MODULE 1

Problems- problem spaces and search, production systems, Problem characteristics, Searching Strategies – Generate and Test, Heuristic Search Techniques- Hill climbing– issues in hill climbing, General Example Problems.
Python-Introduction to Python- Lists Dictionaries & Tuples in Python- Python implementation of Hill Climbing

Artificial Intelligence (ref: AI by Rich & Knight)

Artificial intelligence is the study of how to make computers do things which, at the moment, people do better. Much of the early work in AI focused on **formal tasks, such as game playing and theorem proving.**

Ex. For game playing programs are

Checkers playing program,
Chess.

Ex. For theorem proving programs are

Logic theorist,
Galernter's theorem prover.

Another area of AI is **common sense reasoning.** It includes reasoning about physical objects and their relationships to each other as well as reasoning about actions and their consequences.

Eg. General problem solver (GPS)

As AI research progressed, new tasks were solved such as perception (vision and speech), natural language understanding and problem solving in domains such as medical diagnosis and chemical analysis.

Mundane tasks

- Perception
  - Vision
  - Speech
- Natural language
  - Understanding
  - Generation
  - Translation
- Commonsense reasoning
- Robot control

Formal tasks

- Games
    - Chess
    - Backgammon
    - Checkers
    - Go
- Mathematics
    - Geometry
    - Logic
    - Integral calculus
    - Proving properties of programs

Expert tasks

- Engineering
    - Design
    - Fault finding
    - Manufacturing planning
- Scientific analysis
- Medical diagnosis
- Financial analysis

Definitions (AI by Russel and Norvig)

Definitions of artificial intelligence according to 8 text books are given below.

Artificial intelligence is a system that thinks like human beings.

1. **AI is an exciting effort to make computers think… machines with minds, in the full and literal sense.**
2. **AI is the automation of activities that we associate with human thinking, activities such as decision making, problem solving, learning..**

Artificial intelligence is a system that thinks rationally.

3. **AI is the study of mental faculties through the use of computational models.**
4. **AI is the study of computations that make it possible to perceive, reason and act.**

Artificial intelligence is a system that acts like human beings.

5. **AI is the art of creating machines that perform functions that require intelligence when performed by people.**
6. **AI is the study of how to make computers do things at which , at the moment , people do better.**

Artificial intelligence is a system that acts rationally.

7. **AI is the study of the design of intelligent agents.**
8. **AI is concerned with intelligent behavior in artifacts.**

AI application areas (AI by Luger)

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.

- Game playing
    - Heuristics
- Automated reasoning and theorem proving
- Expert systems
- Natural language understanding and semantic modeling
- Planning and Robotics
- Machine learning
- Neural networks and Genetic algorithms

- **Game playing**

  Much of the early research in AI was done using common board games such as checkers, chess and the 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.

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.

- **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.

- **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.

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.

- **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.

- **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 are planning 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.

- **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.

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.

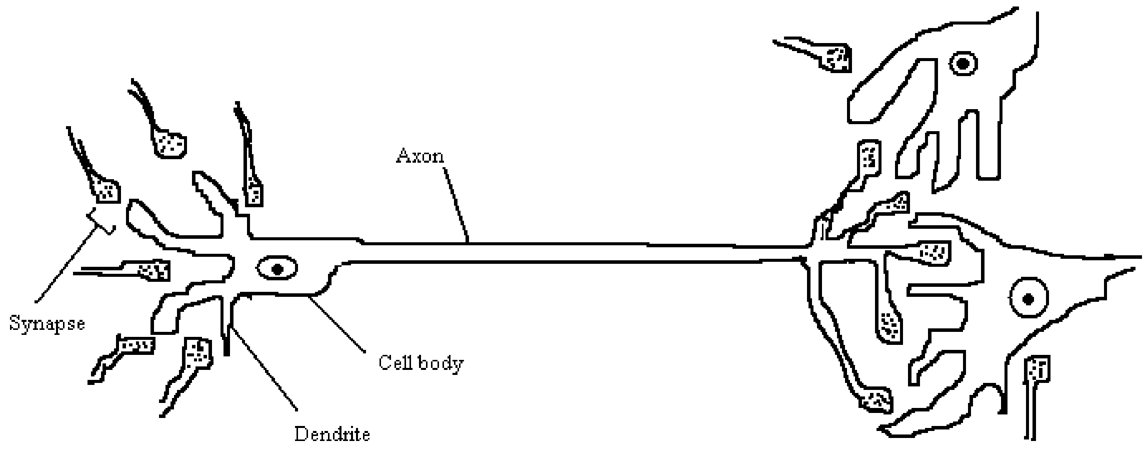
Early work includes Winston's research on the induction of structural concepts such as "arch" from a set of examples in the blocks world.

The ID3 algorithm has proved successful in learning general patterns from examples.

- **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.

Example problems (AI by Russel)

Problems can be classified as toy **problems and real world problems.**

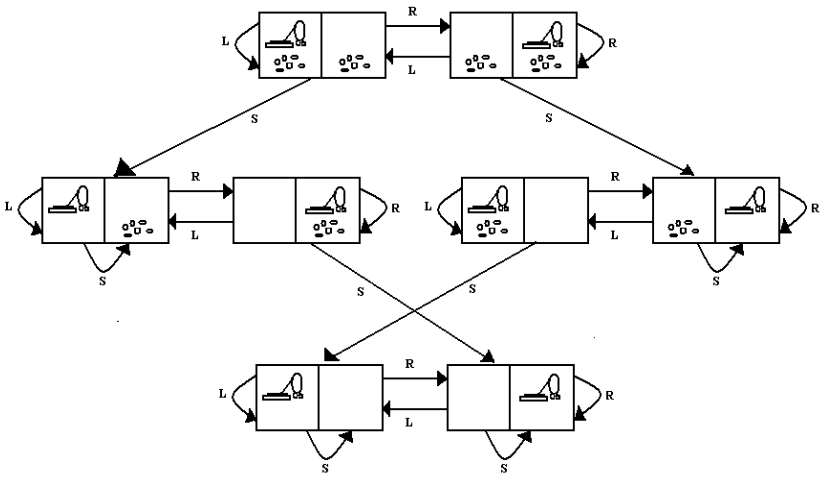
A toy problem is intended to illustrate or exercise various problem solving methods. It can be used by different researchers to compare the performance of algorithms.

