starting position can be described as 8X8 array where each position contains a symbol standing for the appropriate piece in the official chess opening position. We can define as our goal any board position in which the opponent does not have any legal move and his or her king is under attack. The legal moves provide the way of getting from the initial state to a goal state.

Set of rules consisting of two parts:

- A left side that serves as a pattern to be matched against the current board position
- A right side that describe the change to be made to the board position to reflect the move

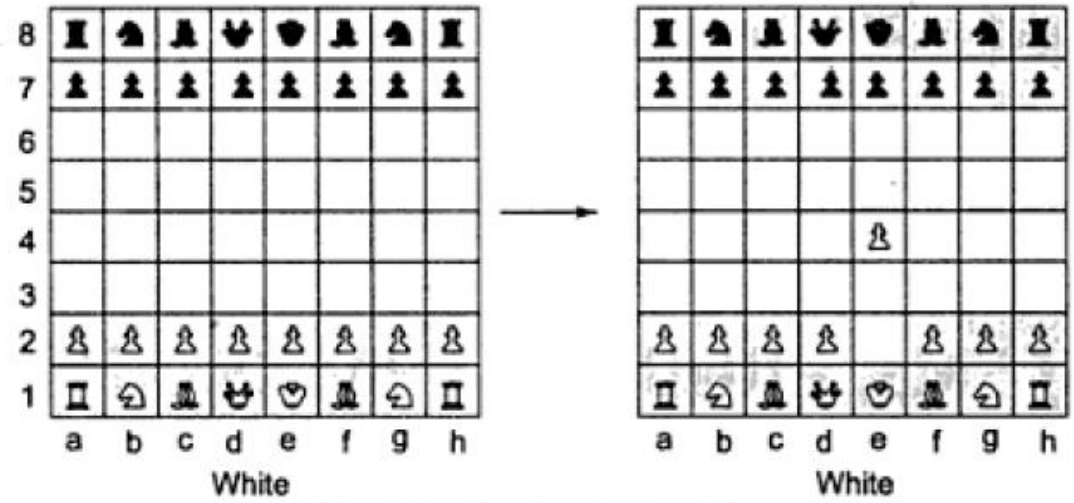


Fig. 2.1 *One Legal Chess Move*

There are several ways in which these rules can be written. (10^{120} possible board positions). We can have board configuration images or write as rules.

Difficulties

1. no person could ever supply a complete set of such rules

2. No program could easily handle all those rules

In order to minimize such problems we should look for a way to write the rules describing the legal moves in as general a way as possible. Eg,

White pawn at
Square(file e, rank 2)
AND
Square(file e, rank 3) → move pawn from Square(file e, rank 2) to Square(file e, rank 4)
is empty
AND
Square(file e, rank 4)
is empty

Fig. 2.2 *Another Way to Describe Chess Moves*

We just defined the problem of playing chess as a problem of moving around in a state space where each state correspond to a legal position of the board. We can then

play chess by starting at an initial states using a set of rules to move from one state to another and attempting to end up in one of a set of final states. This stste space representation seems natural for chess because the set of states which correspond to set of board positions is artificial and well organized.

The state space representation forms the basics of most of the AI methods

Its Structure:

- It allows for a formal definition of a problem as the need to convert some given situation into some desired situation using a set of permissible operations
- It permits us to define the process of solving a particular problem as a combination of known techniques and search the general technique of exploring the space to try to find some path from the current state to goal state. Search is very important process in the solution of hard problems for which no more direct techniques are available

A Water Jug Problem

You are given two jugs, a 4-gallon one and a 3 gallonone. 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 2gallon of water into 4 gallon jar?

The state space for this problem can be described as the set of ordered pair of integers**(x,y) such that x=0,1,2,3 or 4 and y=0,1,2 or 3**.

- x represent number of gallon of water in the 4 gallon jug
- y represent number of gallon of water in the 3 gallon jug

The start state is (0,0).

The goal state is (2,n) n can be any value.

Production rules for water jug problem

1	(x,y) -----> (4,y) If x<4	Fill the 4 gallon jug
2	(x,y) -----> (x,3) If y<3	Fill the 3 gallon jug
3	(x,y) -----> (x-d,y) If x>0	Pour some water out of 4 gallon jug
4	(x,y) -----> (x,y-d) If y>0	Pour some water out of 3 gallon jug
5	(x,y) -----> (0,y) If x>0	Empty the 4 gallon jug on ground
6	(x,y) -----> (x,0)	Empty the 3 gallon jug on

	If y>0	ground
7	(x,y) -----> (4, y-(4-x)) If x+y>=4 and y>0	Pour water from 3 gallon jug into 4 gallon jug until 4 gallon jug is fill
8	(x,y) -----> (x-(3-y), 3) If x+y>=4 and y>0	Pour water from 4 gallon jug into 3 gallon jug until 3 gallon jug is fill
9	(x,y) -----> (x+y,0) If x+y<=4 and y>0	Pour all water from 3 gallon jug into 4 gallon jug
10	(x,y) -----> (0,x+y) If x+y<=4 and y>0	Pour all water from 4 gallon jug into 3 gallon jug
11	(0,2) -----> (2,0) If x+y<=4 and y>0	Pour the 2 gallon from 3 gallon jug into 4 gallon jug.
12	(2,y) -----> (0,y) If x+y<=4 and y>0	Empty the 2 gallon in the 4 gallon on the ground

The state space for this problem can be described as a set of ordered pairs of integers (x,y) such that x=0,1,2,3 or 4 and y= 0,1,2 or 3. x represent number of gallon of water in the 4 gallon jug and y represent number of gallon of water in the 3 gallon jug.

The start state is (0,0).

The goal state is (2,n) n can be any value.

For any value of n, since the problem does not specify how many gallons needed to be in the 3 gallon jug.

