AI Techniques

By

Assist. prof. dr. Israa AbdulAmeer

Introduction to Artificial Intelligence and Search Algorithms

Introduction to Artificial Intelligence

- **What is Intelligence?**
- Intelligence is the ability to learn about, to learn from, to understand about, and interact with one's environment.
-
- **What is Artificial Intelligence (AI)?**
- **A.I:-** Is simply a way of making a computer think.
- **A.I:-** Is the part of computer science concerned with designing intelligent computer system that exhibit the characteristic associated with intelligent in human behavior.
- **This requires many processes:-**
- **1- Learning: -** acquiring the knowledge and rules that used these knowledge.
- **2- Reasoning:-** Used the previous rules to access to nearly reasoning or fixed reasoning.

Introduction to Artificial Intelligence

- **A.I Principles:-**
- 1- The data structures used in knowledge representation.
- 2- The algorithms needed to apply that knowledge.
- 3- The language and programming techniques used their implementation.

- **What are the goals of AI research?**
- The central problems (or goals) of AI research include reasoning, knowledge, planning, learning, natural language processing (communication), perception and the ability to move and manipulate objects.

- **What is problem reduction meaning?**
- Problem Reduction means that there is a hard problem may be one that can be reduced to a number of simple problems. Once each of the simple problems is solved, then the hard problem has been solved.
-

Applications of AI

- Game playing
- Speech recognition
- Understanding natural language
- Computer vision
- Expert systems
- Heuristic classification

Characteristics of AI

- • High societal impact (affect billions of people)
- • Diverse (language, vision, robotics)
- • Complex (really hard)

Search Algorithms

To successfully design and implement search algorithms, a programmer must be able to analyze and predict their behavior.

Many questions needed to be answered by the algorithm these include:

- Is the problem solver guaranteed to find a solution?
- Will the problem solver always terminate, or can it become caught in an infinite loop?
- When a solution is found, is it guaranteed to be optimal?
- What is the complexity of the search process in terms of time usage? Space search?
- How can the interpreter be designed to most effectively utilize a representation language?

State Space Search

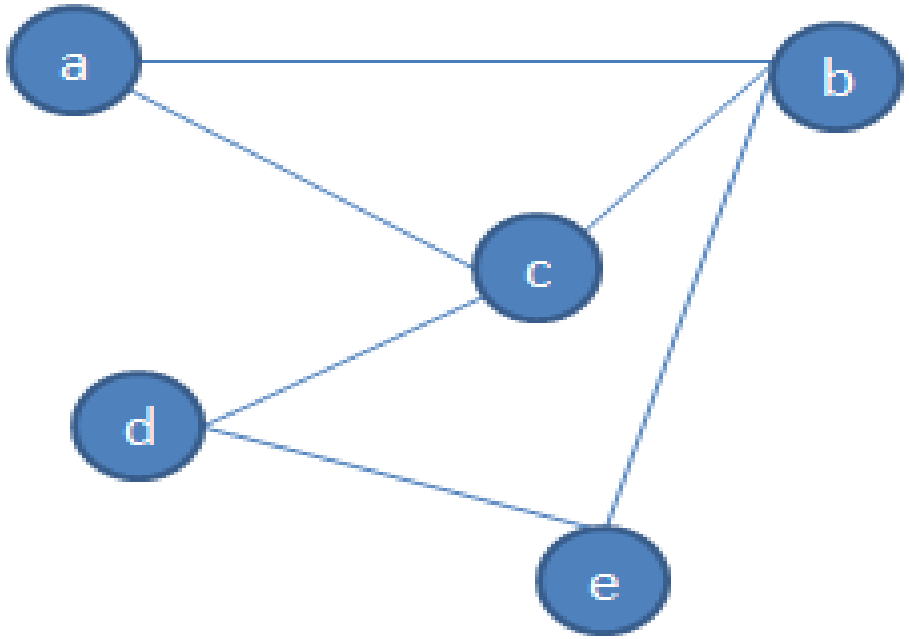
- The theory of state space search is our primary tool for answering these questions, by representing a problem as state space graph, we can use graph theory to analyze the structure and complexity of both the problem and procedures used to solve it.

Graph Theory:-

- A graph consists of a set of a nodes and a set of arcs or links connecting pairs of nodes. The domain of state space search, the nodes are interpreted to be stated in problem solving process, and the arcs are taken to be transitions between states.

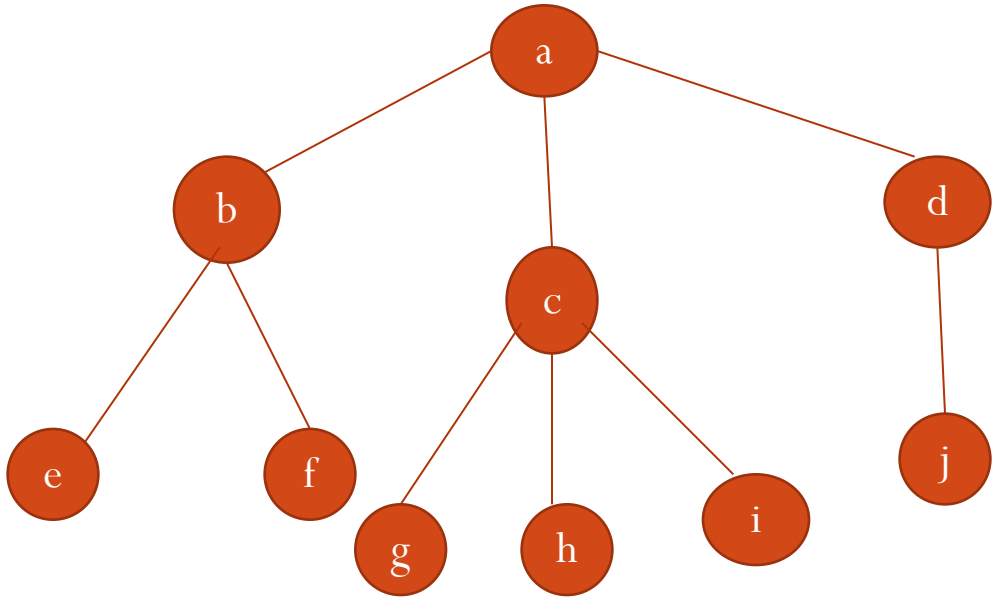
Graph theory is our best tool for reasoning about the structure of objects and relations.

Graph Theory



Nodes={a,b,c,d,e}

Arcs={(a,b), (a,c),(b,c),(b,e),(d,e),(d,c),(e,d)}



Nodes={a,b,c,d,e,f,g,h,i,j}

Arcs={(a,b),(a,c),(a,d),(b,e),(b,f),(c,g),(c,h),(c,i),(d,j)}

State Space Representation

A state space is represented by four tuple [N,A,S,GD], where:-

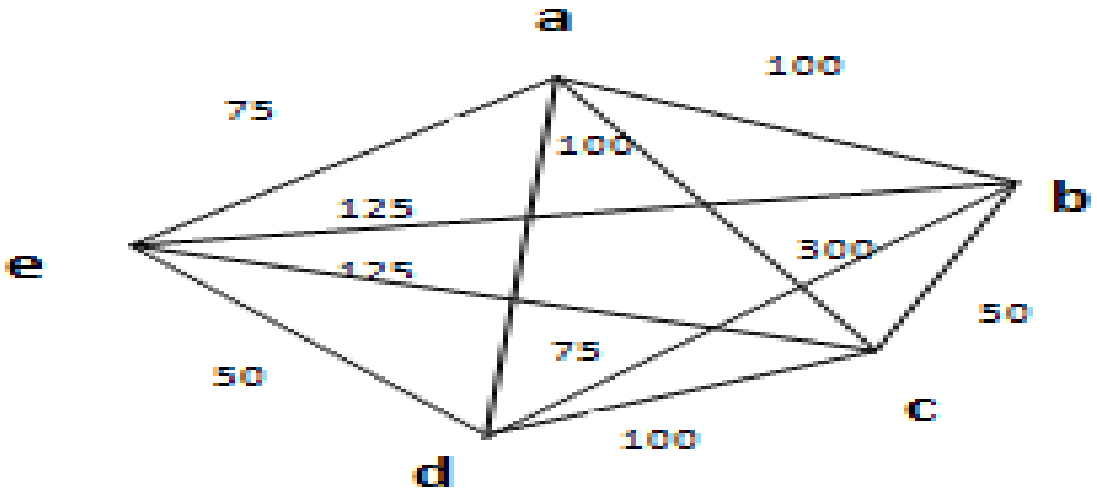
- **N** is a set of nodes or states of the graph. These correspond to the states in a problem –solving process.
- **A** is the set of arcs between the nodes. These correspond to the steps in a problem –solving process.
- **S** a nonempty subset of N , contains the start state of the problem.

- **G** a nonempty subset of N contains the goal state of the problem.