A real world problem is one whose solutions people actually care about.

Toy problems

The first example is the

Vacuum world

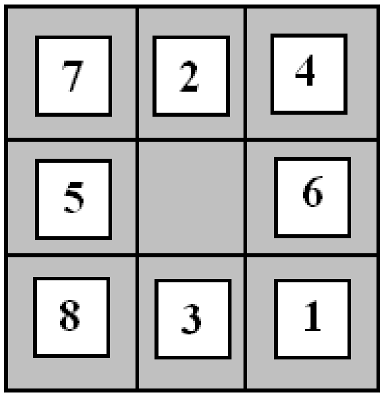


- The agent is in one of the 2 locations, each of which might or might not contain dirt.
- Any state can be designated as the initial state.
- After trying these actions (Left, Right, Suck), we get another state.
- The goal test checks whether all squares are clean.

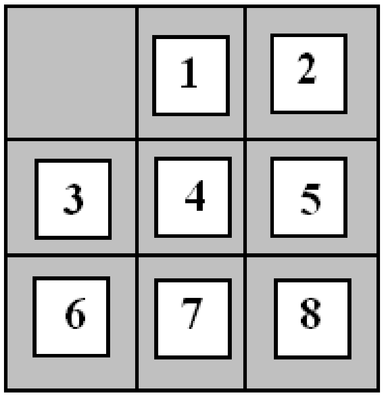
8 – puzzle problem

It consists of a 3 X 3 board with 8 numbered tiles and a blank space as shown below.

A tile adjacent to the blank space can slide into the space. The aim is to reach a specified goal state, such a s the one shown on the right of the figure.



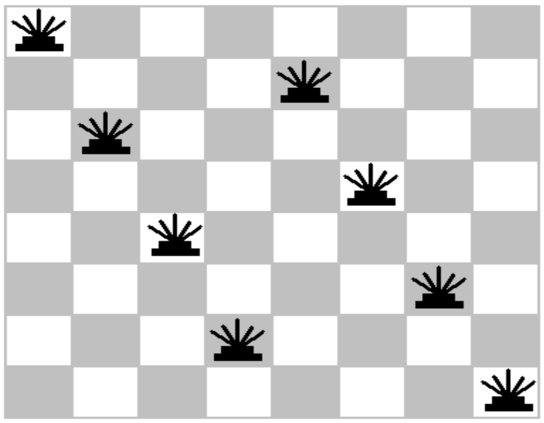
Start state



Goal state

8 – Queens problem

The goal of 8- queens' problem is to place 8 queens on a chess board such that no queen attacks any other.



(a queen attacks any piece in the same row, column or diagonal). Figure shows an attempted solution that that fails: the queen in the rightmost column is attacked by the queen at the top left.

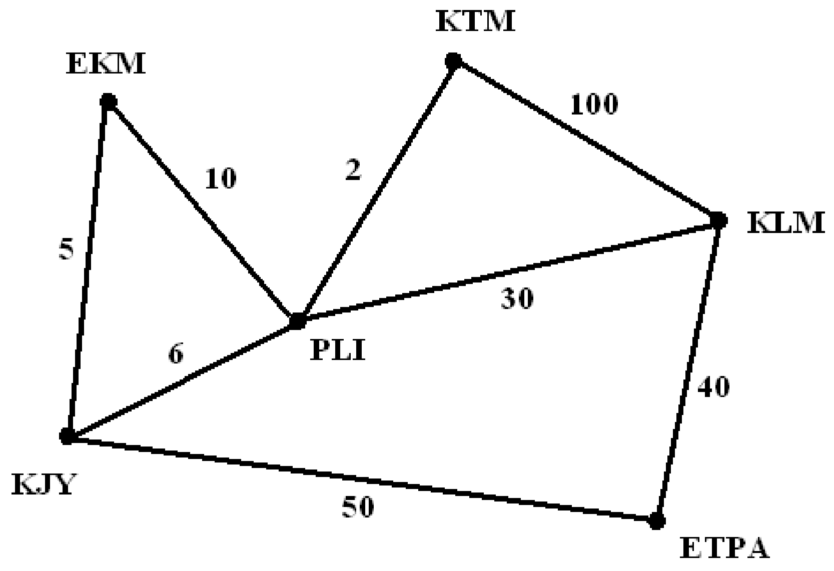
Real world problems

- **Route finding problem**

Route finding algorithms are used in a variety of applications, such as routing in computer networks, military operations planning and air line travel planning systems.

Touring problems

They are related to route finding problems. For example, consider the figure.



Consider the problem. 'Visit every city in the figure at least once starting and ending in Palai'. Each state must include not just the current location but also the set of cities the agent has visited.

Traveling salesperson problem (TSP)

Is a touring problem in which each city must be visited exactly once. The aim is to .find the shortest tour.

VLSI layout problem

It requires positioning of millions of components and connections on a chip to minimize area, minimize circuit delays, minimize stray capacitances, and maximize manufacturing yield.

Robot navigation

It is a generalization of the route finding problem. Rather than a discrete set of routes, a robot can move in a continuous space with an infinite set of possible actions and states.

Automatic assembly sequencing of complex objects by a robot

The assembly of intricate objects such as electric motors is economically feasible. In assembly problems, the aim is to find an order in which to assemble the parts of some object. If the wrong order is chosen, there will be no way to add some part later in the sequence without undoing some of the work already done.

Protein design

Here the aim is to find a sequence of amino acids that will fold into a three dimensional protein with the right properties to cure some disease.