Gallons in 4 gallon jug	Gallons in 3 gallon jug	Rule Applied
0	0	2
0	3	9
3	0	2
3	3	7
4	2	5
0	2	11
2	0	Final state

Production System

A production system consist of:

- A **set of rules** each contsisting of a left side that determines the applicability of the rule and right side that describes the operation to be performed if rule is applied.
- One or more **knowledge/database** that contain whatever information is appropriate for the particular task. Some parts of the database may be permenent, while other parts of it may pertain only to the solution of current problem.
- A **control stratergy** that specifies the order in which the rules will be ccompared to the database and a way of resolving the conflicts that arise when several rules match at once
- A **rule applier.**

Control Stratergies

The first requirement of a good control stratergy is that it causes motion. Consider the water jug problem. Suppose we implemented the simple control stratergy of starting each time at the top of the list of rules and choosing the first applicable rule. If we did that we would never solve the problem. We would continue indefinitley filling the 4 gallon jug with water.

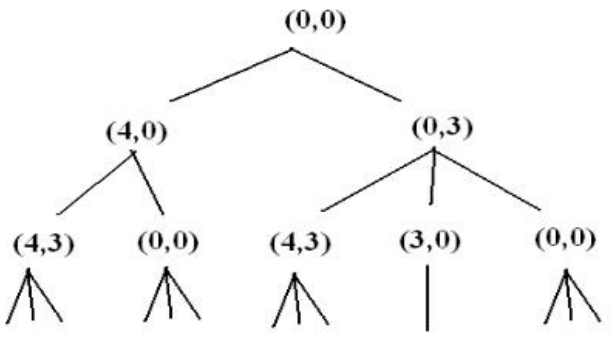
Control stratergies that do not cause motion will never lead to a solution.

The second requirement of a good control stratergy is that it be systematic.

On each cycle choose at random from among the applicable rules. This stratergy is better than the first. It causes motion. It will lead to a solution eventually. But we are likely to arrive at the same state several times suring the process and to use many more steps than are necessary. Because the control stratergy is not systemati, we may explore a particular useless sequence of operators several times before we finally find a solution.

The requirement that a **control stratergy be systemtic** corresponds to the need for global motion as well as for local motion.

One systematic control stratergy for water jug problem is the following. Construct a tree with initial state as its root, generate all offspring of the root by applying each of the applicable rules to the initial state. Now for each leaf node, generate all its sucessors by applying all the rules that are appropriate. Continue this process until some rule produces a goal state. This process is called **Breath First Search.**



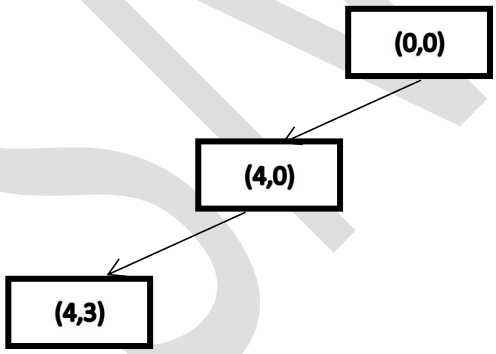
Breadth first Search Algorithm

1. create a variable called NODE_LIST and set it to initial state
2. Until a goal state is found or NODE_LIST is empty
    a) Remove the first element from NODE_LIST and call it E. if NODE_LIST was empty quit
    b) For each way that each rule can match the state described in E do
        i. Apply the rule to generate a new state
        ii. If the new state is a goal state, quit and return this state
        iii. Otherwise add the new state to end of NODE_LIST

Pursue a single branch of tree until a solution or until a decision to determine the path is made. It makes sense to terminate a path if it reaches a dead end. In such a case, backtracking occurs. The most recently created state from which alternative moves are available will be revisited and a new state will be created. This form of backtracking is called chronological backtracking. This search procedure is called depth first search.

Depth first Search Algorithm

1. if the initial state is a goal state, quit and return success
2. Otherwise do the following until success or failure is signaled:
    a) Generate 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 the success is returned, signal success otherwise continue in the loop.



Advantages of BFS

Breath first search will not get trapped exploring a blind alley. This contrasts with DFS which may follow a single, unfruitful path for a very long time, before the path actually terminates in a state that has no successors.

If there is a solution then breath first search is guarenteed to find it. Furthermore if there are multiple solutions, then a minimal solution will be found.

The Traveling Salesman Problem

A 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 starts and finishes at any one of the cities.

A simple motion causing and systematic control structure could in principle solve this problem. It would simply explore all possible path in the tree and return the one with the shortest length. This approach will even work in practice for every short list of cities. But it breaks down quickly as the number of cities grows. If there are N cities then the number of different path among them is 1,2, . . . (N-1) or (N-1)!

Assuming there are only 10cities. 10! is 3628800 which is a large number. The salesman could easily have 25 cities to visit. To solve this problem would take more time than he would take more time than he would be willing to spend. This phenomenon is called combinational explosion. To combat it we need a new control stratergy.

We can beat the simple stratergy outlined above using a technique called branch and bound. Being generating complete paths keeping track of shortest path found so far give up exploring any path as soon as its partial length become greater than the shortest path found so far . Using this technique we are still guarenteed to find the shortest path.

Heuristic Search

In state space search, heuristics are formalized as rule for choosing those branches in a state space that are most likely to lead to an acceptable problem solution.

Two basic solution:

1. a problem may not have an exact solution because of inherent ambiguties in the problem statement or available data. Medical diagnosis is an example of this. A given set of symptoms may have several possible causes, doctors use heuristics to choose the most likely diagnosis and formulate a plan of treatment.

2. A problem may have an exact solution, but the computational cost of finding it may be prohibitive. A heuristic algorithm can defeat this combinational explosion and find an acceptable solution.

Heuristic approach for travelling salesman problem

1. arbitarily select a starting city

2. To select the next city, look at all cities not yet visited and select the one closest to the current city go to it next.