- **A solution path:- Is a path through this graph from a node S to a node in GD.**

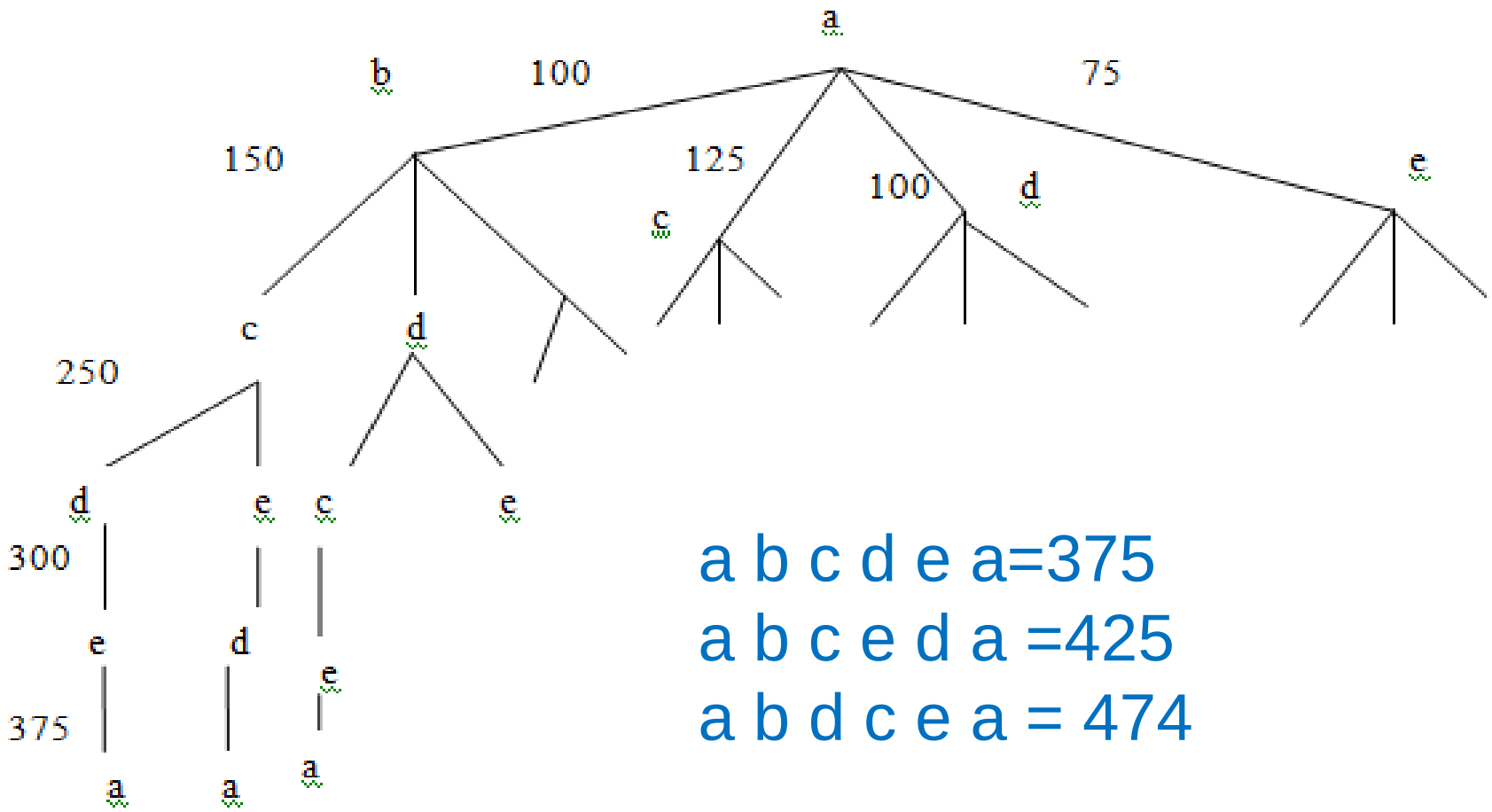
Example:- Traveling Salesman Problem

- Starting at A , find the shortest path through all the cities , visiting each city exactly once returning to A.



The complexity of exhaustive search in the traveling Salesman is **(N-1)!**, where N is the No. of cities in the graph. There are several technique that reduce the search complexity.

1- Branch and Bound Algorithm



a b c d e a=375

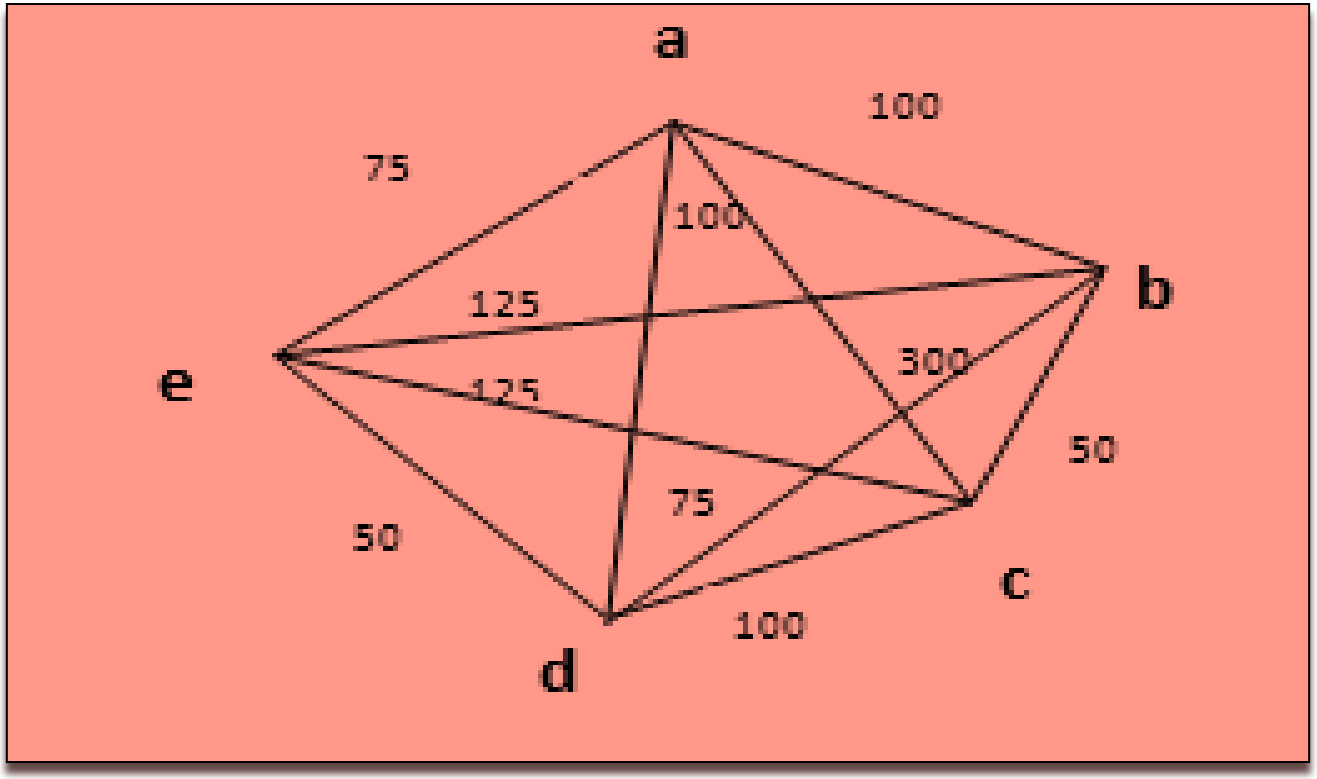
a b c e d a =425

a b d c e a = 474

.....

2- Nearest Neighbor Heuristic

- At each stage of the circuit, go to the nearest unvisited city. This strategy reduces the complexity to N, so it is highly efficient, but it is not guaranteed to find the shortest path, as the following example:



Cost of Nearest neighbor path is a e d b c a=550

Is not the shortest path , the comparatively high cost of arc (C,A) defeated the heuristic

Blind Search

This type of search takes all nodes of tree in specific order until it reaches to goal. The order can be in breath and the strategy will be called breadth – first – search, or in depth and the strategy will be called depth first search.