Internet searching

It means looking for answers to questions for related information or for shopping details. Software robots are being developed for performing this internet searching.

PROBLEMS & PROBLEM SPACES

Problems (AI by Ritchie & Knight)

We have seen different kinds of problems with which AI is typically concerned. To build a system to solve a particular problem, we need to do 4 things.

1. Define the problem precisely. This includes the initial state as well as the final goal state.
2. Analyze the problem.
3. Isolate and represent the knowledge that is needed to solve the problem.
4. Choose the best problem solving technique and apply it to the particular problem.

Problem spaces (AI by Ritchie & Knight)

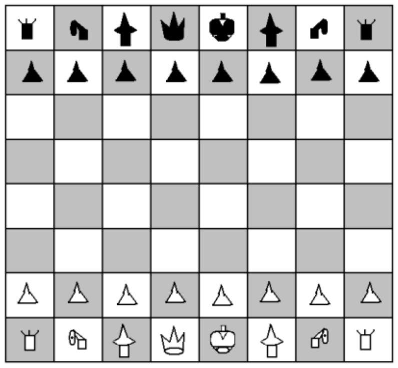
Problem space -- Define the problem as state space search

Suppose we are given a problem statement "play chess". This now stands as a very incomplete statement of the problem we want solved.

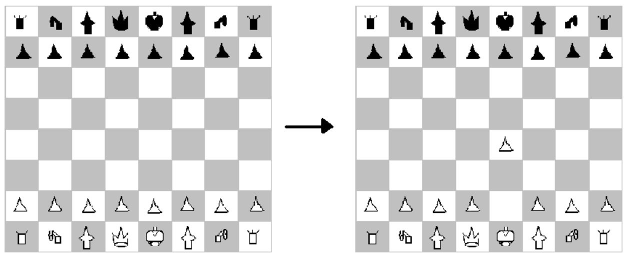
To build a program that could "play chess", we could first have to specify the starting position of the chess board, the rules that define the legal moves and the board positions that represent a win for one side or the other.

For this problem "play chess", it is easy to provide a formal and complete problem description. The starting position can be described as an 8 by 8 array as follows.

We can define as our goal any board position in which the opponent does not have a legal move and his or her king is under attack.



The legal moves provide the way of getting the initial state to a goal state. They can be described easily as a set of rules consisting of 2 parts: a left side that serves as a pattern to be matched against the current board position and a right side that describes the change to be made to the board position to reflect the move.



one legal chess move

We have defined the problem of playing chess as a problem of moving around in a **state space,** where each state corresponds to a legal position of the board. We can then play chess by starting at an initial state, 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.

There are two serious practical issues in defining rules,

- No person could supply complete set of such rules
- No program could easily handle all those rules

In order to reduce such issues we use patterns and substitutions. Like **production rules**. They are represented as rules whose left sides are matched against the current state whose right side represents the new state results from applying the rules.

White pawn at Square (file e, rank 2) move pawn from Square(file e, rank 2)

AND Square (file e, rank 3) is empty → to Square(file e, rank 4)

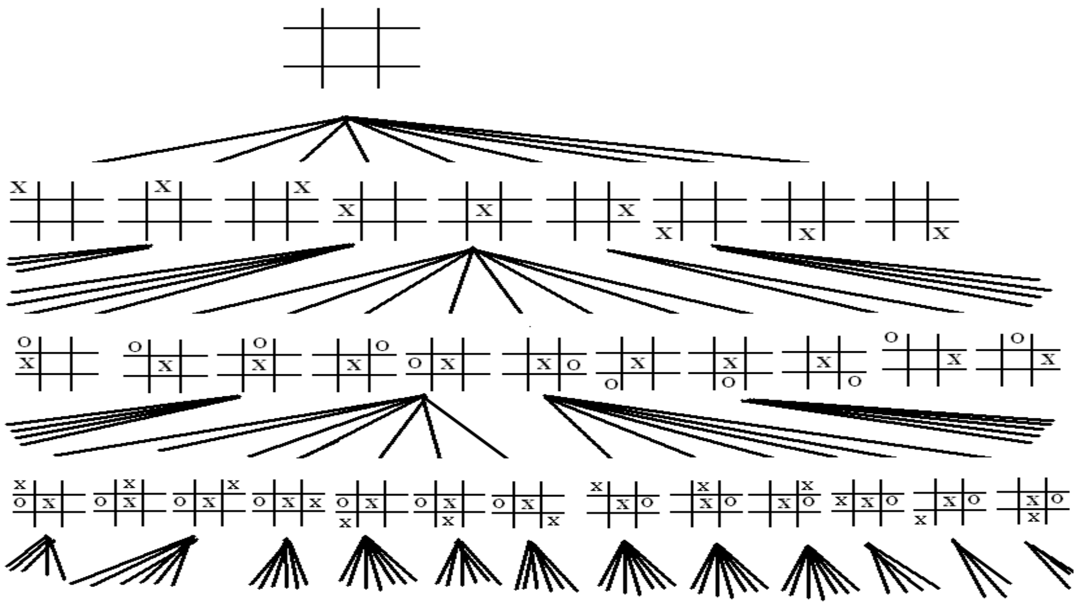
AND Square (file e, rank 4) is empty

. This state space representation forms the basis of most of the AI methods. It helps the problem solving in two-ways

- It allows the rules to define some given situation to some desired situations
- Allows to solve problem using some known techniques and search techniques
- Example 2

The game of tic-tac-toe

Starting with an empty board, the first player may place an X in any one of nine places. Each of these moves yields a different board that will allow the opponent 8 possible responses and so on. We can represent this collection of possible moves and responses by regarding each board configuration as a node in a graph. The links of the graph represent legal moves from one board configuration to another. These nodes correspond to different states of the game board. The resulting structure is called a **state space graph.**



Water jug problem

Problem statement:

Given two jugs, a 4-gallon and 3-gallon having no measuring markers on them. There is a pump that can be used to fill the *jugs with water. How can you get exactly 2 gallons of water into 4-gallon jug.*

The state space for this problem can be described as the 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 represents the number of gallons of water in the 4 gallon jug, and y represents the quantity of water in the 3 gallon jug.