3. Repeat step 2 until all cities have been visited

The procedure executes in time proportional to N^2, a significant improvement over N!

Heuristics and the design algorithms to implement heuristic search have long been a core concern of artificial intelligence research. Game playing and theorm proving are two of the oldest applications in artificial intelligence.

Problem Charateristics

In order to choose the most appropriate method for a particular problem, it is necessary to analyze the problem along several dimensions.

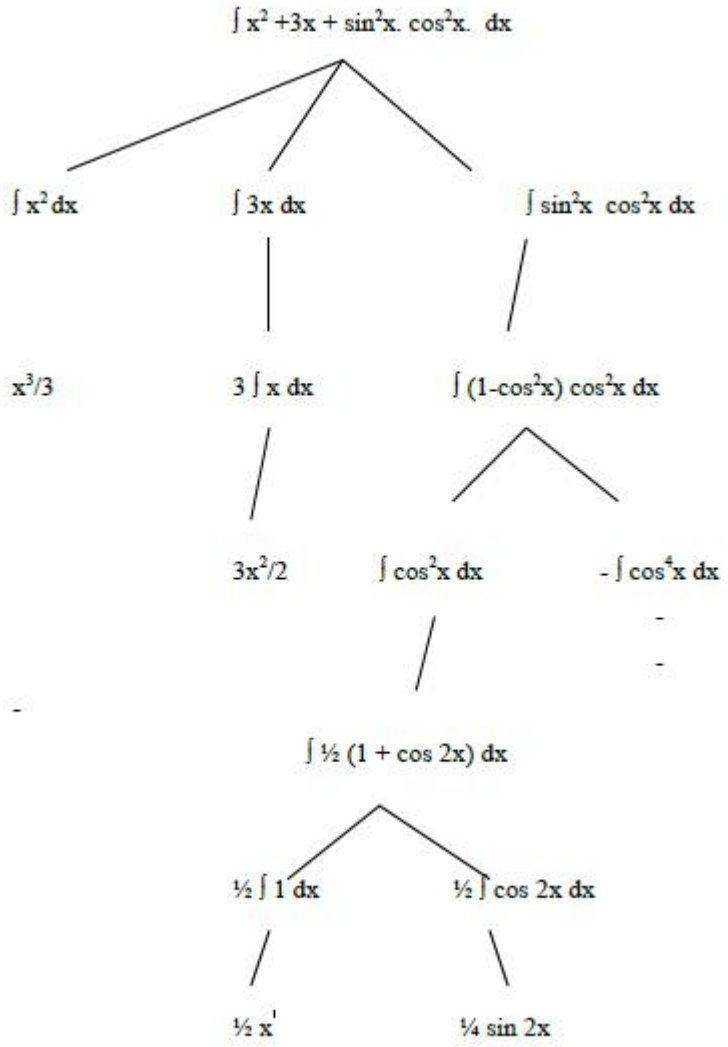
- Is the problem decomposable into a set of independent smaller or easier sub problems?
- Can solution steps be ignored or at least undone if they prove unwise?
- Is the problem's universe predictable?
- Is a good solution to the problem obvious without comparison to all other possible solutions?
- Is the desired solution a state of the world or a path to a state?
- Is a large amount of knowledge absolutely required to solve the problem, or is knowledge important only to constrain the search?
- Can a computer that is simply given the problem return the solution, or will the solution of the problem require interaction between the computer and a person?

1. Is the problem Decomposable?

Suppose we want to solve the problem of computing the expression.

$$\int (x2 + 3x + \sin 2x.\cos 2x)dx$$

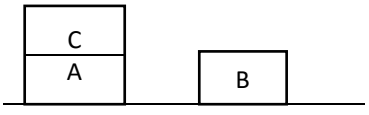
We can solve this problem by breaking it down into three smaller problems, each of which we can then solve by using a small collection of specific rules.



At each step it checks to see whether 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. If it can it creates those problems ans calls itself recursively on them. This technique is called problem decomposition.

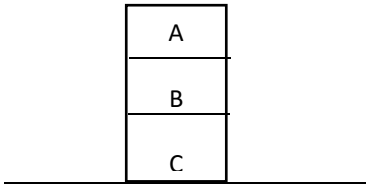
A simple blocks world problem

Start:



ON (C,A)

Goal:



ON(B,C) and ON(A,B)

Assume that the following operators are availale

1. CLEAR(x) ~~ON(x,table)~~

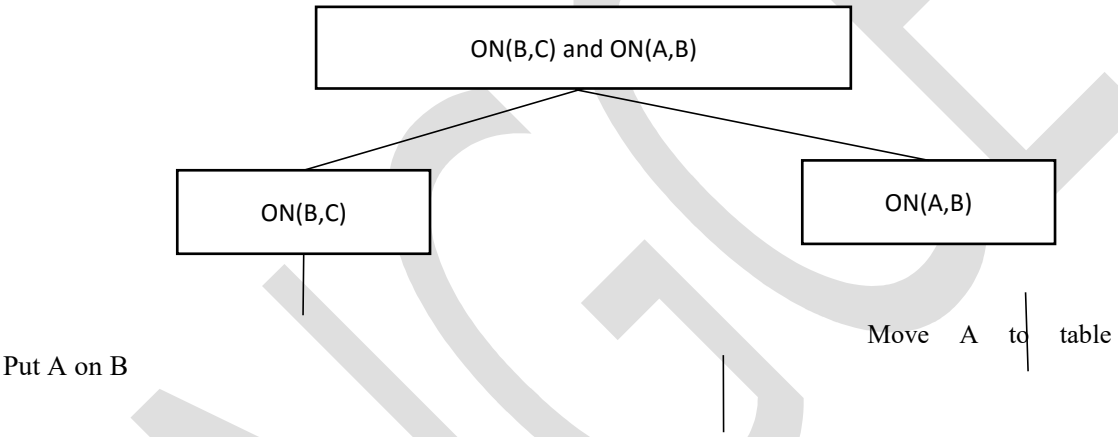
[block x has nothing on it] [pick up x and put it on the table]

2. CLEAR(x) and CLEAR(y) ~~ON(x,y)~~ →

The idea of solution is to reduce the problem of getting B on C and A on B to get two seperated problems. The first of these new problems getting B on C is simple, given he start state. Simply put B on C. The second subgoal is not quite so simple.