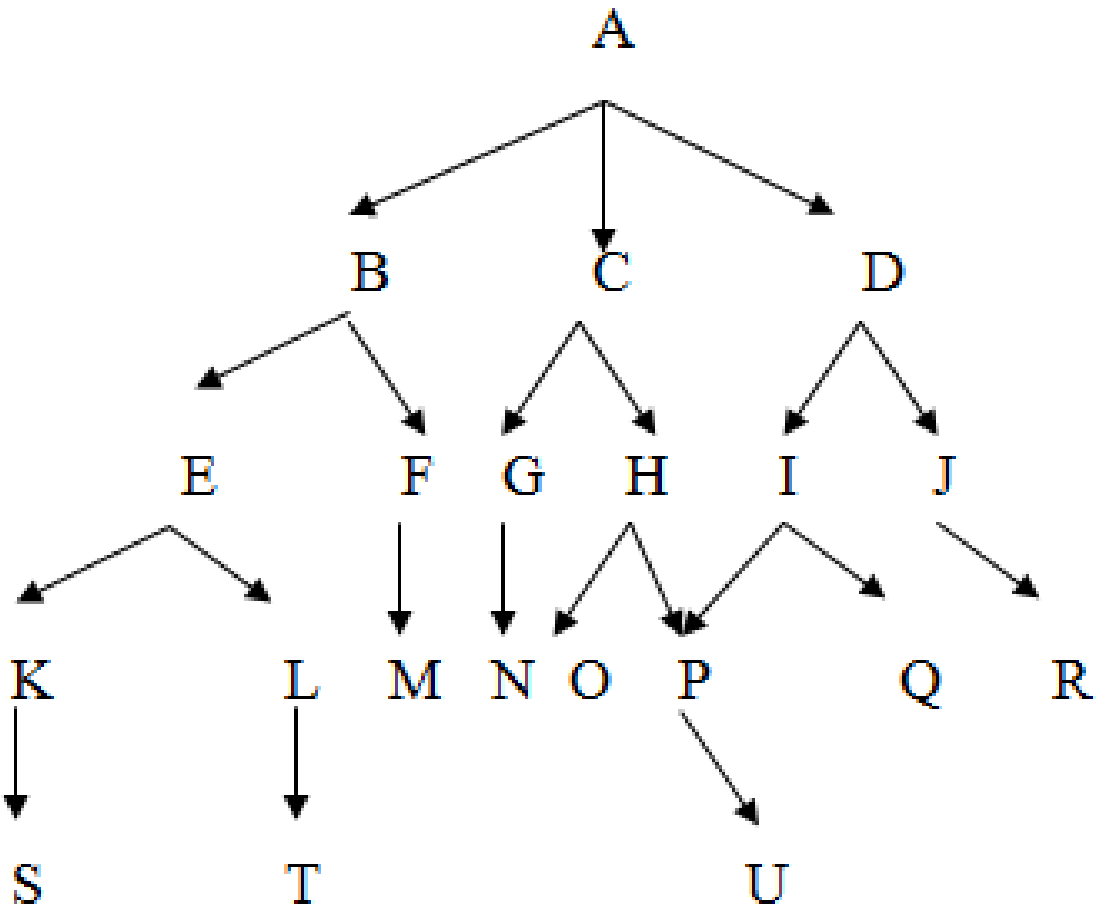
1- Breadth – First – Search

In breadth –first search, when a state is examined, all of its siblings are examined before any of its children. The space is searched level-by-level, proceeding all the way across one level before doing down to the next level.

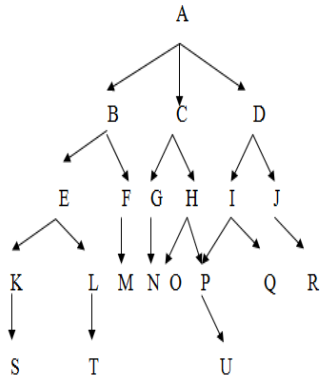
Breadth – first – search Algorithm

- *Begin*
- *Open: = [start];*
- *Closed: = [ ];*
- *While open ≠ [ ] do*
- *Begin*
- *Remove left most state from open, call it x;*
- *If x is a goal the return (success)*
- *Else*
- *Begin*
- *Generate children of x;*
- *Put x on closed;*
- *Eliminate children of x on open or closed;* *(Removing the repeated child or node)*
- *Put remaining children on right end of open*
- *End*
- *End*
- *Return (failure)*
- *End.*

Breadth – First – Search



Breadth – First – Search



- 1 – Open= [A]; closed = [ ].
- 2 – Open= [B, C, D]; closed = [A]
- 3 – Open= [C, D, E, F]; closed = [B,
- 4 – Open= [D, E, F, G, H]; closed = [C, B, A].
- 5 – Open= [E, F, G, H, I, J]; closed = [D, C, B, A].
- 6 – Open= [F, G, H, I, J, K, L]; closed = [E, D, C, B, A].
- 7 – Open= [G, H, I, J, K, L, M]; closed = [F, E, D, C, B, A].
- 8 – Open= [H, I, J, K, L, M, N,]; closed = [G, F, E, D, C, B, A].
- 9 – and so on until either U is found or open = [ ].

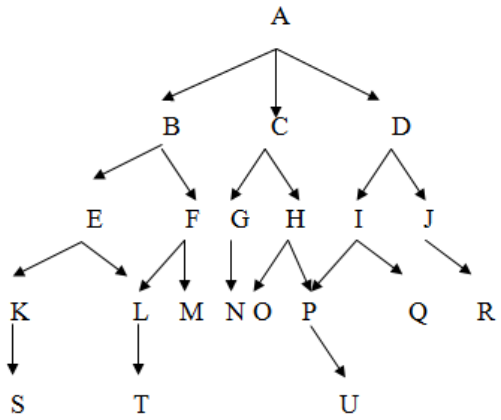
2- Depth – first – search

- In depth – first – search, when a state is examined, all of its children and their descendants are examined before any of its siblings.
- Depth – first search goes deeper in to the search space when ever this is possible only when no further descendants of a state can found.

Depth – first – search Algorithm

- *Begin*
- *Open: = [start];*
- *Closed: = [ ];*
- *While open ≠ [ ] do*
- *Remove leftmost state from open, call it x;*
- *If x is a goal then return (success)*
- *Else begin*
- *Generate children of x;*
- *Put x on closed;*
- *Eliminate children of x on open or closed; (Removing the repeated child or node)*
- *put remaining children on left end of open end*
- *End;*
- *Return (failure)*
- *End.*

Depth – first – search



- 1 – Open= [A]; closed = [ ].
- 2 – Open= [B, C, D]; closed = [A].
- 3 – Open= [E, F, C, D]; closed = [B, A].
- 4 – Open= [K, L, F, , D]; closed = [E, B, A].
- 5 – Open= [S, L, F, C, D]; closed = [K, E, B, A].
- 6 – Open= [L, F, C, D]; closed = [S, K, E, B, A].
- 7 – Open= [T, F, C, D]; closed = [L, S, K, E, B, A].
- 8 – Open= [F, C, D,]; closed = [T, L, S, K, E, B, A].
- 9– Open= [M, C, D] as L is already on; closed = [F, T, L, S, K, E, B, A].
- 10 – and so on until either U is found or open = [ ].

Heuristic Search

A heuristic is a method that might not always find the best solution but is guaranteed to find a good solution in reasonable time. By sacrificing completeness it increases efficiency. Heuristic search is useful in solving problems which:-

- Could not be solved any other way.
- Solution take an infinite time or very long time to compute.

Heuristic search methods generate and test algorithm , from these methods are:-

1- Hill Climbing.

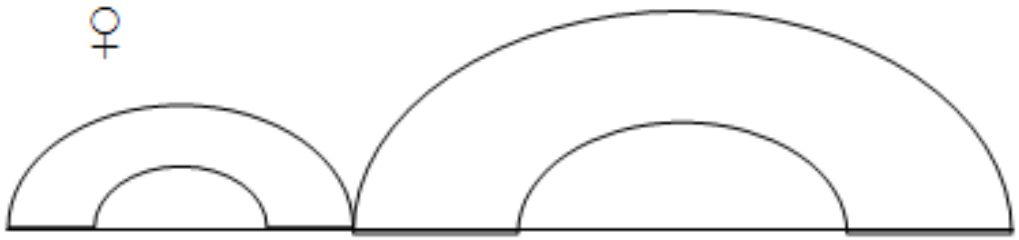
2- Best-First Search.

3- A and A* algorithm.

1-Hill Climbing

- The idea here is that, you don’t keep the big list of states around you just keep track of the one state you are considering, and the path that got you there from the initial state. At every state you choose the state leads you closer to the goal (according to the heuristic estimate), and continue from there.
- The name “Hill Climbing” comes from the idea that you are trying to find the top of a hill , and you go in the direction that is up from wherever you are. This technique often works, but since it only uses local information.

Hill Climbing



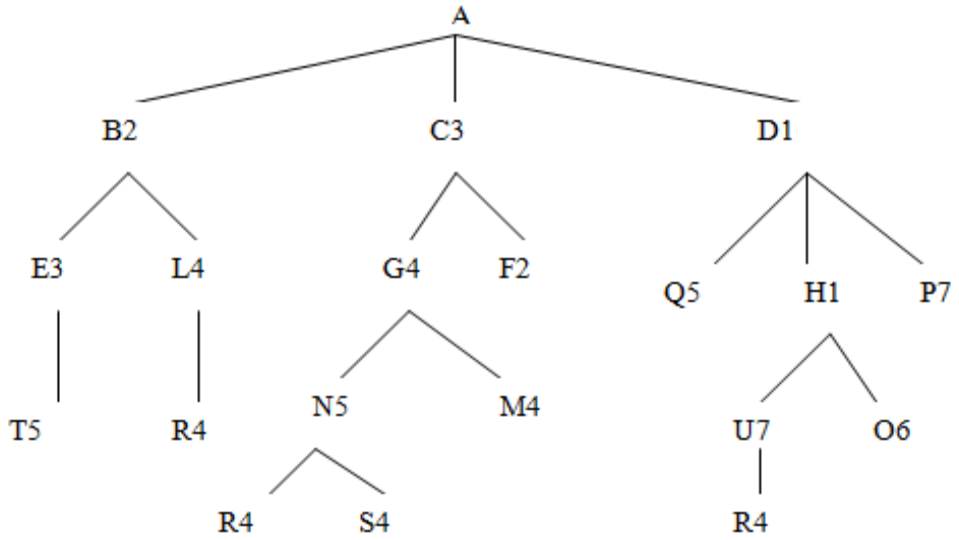
The smaller peak is an example of a “local maxima” in a domain (or “local minima”). Hill climbing search works well (and is fast, takes a little memory) if an accurate heuristic measure is available in the domain, and there are now local maxima.