Solution*: State* space for this problem can be described as the set of ordered pairs of integers (X, Y) such that X represents the number of gallons of water in 4-gallon jug and Y for 3-gallon jug.

1. Start state is (0,0)
2. Goal state is (2, N) for any value of N.

Following are the production rules for this problem

1. (x, y) $\rightarrow$ (4, y)
   if $x < 4$
2. (x, y) $\rightarrow$ (x, 3)
   if $y < 3$
3. (x, y) $\rightarrow$ (x – d, y)
   if $x > 0$
4. (x, y) $\rightarrow$ (x, y – d)
   if $y > 0$
5. (x, y) $\rightarrow$ (0, y)
   if $x > 0$
6. (x, y) $\rightarrow$ (x, 0)
   if $y > 0$
7. (x, y) $\rightarrow$ (4, y – (4 – x))
   if $x + y \geq 4, y > 0$
8. (x, y) $\rightarrow$ (x – (3 – y), 3)
   if $x + y \geq 3, x > 0$

   (x, y) $\rightarrow$ (x + y, 0)
   if $x + y \leq 4, y > 0$

10. (x, y) $\rightarrow$ (0, x + y)
    if $x + y \leq 3, x > 0$
11. (0, 2) $\rightarrow$ (2, 0)
12. (2, y) $\rightarrow$ (0, y)

1. current state = (0, 0)
2. Loop until reaching the goal state (2, 0)

   - Apply a rule whose left side matches the current state

   - Set the new current state to be the resulting state

(0, 0)

(0, 3) 2

(3, 0) 9

(3, 3) 2

(4, 2) 7 (0, 2) 5 or 12 (2, 0) 9 or 11

In order to provide a formal description of a problem, we must do the following.

1. Define a state space that contains all the possible configurations of the relevant objects.
2. Specify one or more states within that space that describe possible situations from which the problem solving process may start. These states are called the initial states.
3. Specify one or more states that would be acceptable as solutions to the problem. These states are called goal states.
4. Specify a set of rules that describe the actions available.

Production Systems

Production systems are useful to structure AI programs to describe search process… ie. it describes the way to find the solution.

A production system consists of:

- A set of rules, each consisting of a left side that determines the applicability of the rule and a right side that describe the action to be performed if rule is applied.
- One or more knowledge/database that contain whatever information appropriate for a task.
- A control strategy that specifies the order in which rules will be compared to the database and a way of resolving the conflicts that arise when several rules match at once.
- A rule applier

  Different classes of production systems are:

- A **monotonic production system** : in which the application of rule never prevents the latter 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 a rule transforms state x into state y, then any permutation of those rules also transforms state x into state y. A **commutative production system** is a production system that is both monotonic and partially commutative.

	monotonic	**Non-monotonic**
partially commutative	Theorem proving	Robot navigation
Not partially commutative	Chemical synthesis	Bridge

Control Strategies

When more than one rule matches the current state, we have 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**

   Otherwise, it will never lead to a solution.

Eg: suppose we are searching the rule from the beginning and selecting the first applicable one. We would never solve the problem. Keep on filling the 4-gallon jug

2. **It is systematic**

   Otherwise, it may use more steps than necessary.

Eg: suppose we are randomly selecting the rule. It causes change & will lead to solution eventually. Because it is not systematic, we may enter useless sequences of operations.

3. **It is efficient**

   Find a good, but not necessarily the best, answer.

Problem characteristics (AI by Ritchie & Knight)

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

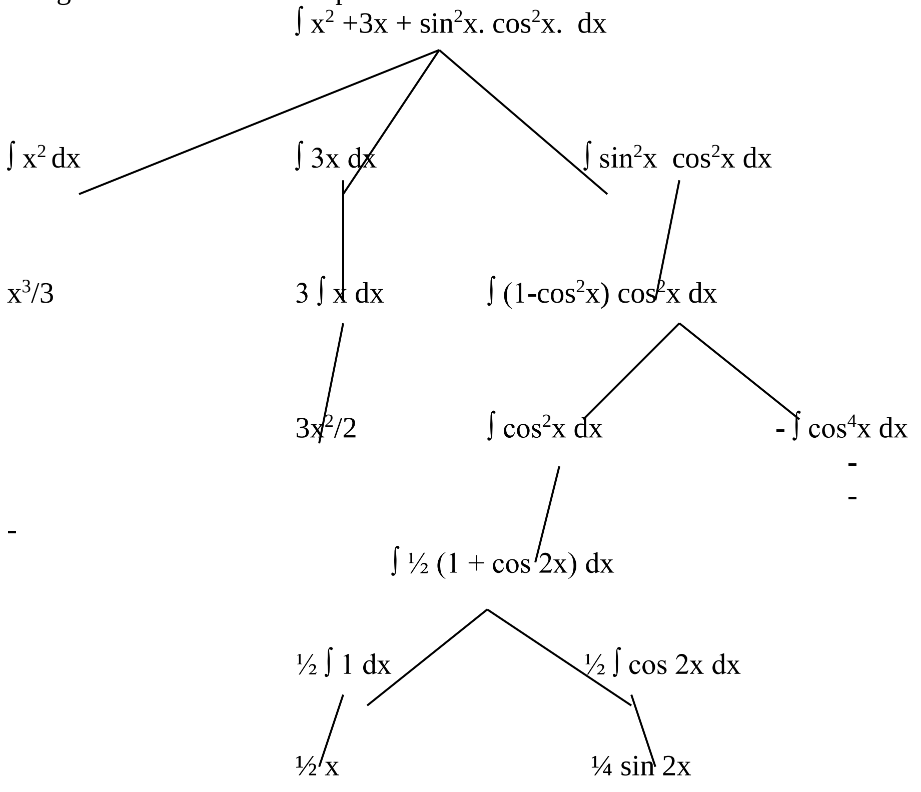
- Is the problem decomposable into a set of independent smaller or easier sub problems?
- Can solution steps be ignored or at least undone if they prove unwise?
- Is the problem's universe predictable?
- Is a good solution to the problem obvious without comparison to all other possible solutions?
- Is 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?