Since the only operators we have allow us to pick up single blocks at a time, we have to clear off A by removing C before we can pick up A and put it on B. this can easily be done. However if we now try to combine the two sub solutions into one solution, we will fqail. Regardless of which one we do first we will not be able to do the second as we had planned. In this problem the two sub problems are not independent. They interact and those interactions must be considered in order to arrive at a solution for entire problem.

These two examples symbolic integration and the block world, illustrate the difference between **decomposable and non decomposible problems**



2. Can solution steps be ignored or undone

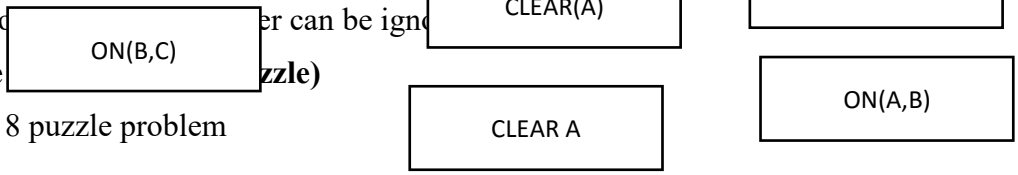
Here we can divide problems into 3 classes.

- Ignorable, in which solution steps can be ignored.
- Recoverable, in which solution steps can be undone.
- Irrecoverable, in which solution steps cannot be undone.

Ignorable Problem (eg, Theorm Proving)

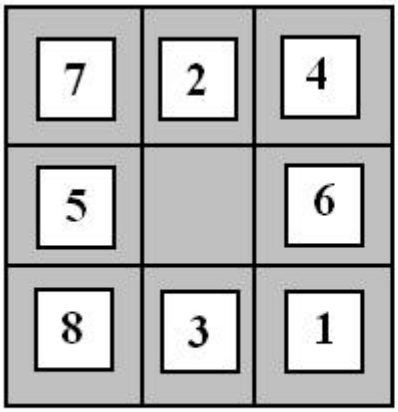
Here solution steps can be ignored.

Suppose we are trying to prove a mathematical theorem. We proceed by first proving a lemma that we think will be useful. Eventually we realize that the lemma is no help at all. Here the different steps in proving the lemma can be ignored start from and ...er can be ignored.

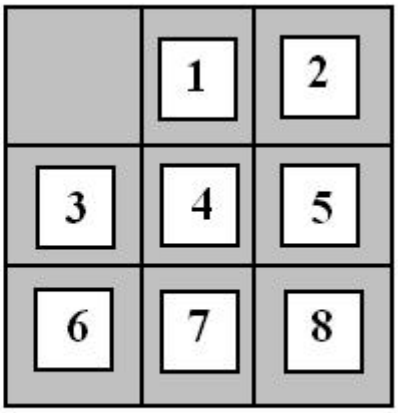


Recoverable ...zzle)

Consider the 8 puzzle problem



Start state



Goal state

The goal is to transform the starting position into the goal position by sliding the tiles around. In an attempt to solve the 8- puzzle, we might make a stupid move. For example, in the game shown above, we might start by sliding tile 5 into the empty space. Having done that, we cannot change our mind and immediately slide tile 6 into the empty space since the empty space will essentially have moved. But we can backtrack and undo the 1st move, sliding tile 5 back to where it was. Then we can move tile 6. here mistakes can be recovered.

Additional step must be performed to undo each incorrect step. The control mechanism for an 8-puzzle solver must keep track of the order in which operations are performed so that the operations can be undone one at a time if necessary.

Irrecoverable problems (eg. Chess)

Consider the problem of playing chess. Suppose a chess playing program makes a stupid move and realizes it a couple of moves later. It cannot simply play as though it had never made the stupid move. Nor can it simply back up and start the game over from that point. All it can do is to try to make the best of the current situation and go from there.

The recoverability of a problem plays an important role in determining the complexity of the control structure necessary for the problem's solution. **Ignorable problems** can be solved using a **simple control structure**. **Recoverable problems** can be solved by a slightly **more complicated** control strategy that does sometimes makes mistakes. **Irrecoverable** problems will need to be solved by a system that expends a great deal of **effort making each decision** since the decision must be final.

3. Is the universe predictable?

Certain outcome problems (eg. 8 puzzle)

Suppose we are playing with the 8 puzzle problem. Every time we make a move, we know exactly what will happen. This means that it is possible to plan an entire sequence of moves and be confident that we know what the resulting state will be.

Uncertain outcome problems (eg. Bridge)

However, in games such as bridge, this planning may not be possible. One of the decisions we will have to make is which card to play on the first trick. What we would like to do is to plan the entire hand before making that first play. But now it is not

possible to do such planning with certainty since we cannot know exactly where all the cards are or what the other players will do on their turns.

Planning can be used to generate a sequence of operators that is guarenteed to lead to a solution. For uncetain outcome problems, planning can at best generate a sequence of operators that has a good probability of leading to a solution. In which the outcome cannot be predicted.

One of the hardest types of problems to solve is the **irrecoverable, uncertain outcome**.

Examples of such problems are

- Playing bridge,
- Controlling a robot arm,
- Helping a lawyer decide how to defend his client against a murder charge.

4. Is a good solution Absolute or relative

Any path problems

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 Pompean.
3. Marcus was born in 40 A. D.
4. all men are mortal.
5. All pompeans 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 the 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. The following shows 2 ways of deciding that Marcus is dead.