Hill Climbing Algorithm

-
- *Begin*
- *Cs=start state;*
- *Open=[start];*
- *Stop=false;*
- *Path=[start];*
- *While (not stop) do*
- *{*
- *if (cs=goal) then*
- *return (path);*
- *generate all children of cs and put it into open*
- *if (open=[]) then*
- *stop=true*
- *else*

- *{*
- *x:= cs;*
- *for each state in open do*
- *{*
- *compute the heuristic value of y (h(y));*
- *if y is better than x then*
- *x=y*
- *}*
- *if x is better than cs then*
- *cs=x*
- *else*
- *stop =true;*
- *}*
- *}*
- *return failure;*
- *}*

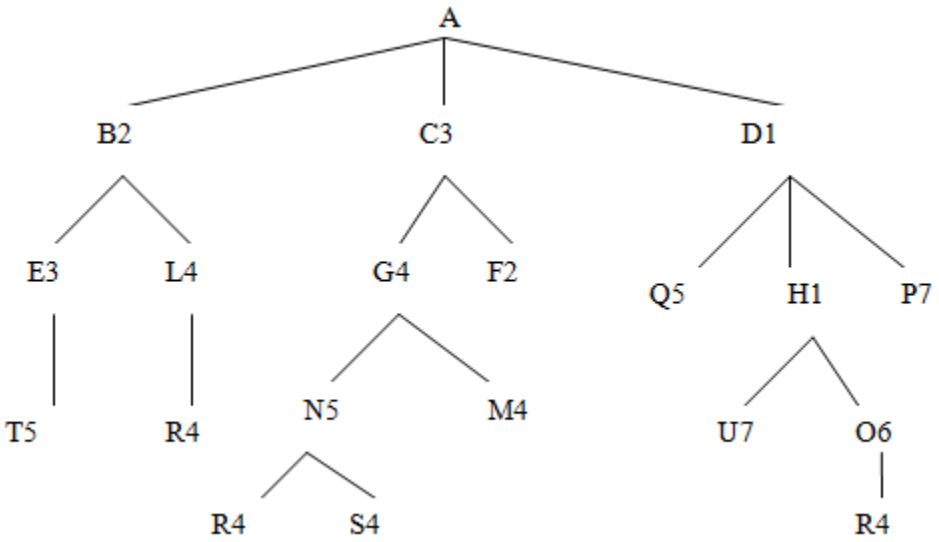
A trace of Hill Climbing Algorithm
LOCAL MAXIMA



Open=[A]	Closed=[]	A
Open=[C3,B2,D1]	Closed=[A]	C3
Open=[G4,F2]	Closed=[A, C3]	G4
Open=[N5,M4]	Closed=[A,C3,G4]	N5
Open=[R4,S4]	closed=[A,C3,G4,N5]	R4

The solution path is: A-C3-G4-N5-R4

A trace of Hill Climbing Algorithm
LOCAL MINIMA

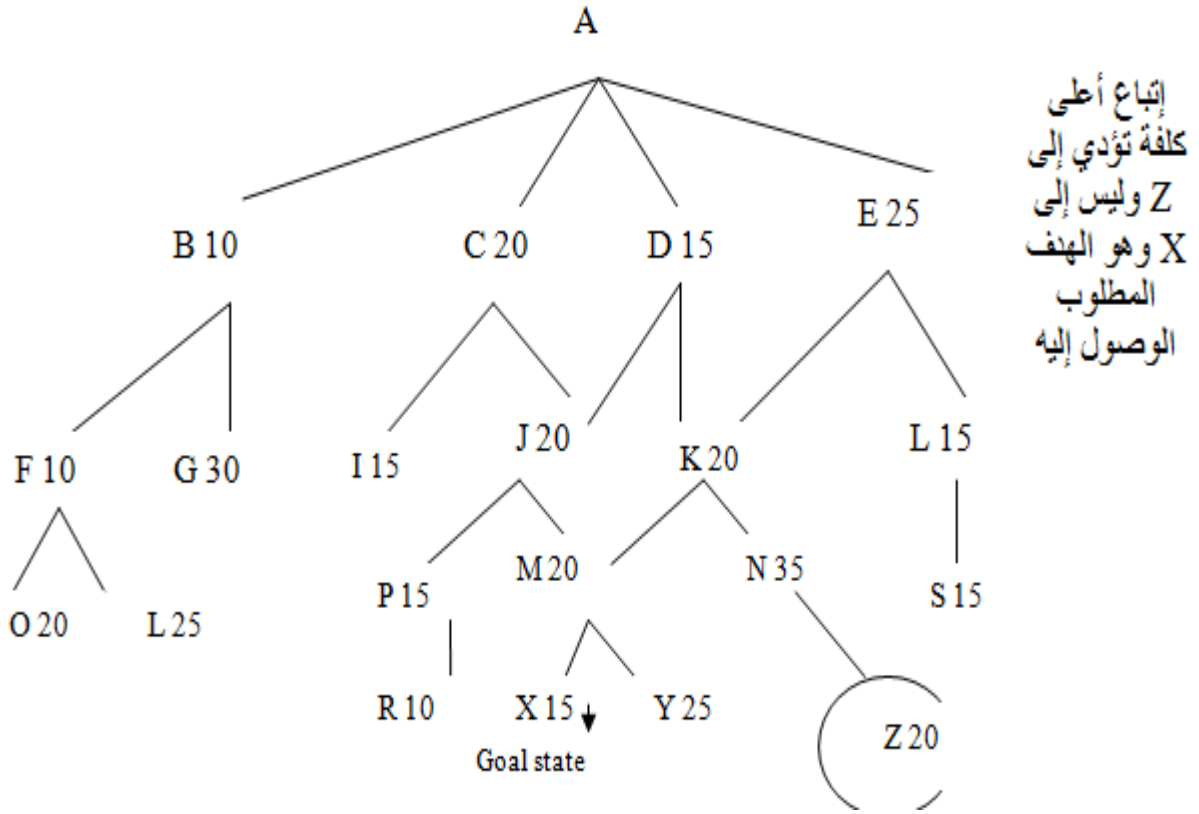


Open=[A]	Closed=[]	A
Open=[D1,B2,C3]	Closed=[A]	D1
Open=[H1,Q5,P7]	Closed=[A, D1]	H1
Open=[O6,U7]	Closed=[A,D1,H1]	O6
Open=[R4]	closed=[A,D1,H1,O6]	R4
	Closed=[A,D1,H1,O6,R4]	

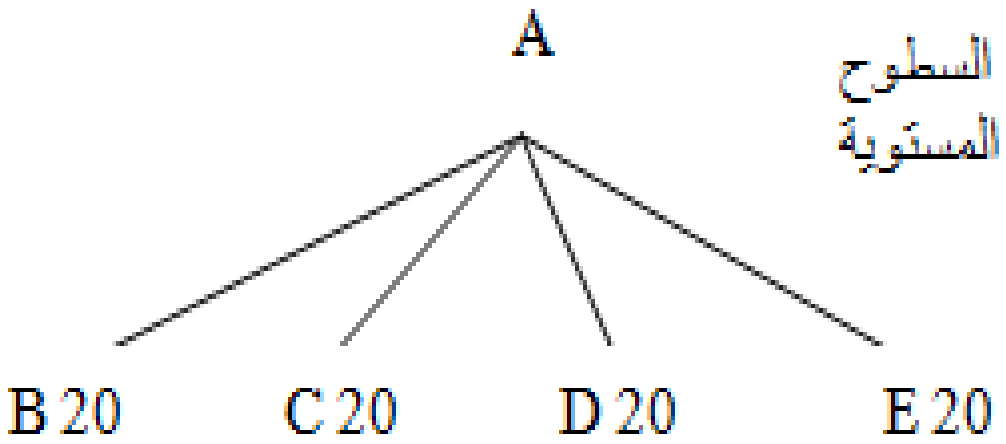
The solution path is: A-D1-H1-O6-R4

Hill climbing Problems:-

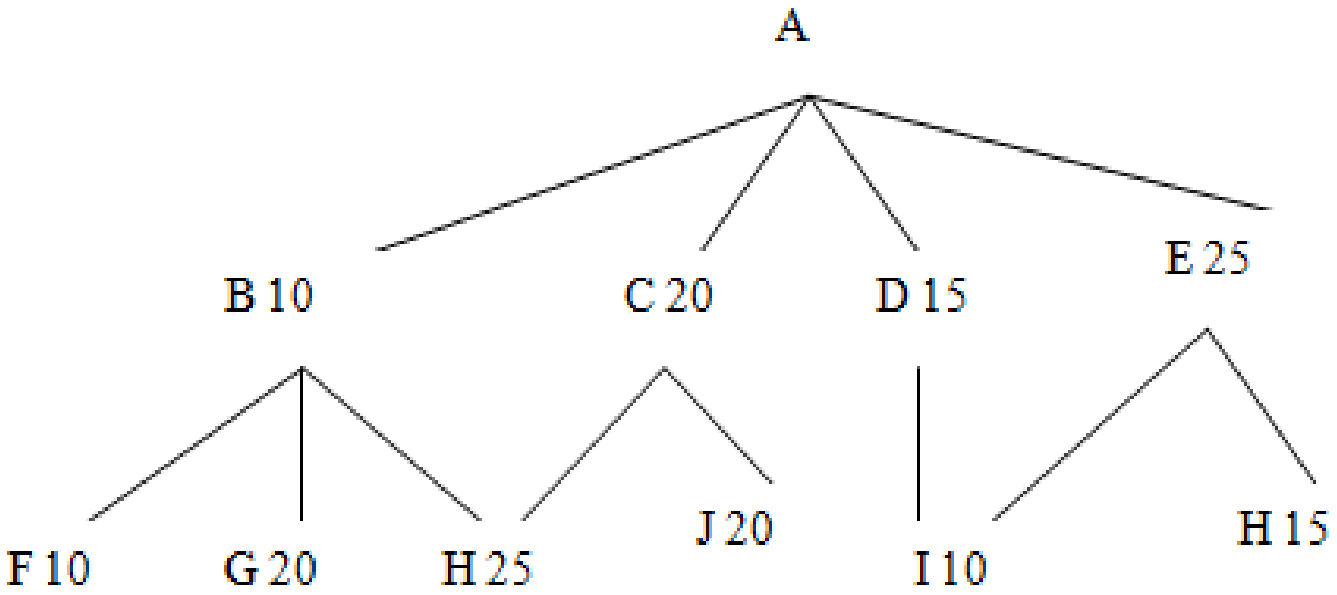
1- a local maxima: Is a state that is better than all of its neighbors but is not better than some other states.