Is the problem decomposable?

Suppose we want to solve the problem of computing the expression

$$\int (x^2 + 3x + \text{Sin}^2 x \,.\, \text{Cos}^2 x)\, dx$$

We can solve this problem by breaking it down into 3 smaller problems, each of which can then solve by using a small collection of specific rules.


$\int x^2 + 3x + \sin^2 x.\ \cos^2 x.\ dx$
$\int x^2\, dx$
$\int 3x\, dx$
$\int \sin^2 x\ \cos^2 x\, dx$
$x^3/3$
$3 \int x\, dx$
$\int (1-\cos^2 x) \cos^2 x\, dx$
$3x^2/2$
$\int \cos^2 x\, dx$
$- \int \cos^4 x\, dx$
-
-
-
$\int \frac{1}{2} (1 + \cos 2x)\, dx$
$\frac{1}{2} \int 1\, dx$
$\frac{1}{2} \int \cos 2x\, dx$
$\frac{1}{2} x$
$\frac{1}{4} \sin 2x$

Can solution steps be ignored or undone?

Here we can divide problems into 3 classes.

Ignorable -- in which solution steps can be ignored. Eg: Theorem proving

Recoverable -- in which solution steps can be undone. Eg: 8-puzzle

Irrecoverable -- which solution steps cannot be undone.eg: Chess

Ignorable problems (eg. Theorem 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 theorem can be ignored. Then we can start from another rule. The former can be ignored.

Recoverable problems (eg. 8 puzzle problem)

Consider the 8 puzzle problem.

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.

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.

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.

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.

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.

	axioms
1. Marcus was a man.	1
4. All men are mortal.	4
8. Marcus is mortal.	1,4
3. Marcus was born in 40 A.D.	3
7. It is now 1991 A. D.	7
9. Marcus’ age is 1951 years.	3,7
6. no mortal lives longer than 150 years.	6
10. Marcus is dead.	8,6,9

OR

7. It is now 1991 A.D.	7
5. All Pompeans died in 79 A. D.	5
11. All Pompeans are dead now.	7,5
2. Marcus was a Pompoean.	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.

Boston – San Fransisco – Dallas—New York—Miami – Boston (8850)

Boston -- New York -- Miami -- Dallas -- San Fransisco – Boston (7750)

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.

Is the solution a state or a path?

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.

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 appropriate search 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.

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

1. All we want is to know that there is a proof.
2. 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 Strategies / Heuristic Search

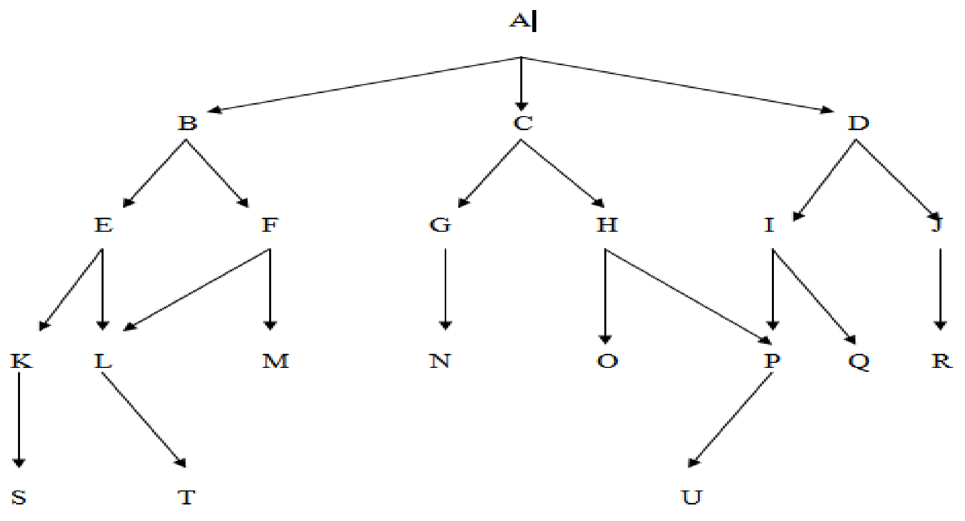
- Heuristics are criteria for deciding which among several alternatives be the most effective in order to achieve some goal.
- Heuristic is a technique that
  - Improves the efficiency of a search process possibly by sacrificing claims of systematicity and completeness.
  - It no longer guarantees to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

Basic search strategies

- ❖ DFS & BFS

Breadth first search (AI by Luger)

Consider the graph shown below.



States are labeled (A, B, C….).

Breadth first search explores the space in a level by level fashion. Only when there are no more states to be explored at a given level does the algorithm move on to the next level. A breadth first search of the above graph considers the states in the order A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U.

```
void  breadth_ first _ search ( )
{
        open = [ start ];
        closed = [ ];
        while ( open not empty )
        {
                Remove the leftmost state from open, call it X;
                if  X  is a goal,
                        then return SUCCESS;
                else
                {
                        Generate children of X;
                        Put X on closed;
                        Discard children of x, if already on open or closed;
                        Put remaining children on right end of open;
                }
        }
return FAIL;
}
```

A trace of the breadth first search on the graph appears below.

Open	closed
A	empty
BCD	A
CDEF	BA
DEFGH	CBA
EFGHIJ	DCBA
FGHIJKL	EDCBA
GHIJKLM	FEDCBA
HIJKLMN	GFEDCBA

.
And so on until open = [ ].