	Solutions	Axiom
1	Marcus was a man.	1
4	All men are mortal.	4
3	Marcus was born in 40 A.D.	3
7	It is now 2017 A. D.	7
9	Marcus' age is 1977 years.	3,7
6	no mortal lives longer than 150 years.	6

10	Marcus is dead.	8,6,9

OR

	Solutions	Axiom
1	It is now 2017AD	7
5	All Pompeians died in 79 AD	5
11	All pompeians are dead now	7,5
2	Marcus was a pompeian	2
12	Marcus is dead.	11,2

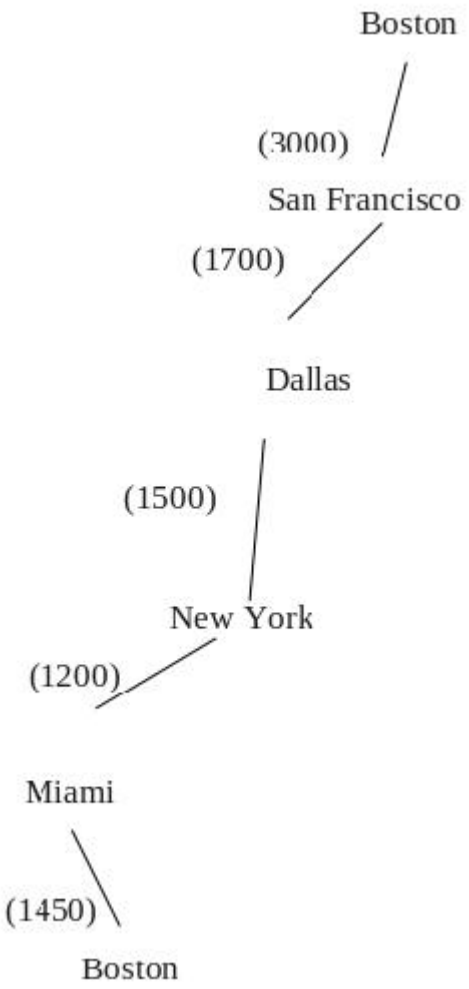
Since all we are interested in is the answer to the question, it does not matter which path we follow.

Best path problems (eg. Traveling salesman problem)

Consider the traveling salesman problem. Our goal is to find the shortest route that visits each city exactly once. Suppose the cities to be visited and the distances between them are shown below.

	Boston	NY	Miami	Dallas	SF
Boston		250	1450	1700	3000
NY	250		1200	1500	2900
Miami	1450	1200		1600	3300
Dallas	1700	1500	1600		1700
SF	3000	2900	3300	1700	

One place the salesman could start is Boston. In that case, one path that might be followed is the one shown below which is 8850 miles long.



But is this the solution to the problem? The answer is that we cannot be sure unless we also try all other paths to make sure that none of them is shorter.

Best path problems are computationally harder than any path problems. Any path problem can often be solved in a reasonable amount of time by using heuristics that suggest good path to explore.

5. <u>Is the solution a state or path?</u>

Problems whose solution is a state of the world. eg. Natural language understanding

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, in isolation, 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 financial institution or to a side of a river. The word ’dish’ is the object of the verb ‘eat’. It is possible that a dish was eaten. But it is more likely that the pasta salad in the dish was eaten.

Pasta salad is a salad containing pasta. But there are other ways interpretations can be formed from pairs of nouns.

For example, dog food does not normally contain dogs. The phrase 'with the fork' could modify several parts of the sentence. In this case, it modifies the verb 'eat'. But, if the phrase had been 'with vegetables', then the modification structure would be different. Because of the interaction among the interpretations of the constituents of this sentence, some search may be required to find a complete interpretation 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.

Problems whose solution is a path to a state? Eg. Water jug problem

In water jug problem, it is not sufficient to report that we have solved the problem and that the final state is (2,0). For this kind of problem, what we really must report is not the final state, but the path that we found to that state.

6. What is the Role of Knowledge

Problems for which a lot of knowledge is important only to constrain the search for a solution.

Eg. Chess

Consider the problem of playing chess. How much knowledge would be required by a perfect chess playing program?

Just the rules for determining the legal moves and some simple control mechanism that implements an appropriatesearch procedure.

Problems for which a lot of knowledge is required even to be able to recognize a solution.

Eg. News paper story understanding

Consider the problem of scanning daily newspapers to decide which are supporting democrats and which are supporting the republicans in some upcoming election. How much knowledge would be required by a computer trying to solve this problem? Here a great deal of knowledge is necessary.

7. Does the task require interaction with a person?

Solitary problems

Here the computer is given a problem description and produces an answer with no intermediate communication and with no demand for an explanation for the reasoning process.

Consider the problem of proving mathematical theorems. If

- All we want is to know that there is a proof.
- The program is capable of finding a proof by itself.

Then it does not matter what strategy the program takes to find the proof.

Conversational problems

In which there is intermediate communication between a person and the computer, either to provide additional assistance to the computer or to provide additional information to the user. Eg. Suppose we are trying to prove some new, very difficult theorem. Then the program may not know where to start. At the moment, people are still better at doing the high level strategy required for a proof. So the computer might like to be able to ask for advice. To exploit such advice, the computer's reasoning must be analogous to that of its human advisor, at least on a few levels.

SEARCHING

Problem solving in artificial intelligence may be characterized as a systematic search through a range of possible actions in order to reach some predefined goal or solution. In AI problem solving by search algorithms is quite common technique. In the coming age of AI it will have big impact on the technologies of the robotics and path finding. It is also widely used in travel planning. This chapter contains the different search algorithms of AI used in various applications. Let us look the concepts for visualizing the algorithms. A search algorithm takes a problem as input and returns the solution in the form of an action sequence. Once the solution is found, the actions it recommends can be carried out. This phase is called as the execution phase. After formulating a goal and problem to solve the agent cells a search procedure to solve it. A problem can be defined by 5 components.

a) The initial state: The state from which agent will start.

b) The goal state: The state to be finally reached.