A Plateau: Is a flat area of the search space in which a number of states have the same best value , on plateau its not possible to determine the best direction in which to move.

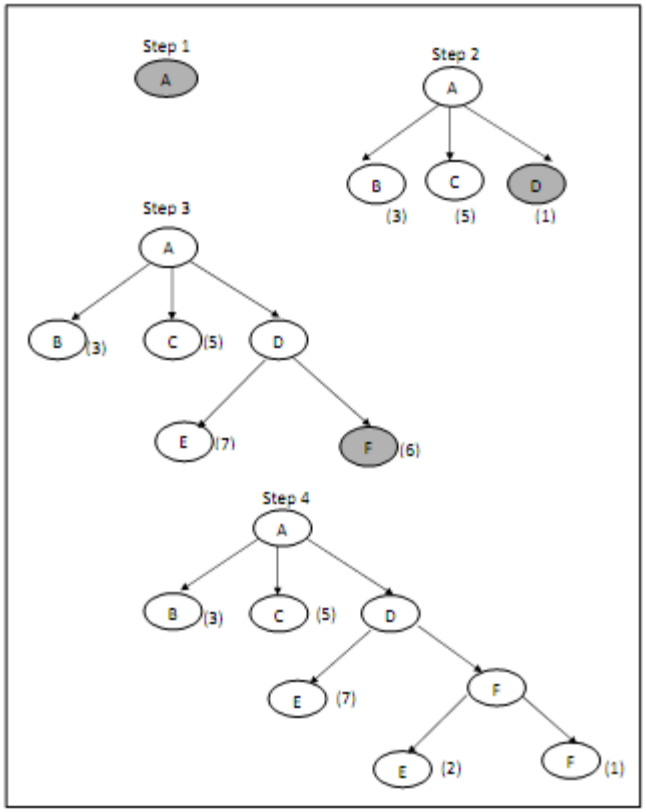


A ridge: Is an area of the search space that is higher than surrounding areas , but that can not be traversed by a single move in any one direction.



سلسلة الإنخفاضات و الإرتفاعات

The figure bellow illustrates the hill climbing steps al gorithm as it described in tree data structure.



2- Best-First-Search

- Best First search is away of combining the advantages of both depth-first and breadth-first s earch into a single method.

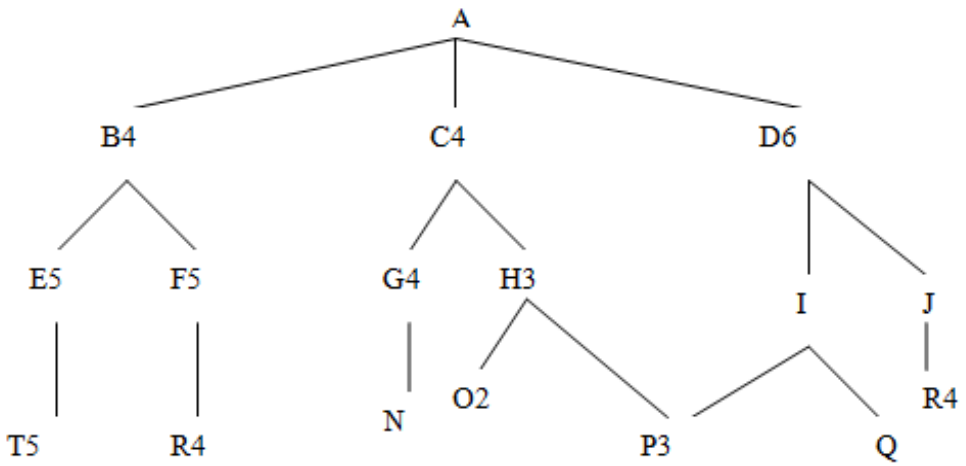
- In Best-First search, the search space is evaluated according to a heuristic function. Nodes yet to be evaluated are kept on an OPEN list and those that have already been evaluated are stored on a CLOSED list. The OPEN list is represented as a priority queue, such that unvisited nodes can be queued in order of their evaluation function. The evaluation function f(n) is made from only the heuristic function (h(n)) as: $f(n) = h(n)$.

Best-First-Search Algorithm

- {
- *Open:=[start];*
- *Closed:=[];*
- *While open ≠ [] do*
- {
- *Remove the leftmost from open, call it x;*
- *If x= goal then*
- *Return the path from start to x*
- *Else*
- {
- *Generate children of x;*
- *For each child of x do*
- *Do case*
- *The child is not already on open or closed;*
- *{ assign a heuristic value to the child state ;*
- *Add the child state to open;*
- }

- *The child is already on open:*
- *If the child was reached along a shorter path than the state currently on open then give the state on open this shorter path value.*
- *The child is already on closed:*
- *If the child was reached along a shorter path than the state currently on open then*
- *{*
- *Give the state on closed this shorter path value*
- *Move this state from closed to open*
- *}*
- *}*
- *Put x on closed;*
- *Re-order state on open according to heuristic (best value first)*
- *}*
- *Return (failure);*
- *}*

2- Best-First-Search



Open	Closed
Open=[A5]	Closed=[]
Open=[B4,C4,D6]	Closed=[A5]
Open=[C4,E5,F5,D6]	Closed=[B4,A5]
Open=[H3,G4, E5,F5,D6]	Closed=[C4,B4,A5]
Open=[O2,P3,G4,E5,F5,D6]	Closed=[H3, C4,B4,A5]
Open=[P3,G4,E5,F5,D6]	Closed=[O2,H3, C4,B4,A5]
Open=[G4,E5,F5,D6]	Closed=[P3,O2,H3,C4,B4,A5]

The solution path is: A5 - B4 - C4 - H3 – O2- P3

A- Algorithm

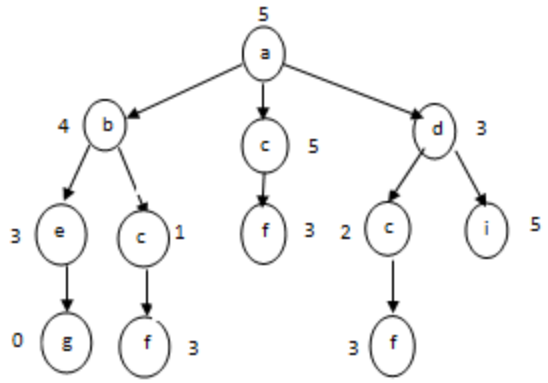
A algorithm is simply define as a best first search plus specific function. This specific function represent the actual distan ce
(levels) between the initial state and the current state and is denoted by g(n). A notice will be mentioned here that the same steps that are used in the best first search are used in an A algorithm but in addition to the g (n) as follow;

$$\mathbf{Fn= g(n) + h(n)}$$

g(n):- Measures the actual length of path from any state (n) to the start state.

H(n):- is a heuristic estimate of the distance from state n to the goal.

A-Algorithm :Example 2



Open	Closed
[A5]	[]
[D4,B5,C6]	[A5]
[C4,B5,I7]	[A5,D4]
[B5,F6,I7]	[A5,D4,C4]
[C3,E5,F6,I7]	[A5,D4,B5]
[E5,F6,I7]	[A5,D4,B5,C3]
[G3,F6,I7]	[A5,D4,B5,C3,E5]
	[A5,D4,B5,C3,E5,G3]

The goal is found &the resulted path is:

A5 → D3 → B4 → C1 → E3 → G0 =16

.