- BFS explores the space in level by level
- It moves to next level, only when there is no more stste to be visited at a given level.
- Its implemented using FIFO data structure. Open_Oueue
- If there is a solution, then BFS is guaranteed to find it. If there are multiple solution then minimal solution will be found.

Depth first search

Dept first search goes deeper in to the search space whenever this is possible. Consider the same graph. Depth first search examines the states in the graph in the order A, B, E, K, S, L, T, F, M, C, G, N, H, O, P, U, D, I, Q, J, R.In depth first search, the descendent states are added and removed from the left end of open. That is, open is maintained as a stack (LIFO) data structure.

```
void depth_first_search ()
{
        open = [start];
        closed = [ ];
        while (open not empty )
        {
                remove leftmost state from open, call it X;
                if X is a goal state
                        then return SUCCESS;
                else
                {
                        generate children of X;
                        put X on closed;
                        discard children of X, if already on open or closed;
                        put remaining children on left end of open;
                }
        }
return FAIL;
}
```

A trace of depth first search on the above graph is shown below.

open	closed
A	empty
B C D	A
E F C D	B A
K L F C D	E B A
S L F C D	K E B A
L F C D	S K E B A
T F C D	L S K E B A
F C D	T L S K E B A
M C D	F T L S K E B A
C D	M F T L S K E B A
G H D	C M F T L S K E B A

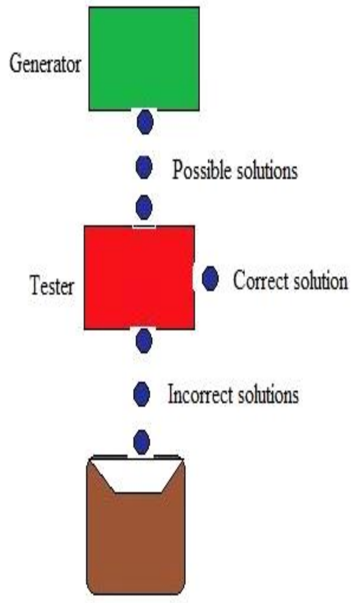
And so on until open = [ ];

- Open is implemented using stack LIFO
- DFS goes deeper in search space whenever this is possible.
- It requires less memory since only one path on the current path are stored.
- DFS may find a solution, that maynot be the minimal solution. Ie, its not guaranteed to find the best solution.

ALGORITHM GENERATE AND TEST

Generate and test is the simplest of all the approaches we discuss.

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, path from the start state.



Steps:

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 step1.

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.

<u>Systematic Generate and Test</u>

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

DFS 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.

HILL CLIMBING

Hill climbing strategies expand the current state in the search and evaluate its children. The best child is selected for further expansion; neither its siblings nor its parent is retained. Search halts when it reaches a state that is better than any of its children. Hill climbing is named for the strategy that might be used by an eager, but blind mountain climber: go uphill along the steepest possible path until you can go no farther. Because it keeps no history, the algorithm cannot recover from failures of its strategy. In pure generate and test procedure, the test function respond with only a yes or no. But if the test function is augmented with a heuristic function that provides an estimate how close the given state is to a goal state.

There are three various strategies for hill climbing. They are

- Simple hill climbing,
- Steepest ascent hill climbing and
- Simulated annealing.

Simple hill climbing

The simplest way to implement hill climbing is as follows.

Algorithm

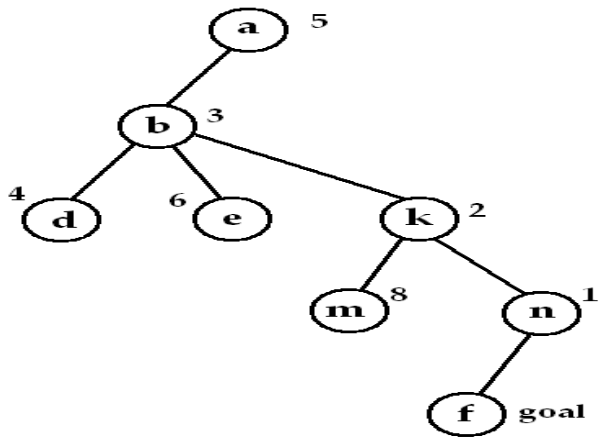
1. Evaluate the initial state. If it is also a goal state, return it and quit. Otherwise continue with the initial state as the current state.
2. Loop until a solution is found or until there are no new operators left to be applied in the current state:
    a. Select an operator that has not yet been applied to the current state and apply it to produce a new state.
    b. Evaluate the new state,

i. If it is a goal state, then return it and quit.
ii. If it is not a goal state, but it is better than the current state, then make it the current state.
iii. If it is not better than the current state, then continue in the loop.

Example:

A problem is given. Given the start state as 'a' and the goal state as 'f'.

Suppose we have a heuristic function h (n) for evaluating the states. Assume that a lower value of heuristic function indicates a better state.



Here *a* has an evaluation value of 5. a is set as the current state.

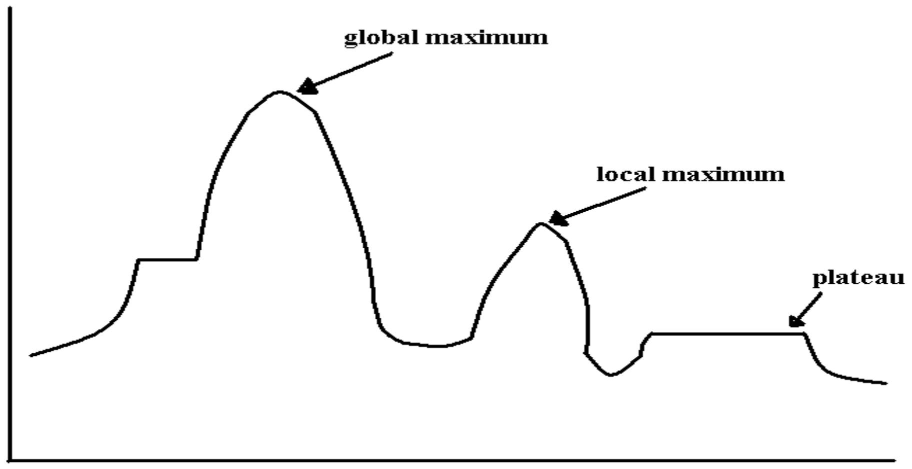
Generate a successor of a. Here it is b. The value of b is 3. It is less than that of a. that means b is better than the current state a. So set b as the new current state.