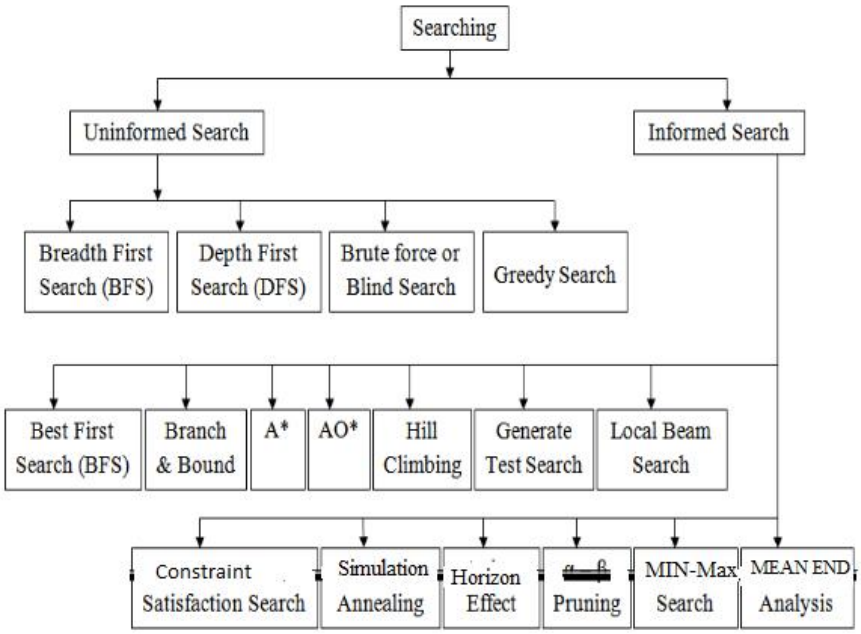
c) The current state: The state at which the agent is present after starting from the initial state.

d) Successor function: It is the description of possible actions and their outcomes.

e) Path cost: It is a function that assigns a numeric cost to each path.

DIFFERENT TYPES OF SEARCHING

The searching algorithms can be various types. When any type of searching is performed, there may some information about the searching or mayn't be. Also it is possible that the searching procedure may depend upon any constraints or rules. However, generally searching can be classified into two types i.e. uninformed searching and informed searching. Also some other classifications of these searches are given below in the figure



Informed Search

Informed Search

- A search using domain-specific knowledge.
- Suppose that we have a way to estimate how close a state is to the goal, with an evaluation function.
- General strategy: expand the best state in the open list first. It's called a best-first search or ordered state-space search.
- In general the evaluation function is imprecise, which makes the method a heuristic (works well in most cases).
- The evaluation is often based on empirical observations.

Informed (Heuristic) Search Strategies

To solve large problems with large number of possible states, problem-specific knowledge needs to be added to increase the efficiency of search algorithms.

Heuristic Evaluation Functions

They calculate the cost of optimal path between two states. A heuristic function for sliding-tiles games is computed by counting number of moves that each tile makes from its goal state and adding these number of moves for all tiles.

Pure Heuristic Search

It expands nodes in the order of their heuristic values. It creates two lists, a closed list for the already expanded nodes and an open list for the created but unexpanded nodes.

In each iteration, a node with a minimum heuristic value is expanded, all its child nodes are created and placed in the closed list. Then, the heuristic function is applied to the child nodes and they are placed in the open list according to their heuristic value. The shorter paths are saved and the longer ones are disposed.

A * Search

It is best-known form of Best First search. It avoids expanding paths that are already expensive, but expands most promising paths first.

f(n) = g(n) + h(n), where

- g(n) the cost (so far) to reach the node
- h(n) estimated cost to get from the node to the goal
- f(n) estimated total cost of path through n to goal. It is implemented using priority queue by increasing f(n).

Greedy Best First Search

It expands the node that is estimated to be closest to goal. It expands nodes based on f(n) = h(n). It is implemented using priority queue.

Disadvantage − It can get stuck in loops. It is not optimal.

Local Search Algorithms

They start from a prospective solution and then move to a neighboring solution. They can return a valid solution even if it is interrupted at any time before they end.

Generate-And-Test Algorithm

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

Algorithm: Generate-And-Test

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

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

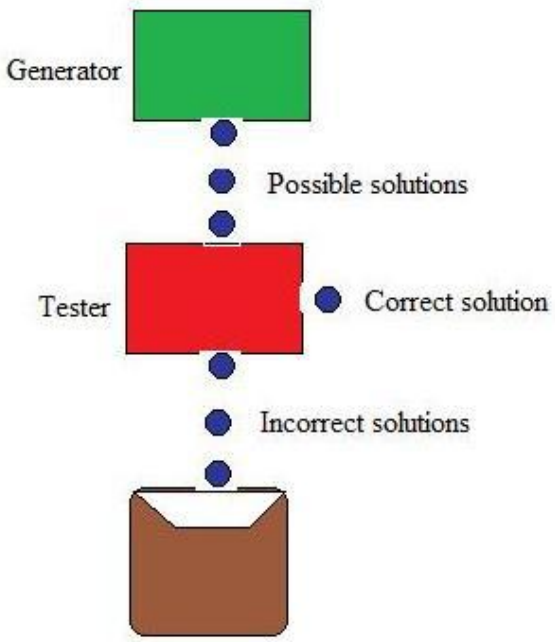


Figure: Generate And Test

Generate-and-test, like depth-first search, requires that complete solutions be generated for testing. In its most systematic form, it is only an exhaustive search of the problem space. Solutions can also be generated randomly but solution is not guaranteed. This approach is what is known as British Museum algorithm: finding an object in the British Museum by wandering randomly.