A*Algorithm

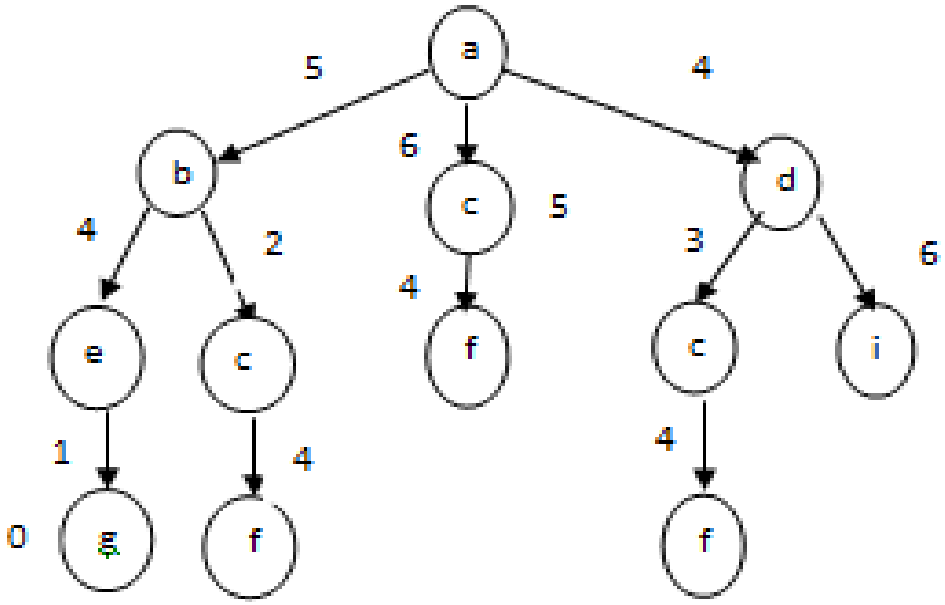
A* algorithm is simply define as a best first search plus specific function. This specific function represents the actual distance (levels) between the current state and the goal state and is denoted by h(n). It evaluates nodes by combining g(n), the cost to reach the node, a nd h(n), the cost to get from the node to the goal:

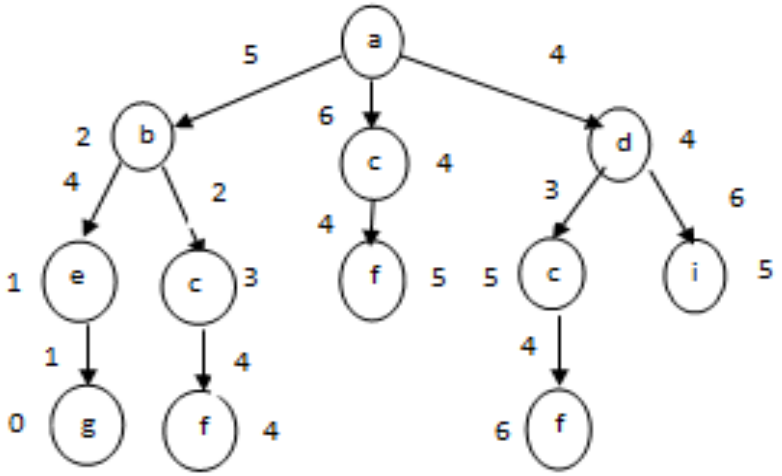
$$\mathbf{f(n) = g(n) + h(n).}$$

Since g(n) gives the path cost from the start node to node n, and h(n) is the estimated cost of the cheapest path from n to the goal, we have f (n) = estimated cost of the cheapest solution through n .

Thus, if we are trying to find the cheapest solution, a reasonable thing to try first is the node with the lowest value of g(n) + h(n). It turns out that this strategy is more than just reasonable: provided that the heuristic function h(n) satisfies certain conditions, A* search is both complete and optimal.

A* Example





- Open=[A] Closed[]
- Open=[B7,D8,C10] Closed=[A]
- Open=[D8,E10,C10] Closed=[A,B7]
- Open=[E10,C10,I15] Closed=[A,B7,D8]
- Open=[G10,C10,I15] Closed= [A,B7,D8,E10]
- Closed=[A,B7,D8,E10,G10]

- The goal is found &the resulted path is:
  - A0 - B5- D4-E4-G1 =14

A* Algorithm Properties

1) Admissibility

- Admissibility means that h(n) is less than or equal to the cost of the minimal path from n to the goal.

- the admissibility should satisfy these two conditions:
- 1- $h(n) \leq h^*(n)$
- 2- $g(n) \geq g^*(n)$

2) Monotonicity (consistency)

A heuristic function h is monotone if :-

a. For all state n_i and n_j , where n_j is a descendant of n_i

$$\mathbf{h(n_i)-h(n_j) \leq cost(n_i,n_j).}$$

Where cost (n_i,n_j) is the actual cost of going from state n_i to n_j.

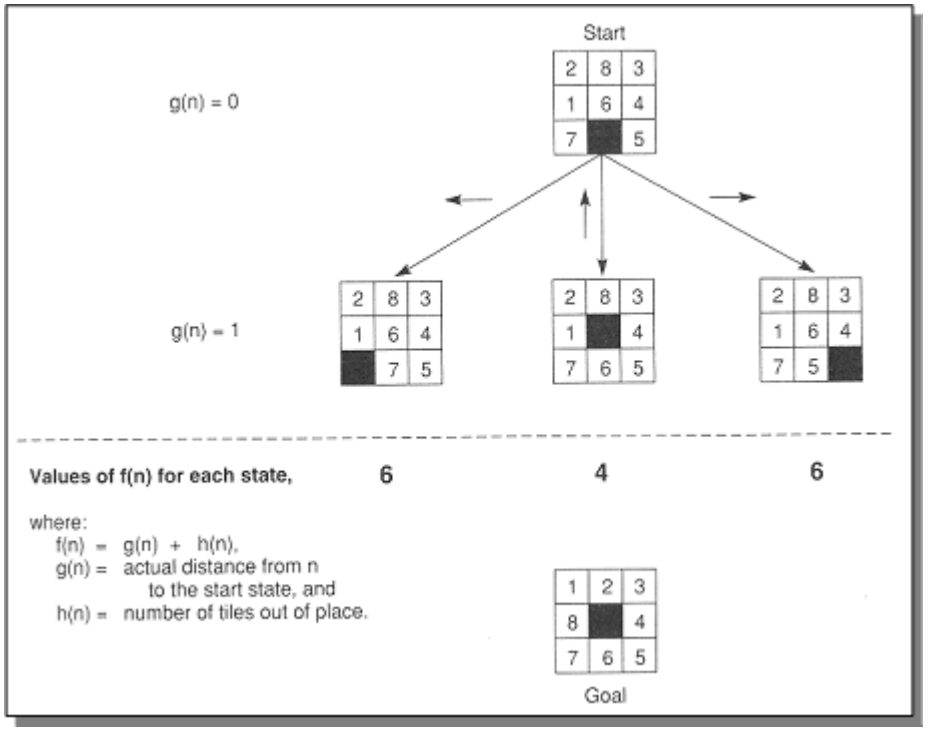
b. The heuristic evaluation of the goal state is zero , or h(goal)=0.

3) Informedness

For two A^* heuristics h1 and h2 , if $h1(n) \leq h2(n)$, for all states n in the search space , heuristics h2 is said to be more informed than h1.

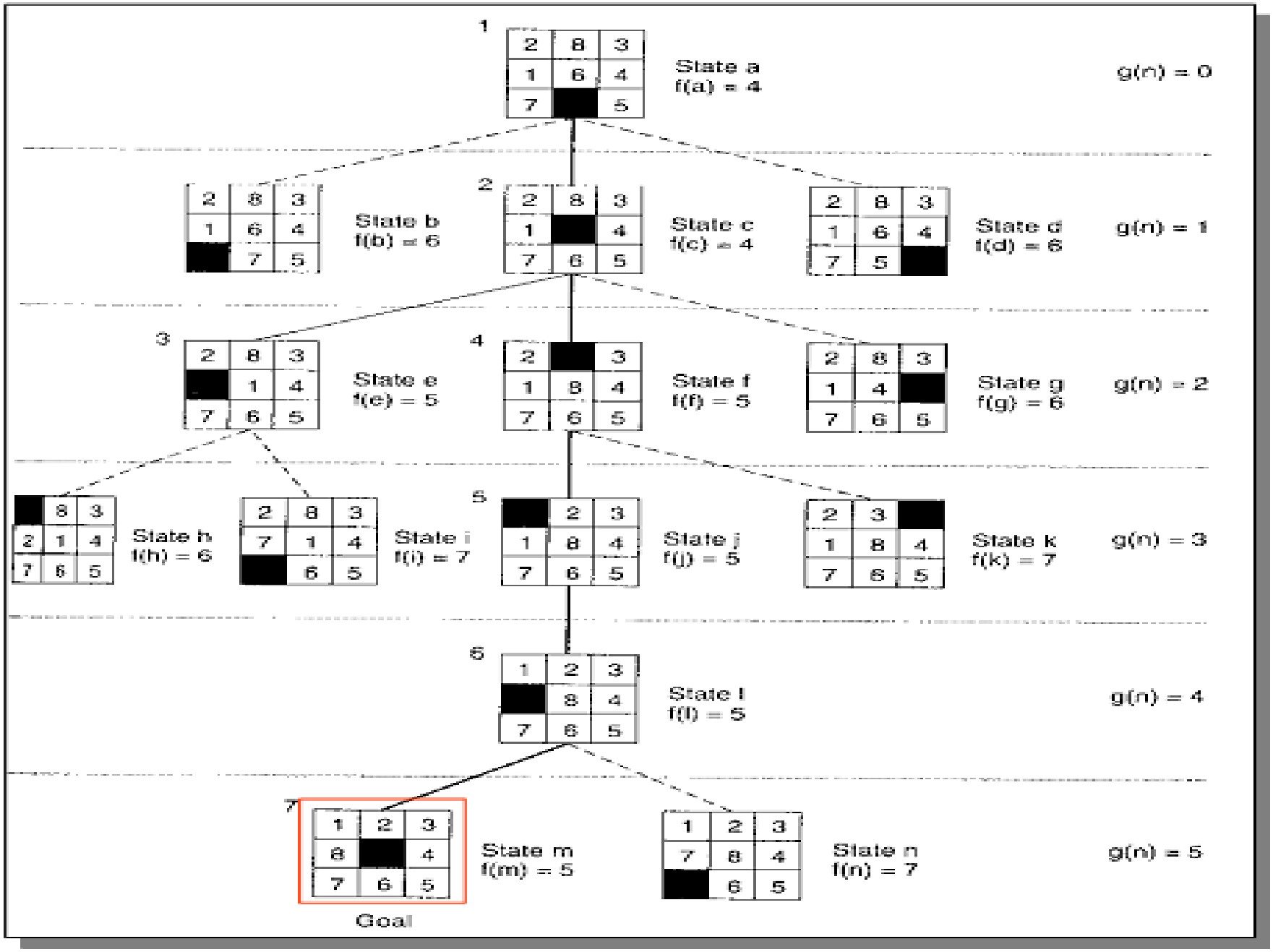
Complex Search Space and problem solving Approach