Generate a successor of b. it is d. D has a value of 4. It is not better than the current state b (3). So generate another successor of b. it is e. It has a value of 6. It is not better than the current state b (3). Then generate another successor of b. We get k. It has an evaluation value of 2. k is better than the current state b. (3). So set k as the new current state. Now start hill climbing from k. Proceed with this, we may get the goal state f.
Both basic and steepest ascent hill climbing may fail to find a solution. Either algorithm may stop not by finding a goal state but by getting to a state from which no better states can be generated. This will happen if the program has reached a local maximum, a plateau or a ridge.

(a) A "local maximum" which is a state better than all its neighbors, but is not better than some other states farther away. Local maxim sometimes occurs within sight of a solution. In such cases they are called "Foothills".

(b) A "plateau" which is a flat area of the search space, in which neighboring states have the same value. On a plateau, it is not possible to determine the best direction in which to move by making local comparisons.

(c) A "ridge" which is an area in the search that is higher than the surrounding areas, but cannot be searched in a simple move. flat like a plateau, but with drop-offs to the sides; steps to the North, East, South and West may go down, but a step to the NW may go up. **It may require unreasonable length of time towards a better position since it takes tiny zig-zag steps.**



To overcome these problems we can

- Back track to some earlier nodes and try a different direction. This is a good way of dealing with local maxim.

- Make a big jump and some direction to a new area in the search. This can be done by applying two more rules of the same rule several times, before testing. This is a good strategy is dealing with plate and ridges.

- Apply 2 or more rules before doing the test. This corresponds to moving in several directions at once. This is a good way for dealing with ridges.

Hill climbing becomes inefficient in large problem spaces, and when combinatorial explosion occurs. But it is a useful when combined with other methods.

Steepest ascent hill climbing

A useful variation on simple hill climbing considers all the moves from the current state and selects the best one as the next state.

Algorithm

1. Evaluate the initial state. If it is also a goal state, then return it and quit. Otherwise, continue with the initial state as the current state.
2. Loop until a solution is found or until a complete iteration produces no change to current state.
    a. Let SUCC be a state such that any possible successor of the current will be better than SUCC.
    b. For each operator that applies to the current state, do:
        i. Apply the operator and generate a new state.

   ii. Evaluate the new state. If it is a goal state, then return it and quit. If not, compare it to SUCC. If it is better, then set SUCC to this state. If it is not better, leave SUCC alone.

c. if the SUCC is better than current state, then set current state to SUCC.

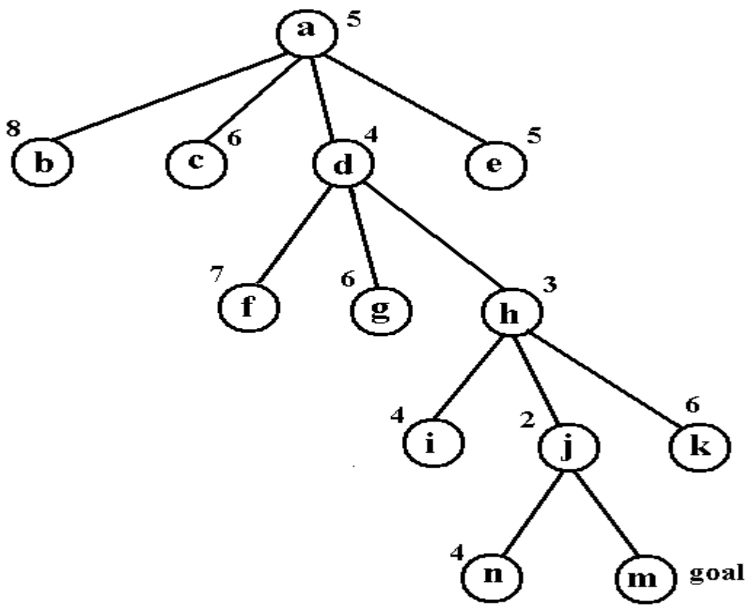
An example is shown below.

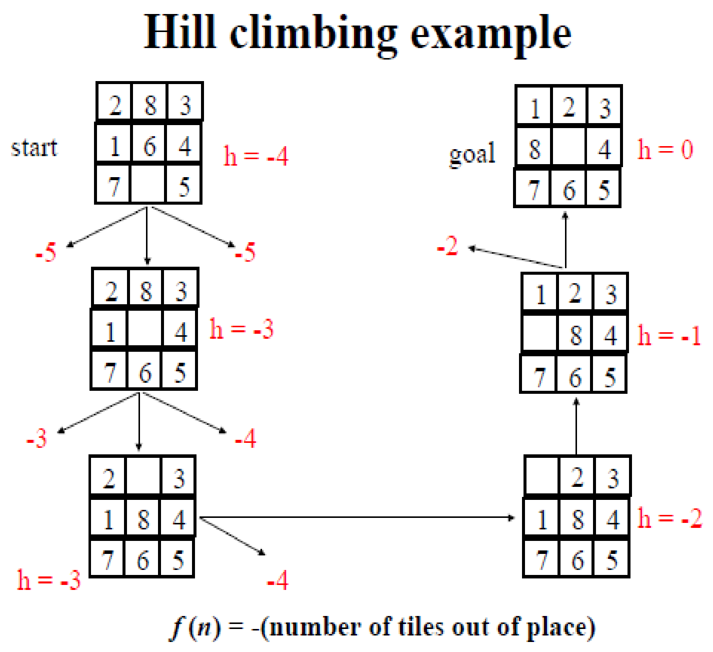
Consider a problem. Initial state is given as a. the final state is m.