Systematic Generate-And-Test

While generating complete solutions and generating random solutions are the two extremes there exists another approach that lies in between. The approach is that the search process proceeds systematically but some paths that unlikely to lead the solution are not considered. This evaluation is performed by a heuristic function.

Depth-first search tree with backtracking can be used to implement systematic generate-and-test procedure. As per this procedure, if some intermediate states are likely to appear often in the tree, it would be better to modify that procedure to traverse a graph rather than a tree.

Generate-And-Test And Planning

Exhaustive generate-and-test is very useful for simple problems. But for complex problems even heuristic generate-and-test is not very effective technique. But this may be made effective by combining with other techniques in such a way that the space in which to search is restricted. An AI program DENDRAL, for example, uses plan-Generate-and-test technique. First, the planning process uses constraint-satisfaction techniques and creates lists of recommended and contraindicated substructures. Then the generate-and-test procedure uses the lists

generated and required to explore only a limited set of structures. Constrained in this way, generate-and-test proved highly effective. A major weakness of planning is that it often produces inaccurate solutions as there is no feedback from the world. But if it is used to produce only pieces of solutions then lack of detailed accuracy becomes unimportant.

Heuristic Search Techniques.

Introduction:- Many if the problems are too complex to be solvable by direct techniques. They have to be solved only by suitable heuristic search techniques. Though the heuristic techniques can be described independently, they are domain specific. They are called "Weak Methods", since they are vulnerable to combinatorial explosion. Even so, they provide the frame work into which domain specific knowledge can be placed.

Every search process can be viewed as a traversal of a directed graph, in which the nodes represent problem states and the arcs represent relationships between states. The search process must find a path through this graph, starting at an initial state and ending in one or more final states. The following issues have to be considered before going for a search.

Heuristic Search Techniques.

Heuristic techniques are called weak methods, since they are vulnerable to combinatorial explosion. Even then these techniques continue to provide framework into which domain specific knowledge can be placed, either by hand or as a result of learning. The following are some general purpose control strategies (often called weak methods).

- Generate - and - test
- Hill climbing
- Breadth - First search
- Depth - First search
- Best First Search (A* search)
- Problem reduction(AO* search)
- Constraint satisfaction
- Means - ends analysis

A heuristic procedure, or heuristic, is defined as having the following properties.

1. It will usually find good, although not necessary optimum solutions.
2. It is faster and easier to implement than any known exact algorithm (one which guarantees an optimum solution).

In general, heuristic search improve the quality of the path that are exported. Using good heuristics we can hope to get good solutions to hard problems such as the traveling salesman problem in less than exponential time. There are some good general purpose heuristics that are useful in a wide variety of problems. It is also possible to construct special purpose heuristics to solve particular problems. For example, consider the traveling salesman problem.

Hill Climbing

Hill climbing search algorithm is simply a loop that continuously moves in the direction of increasing value. It stops when it reaches a “peak” where no neighbour has higher value. This algorithm is considered to be one of the simplest procedures for implementing heuristic search. The hill climbing comes from that idea if you are

trying to find the top of the hill and you go up direction from where ever you are. This heuristic combines the advantages of both depth first and breadth first searches into a single method. The name hill climbing is derived from simulating the situation of a person climbing the hill. The person will try to move forward in the direction of at the top of the hill. His movement stops when it reaches at the peak of hill and no peak has higher value of heuristic function than this. Hill climbing uses knowledge about the local terrain, providing a very useful and effective heuristic for eliminating much of the unproductive search space. It is a branch by a local evaluation function. The hill climbing is a variant of generate and test in which direction the search should proceed. At each point in the search path, a successor node that appears to reach for exploration.

Algorithm:

Step 1: Evaluate the starting state. If it is a goal state then stop and return success.
Step 2: Else, continue with the starting state as considering it as a current state.
Step 3: Continue step-4 until a solution is found i.e. until there are no new states left to be applied in the current state.
Step 4: a) Select a state that has not been yet applied to the current state and apply it to produce a new state.
b)Procedure to evaluate a 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 5:Exit.

Advantages:

- Hill climbing technique is useful in job shop scheduling, automatic programming, circuit designing, and vehicle routing and portfolio management.
- It is also helpful to solve pure optimization problems where the objective is to find the best state according to the objective function.
- It requires much less conditions than other search techniques.

Disadvantages:

The question that remains on hill climbing search is whether this hill is the highest hill possible.
Unfortunately without further extensive exploration, this question cannot be answered. This technique works but as it uses local information that's why it can be fooled. The algorithm doesn't maintain a search tree, so the current node data structure need only record the state and its objective function value. It assumes that local improvement will lead to global improvement.
There are some reasons by which hill climbing often gets suck which are stated below.

Local Maxima:

A local maxima is a state that is better than each of its neighbouring states, but not better than some other states further away. Generally this state is lower than the global maximum. At this point, one cannot decide easily to move in which direction! This difficulties can be extracted by the process of backtracking i.e. backtrack to any of one earlier node position and try to go on a different event direction. To implement this strategy, maintaining in a list of path almost taken and go back to one of them. If the path was taken that leads to a dead end, then go back to one of them.

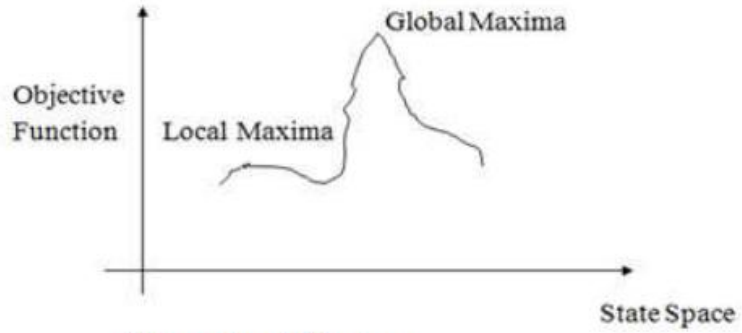


Figure Local Maxima

Ridges:

It is a special type of local maxima. It is a simply an area of search space. Ridges result in a sequence of local maxima that is very difficult to implement ridge itself has a slope which is difficult to traverse. In this type of situation apply two or more rules before doing the test. This will correspond to move in several directions at once.