8-puzzle problem



To summarize

- 1. Operations on states generate children of the state currently under examination.
- 2. Each new state is checked to see whether it has occurred before (is on either open or closed), thereby preventing loops.
- 3. Each state n is given an I value equal to the sum of its depth in the search space g(n) and a heuristic estimate of its distance to a goal h(n). The h value guides search toward heuristically promising states while the **g** value prevents search from persisting indefinitely on a fruitless path.
- 4. States on open are sorted by their f values. By keeping all states on open until they are examined or a goal is found, the algorithm recovers from dead ends.
- 5. As an implementation point, the algorithm's efficiency can be improved through maintenance of the open and closed lists, perhaps as heaps or leftist trees.


1
State a
f(a) = 4
g(n) = 0
State b
f(b) = 6
2
State c
f(c) = 4
State d
f(d) = 6
g(n) = 1
3
State e
f(e) = 5
4
State f
f(f) = 5
State g
f(g) = 6
g(n) = 2
State h
f(h) = 6
State i
f(i) = 7
5
State j
f(j) = 5
State k
f(k) = 7
g(n) = 3
6
State l
f(l) = 5
g(n) = 4
7
State m
f(m) = 5
Goal
State n
f(n) = 7
g(n) = 5

After implementation of **A** algorithm, the Open and Closed is shown as follows:

- **1. Open=[a4], Closed=[]**
- **2. Open=[c4,b6,d6], Closed=[a4]**
- **3. Open=[e5,f5,b6,d6,g6], Closed=[a4,c4]**
- **4. Open=[f5,b6,d6,g6,h6,i7], Closed=[a4,c4,e5]**
- **5. Open=[j5,b6,d6,g6,h6,j7,k7], Closed=[a4,c4,e5,f5]**
- **6. Open=[l5, b6,d6,g6,h6,j7,k7], Closed=[a4,c4,e5,f5,j5]**
- **7. Open=[m5, b6,d6,g6,h6,j7,k7,n7], Closed=[a4,c4,e5,f5,j5,l5]**
- **8. Success, m=goal!!**

Example: Consider 8-puzzle problem with start state is shown as follows:

2	8	3
1	6	4
7		5

And the goal state is:

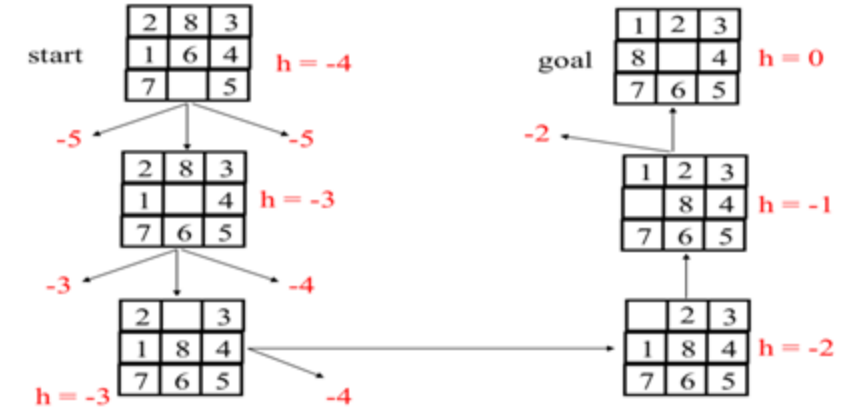
1	2	3
8		4
7	6	5

Assume the heuristic is calculated as following:

$h(n)$ = -(number of tiles out of place)

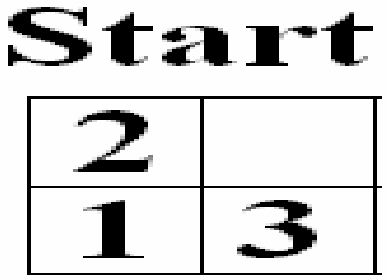
Draw the path to get the goal using Hill Climbing search algorithm?

Answer:

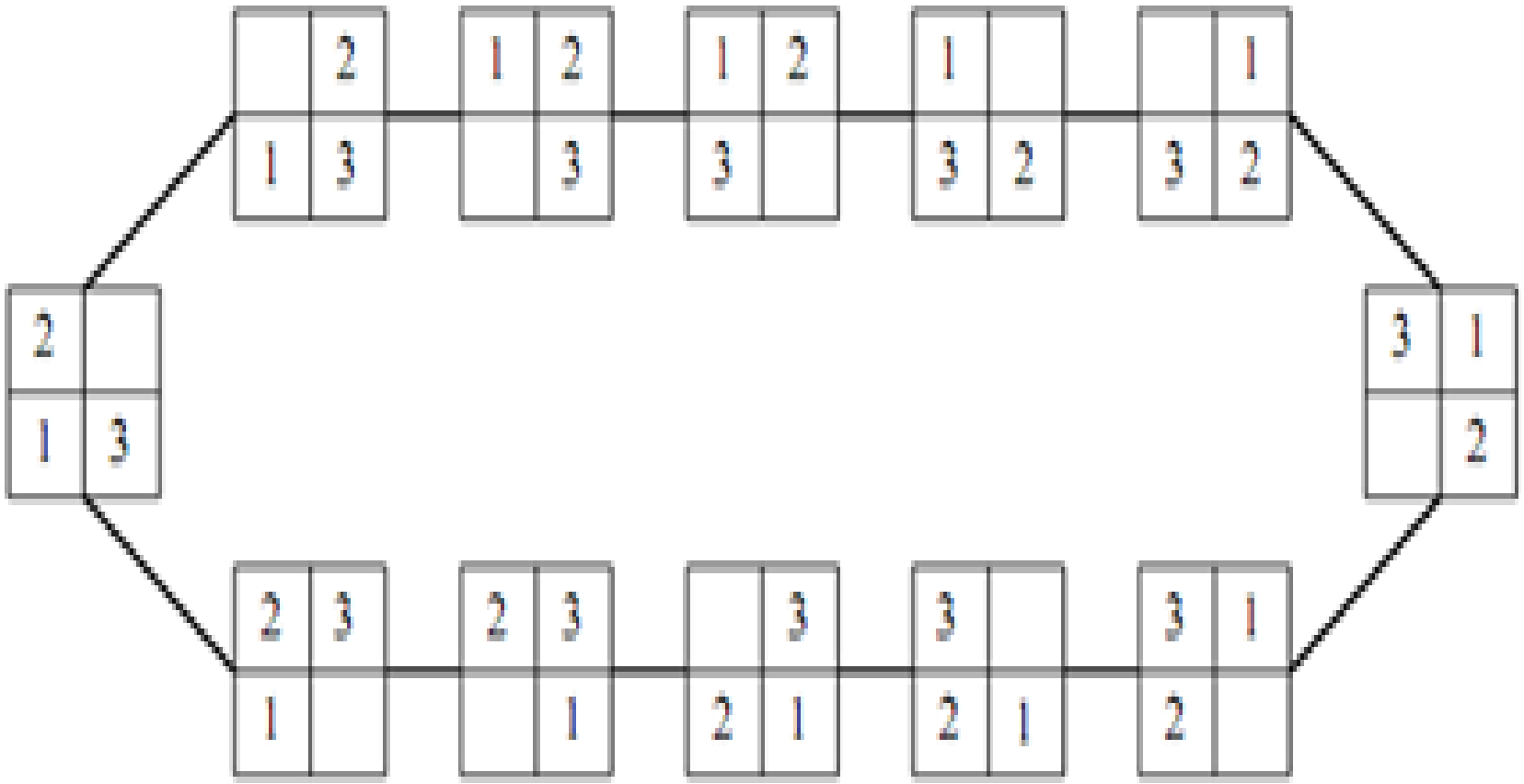


3-puzzle

Example: Consider the 3-puzzle problem, which is a simpler version of the 8-puzzle where the board is 2×2 and there are three tiles, numbered 1, 2, and 3. There are four moves: move the blank up, right, down, and left. The cost of each move is 1. Consider this start state:



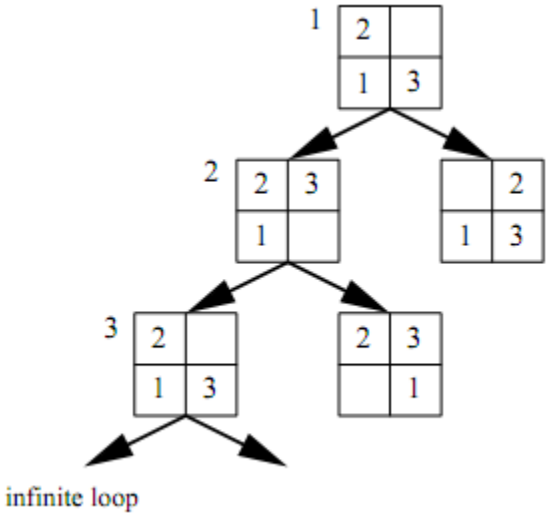
Draw the entire non-repeating state space for this problem, labeling nodes and arcs clearly?.