Let h(n) be a heuristic function for evaluating the states.

In this, all the child states (successors) of a are generated first. The state d has the least value of heuristic function. So d is the best among the successors of a. then d is better than the current state a. so make d as the new current state. Then start hill climbing from d.

In simple hill climbing, the first closer node is chosen, whereas in steepest ascent hill climbing all successors are compared and the closest to the solution is chosen. Both forms fail if there is no closer node, which may happen if there are local maxima in the search space which are not solutions.





Hill climbing is not always very effective. It is practically unsuited to problems where the value of the heuristic function drops off suddenly as you move away from a solution. Hill climbing is a local method, by which we mean that it decides what to do next by looking only at the "immediate" consequences of its choice rather than exploring all the consequences.

SIMULATED ANNEALING

We have seen that hill climbing never makes a downhill move. As a result, it can stuck on a local maximum. In contrast, a purely random walk- that is, moving to a successor chosen uniformly at random from the set of successors- is complete, but extremely inefficient.

Simulated annealing combines hill climbing with a random walk. As a result, it yields efficiency and completeness. Simulated annealing is a variation of hill climbing in which, at the beginning of the process, some downhill moves may be made. The idea is to do enough exploration of the whole space early on so that the final solution is relatively insensitive to the starting state. This should lower the chances of getting caught at a local maximum, a plateau or a ridge.

In metallurgy, annealing is the process in which metals are melted and then gradually cooled until some solid state is reached. It is used to tamper or harden metals and glass by heating them to a high temperature and then gradually cooling them, thus allowing the material to coalesce in to a low energy crystalline state.

Physical substances usually move from higher energy configurations to lower ones. But there is some probability that a transition to a higher energy state will occur. This probability is given by the function

$\mathbf{P = e^{-\Delta E / kT}}$

Where ΔE is positive change in the energy level, T is the temperature and k is Boltzmann's constant.

Physical annealing has some properties. The probability of a large uphill move is lower than the probability of a small one. Also, the probability that an uphill move will be made decreases as the temperature decreases. Large uphill moves may occur early on, but as the process progresses, only relatively small upward moves are allowed until finally the process converges to a local maximum configuration.

If cooling occurs so rapidly, stable regions of high energy will form. If however, a slower schedule is used, a uniform crystalline structure which corresponds to a global minimum is more likely to develop. If the schedule is too slow, time is wasted. The rate at which system is called annealing schedule.

These properties of physical annealing can be used to define the process of simulated annealing. In this process, ΔE represents change in the value of heuristic evaluation function instead of change in energy level. k can be integrated in to T. hence we use the revised probability formula.

$$\mathbf{P' = e^{-\Delta E / T}}$$

Thermodynamic Simulation	**Combinatorial Optimisation**
System States	Feasible Solutions
Energy	Cost
Change of State	Neighbouring Solutions
Temperature	Control Parameter
Frozen State	Heuristic Solution

Algorithm

1. Evaluate the initial state. If it is also a goal state, then return it and quit. Otherwise, continue with the initial state as the current state.
2. Initialize BEST-SO-FAR to the current state.
3. Initialize T according to the annealing schedule.
4. Loop until a solution is found or until there are no new operators left to be applied in the current state.
    a. Select an operator that has not yet been applied to the current state and apply it to produce a new state.
    b. Evaluate the new state. Compute

       ΔE = (value of current) – (value of new state)

        - If the new state is a goal state, then return it and quit.
        - If it is not a goal state, but is better than the current state, then make it the current state. Also set BEST-SO-FAR to this new state.
        - If it is not better than the current state, then make it the current state with probability p' as defined above. This step is usually implemented by invoking a random number generator to produce a number in the range [0, 1]. If that number is less than p', then the move is accepted. Otherwise, do nothing.

    c. Revise T as necessary according to the annealing schedule.

5. Return BEST- SO- FAR as the answer.

To implement this algorithm, it is necessary to maintain an annealing schedule. That is first we must decide the initial value to be used for temperature. The second criterion is to decide when the temperature of the system should be reduced. The third is the amount by which the temperature will be reduced each time it is changed.

In this algorithm, instead of picking the best move, it picks a random move. If the move improves the situation, it is always accepted. Otherwise the algorithm accepts the move with some probability less than 1.

The probability decreases

- **Exponentially with the amount ΔE by which the evaluation is worsened.**
- **As the temperature T goes down.**

Thus bad moves are more likely to be allowed at the start when the temperature is high, and they become more unlikely as T decreases.

Simulated annealing was first used extensively to solve VLSI layout problems. The algorithm for simulated annealing is slightly different from the simple hill climbing.

- The annealing schedule must be maintained.
- Moves to worse states may be accepted
- It maintains the current state as well as best_state_found so far. Then if the final state is worse than that earlier state, the earlier state is still available.

Lists, Tuples, and Dictionaries

- **Lists** are what they seem - a list of values. Each one of them is numbered, starting from zero - the first one is numbered zero, the second 1, the third 2, etc. You can remove values from the list, and add new values to the end. Example: Your many cats' names.
- **Tuples** are just like lists, but you can't change their values. The values that you give it first up, are the values that you are stuck with for the rest of the program. Again, each value is numbered starting from zero, for easy reference. Example: the names of the months of the year.
- **Dictionaries** are similar to what their name suggests - a dictionary. In a dictionary, you have an 'index' of words, and for each of them a definition. In python, the word is called a 'key', and the definition a 'value'. The values in a dictionary aren't numbered - tare similar to what their name suggests - a dictionary. In a dictionary, you have an 'index' of words, and for each of them a definition. In python, the word is called a 'key', and the definition a 'value'. The values in a dictionary aren't numbered - they aren't in any specific order, either - the key does the same thing. You can add, remove, and modify the values in dictionaries. Example: telephone book.