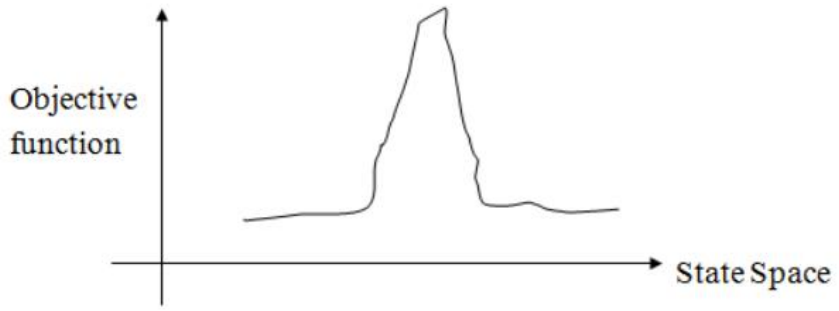


Figure Ridges

Plateau:

It is a flat area of search space in which the neighbouring have same value. So it is very difficult to calculate the best direction. So to get out of this situation, make a big jump in any direction, which will help to move in a new direction this is the best way to handle the problem like plateau.

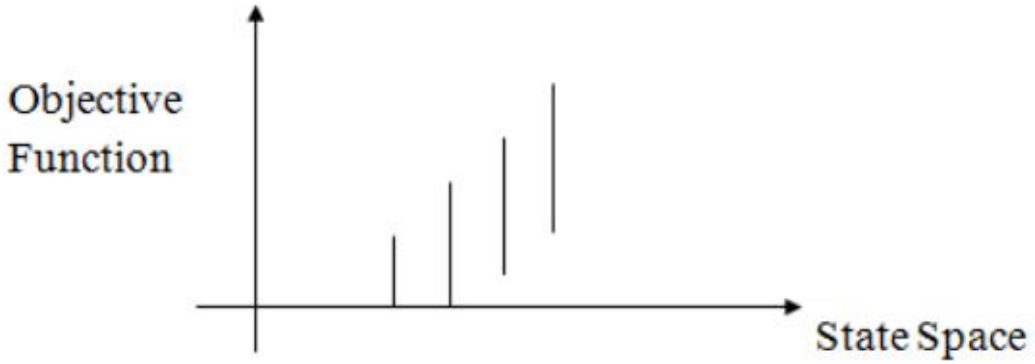


Figure Plateau

Travelling Salesman Problem

In this algorithm, the objective is to find a low-cost tour that starts from a city, visits all cities en-route exactly once and ends at the same starting city.

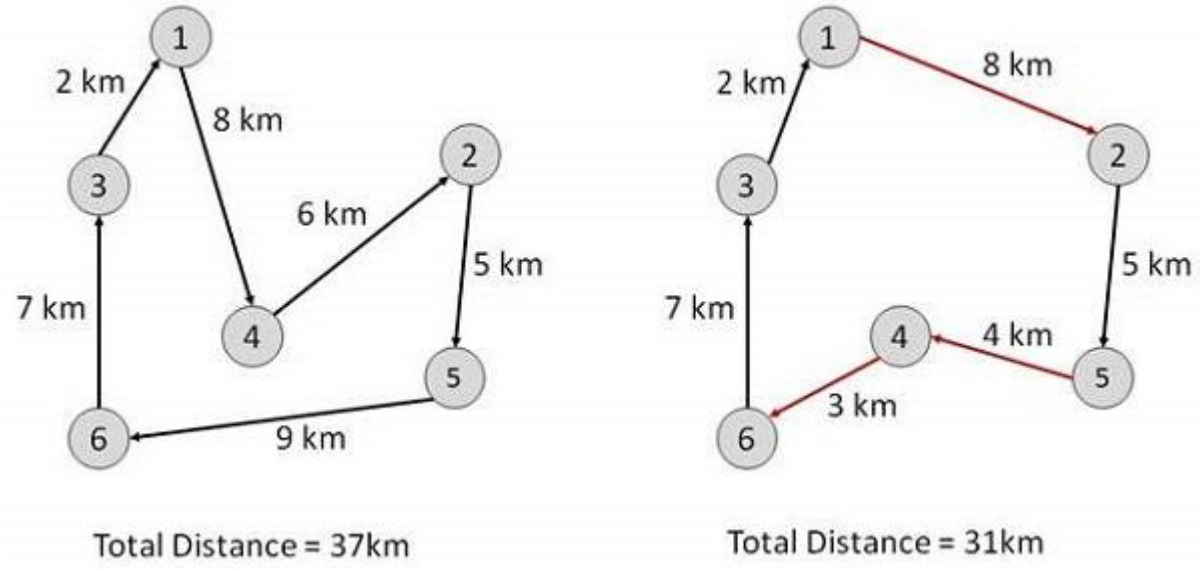
Start

Find out all (n -1)! Possible solutions, where n is the total number of cities.

Determine the minimum cost by finding out the cost of each of these (n -1)! solutions.

Finally, keep the one with the minimum cost.

end



Uninformed Search Strategies

A problem determines the graph and the goal but not which path to select from the frontier. This is the job of a search strategy. A search strategy specifies which paths are selected from the frontier. Different strategies are obtained by modifying how the selection of paths in the frontier is implemented.

This section presents three **uninformed search strategies** that do not take into account the location of the goal. Intuitively, these algorithms ignore where they are going until they find a goal and report success.

- Depth-First Search
- Breadth-First Search
- Lowest-Cost-First Search