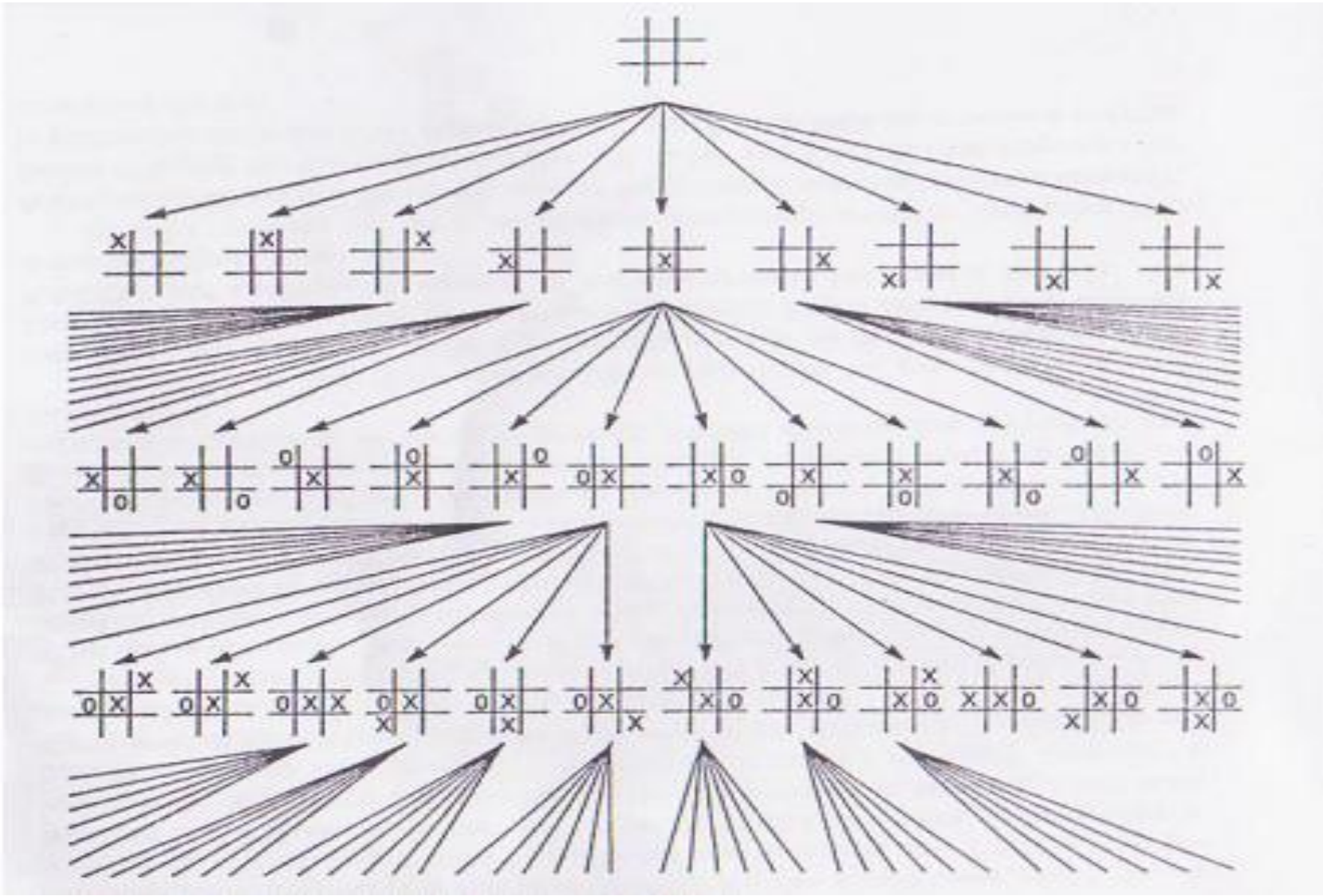
- Assume the goal is:
- 

- Are the goal is found using Depth First Search algorithm? If not explain why?

Search algorithm because there is an infinite loop as shown below:



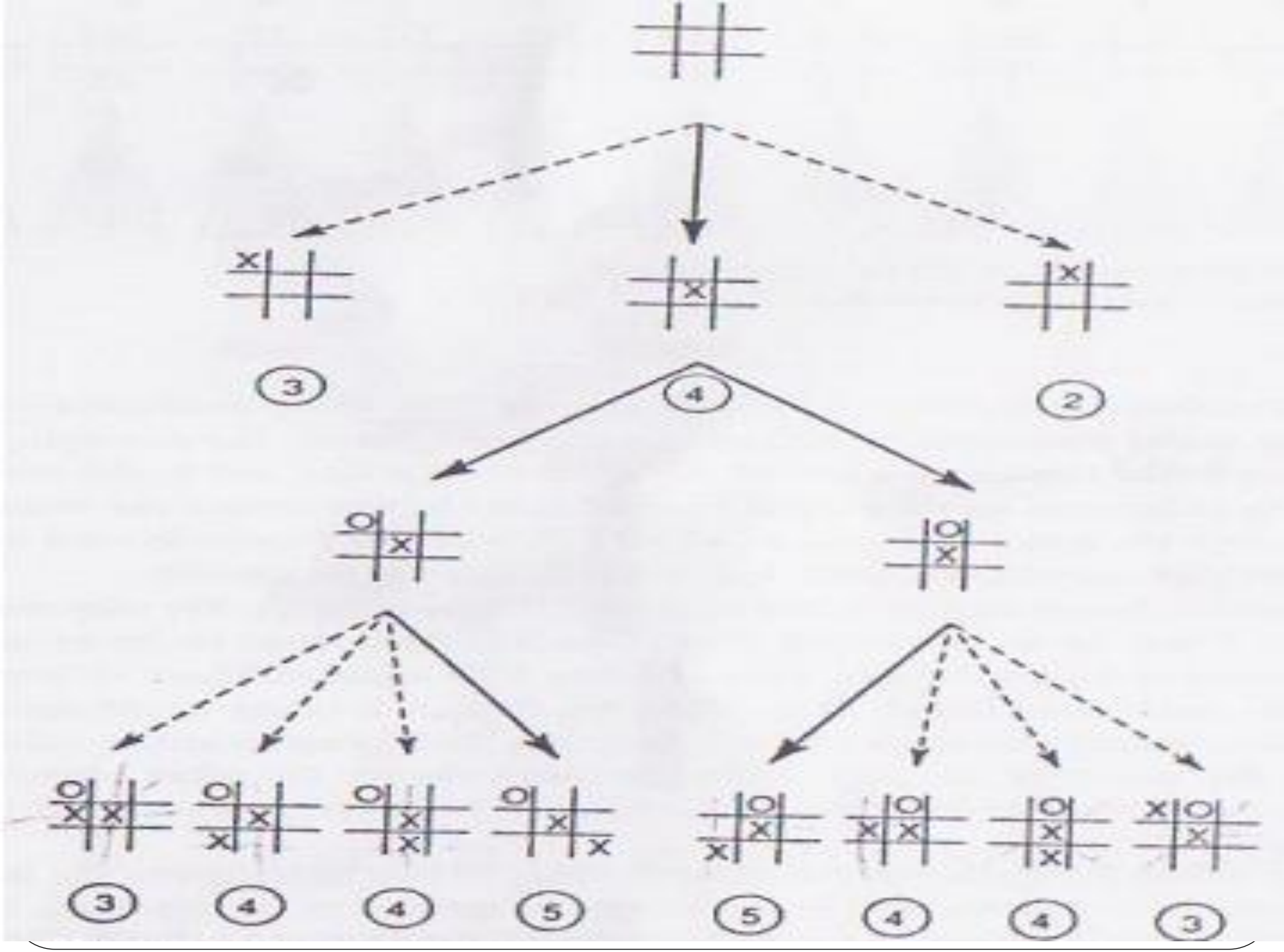
2) Tic – Tac – Toe Game



- The complexity of the search space is 9!
- 9! =9*8*7*……*1

- Therefore it is necessary to implement two conditions:
- 1- Problem reduction
- 2- Guarantee the solution

Three wins through a corner square	Four wins through a center square	Two wins through a side square

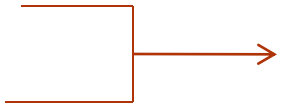

3
4
2
3
4
4
5
5
4
4
3

Knowledge Representation

Knowledge Representation

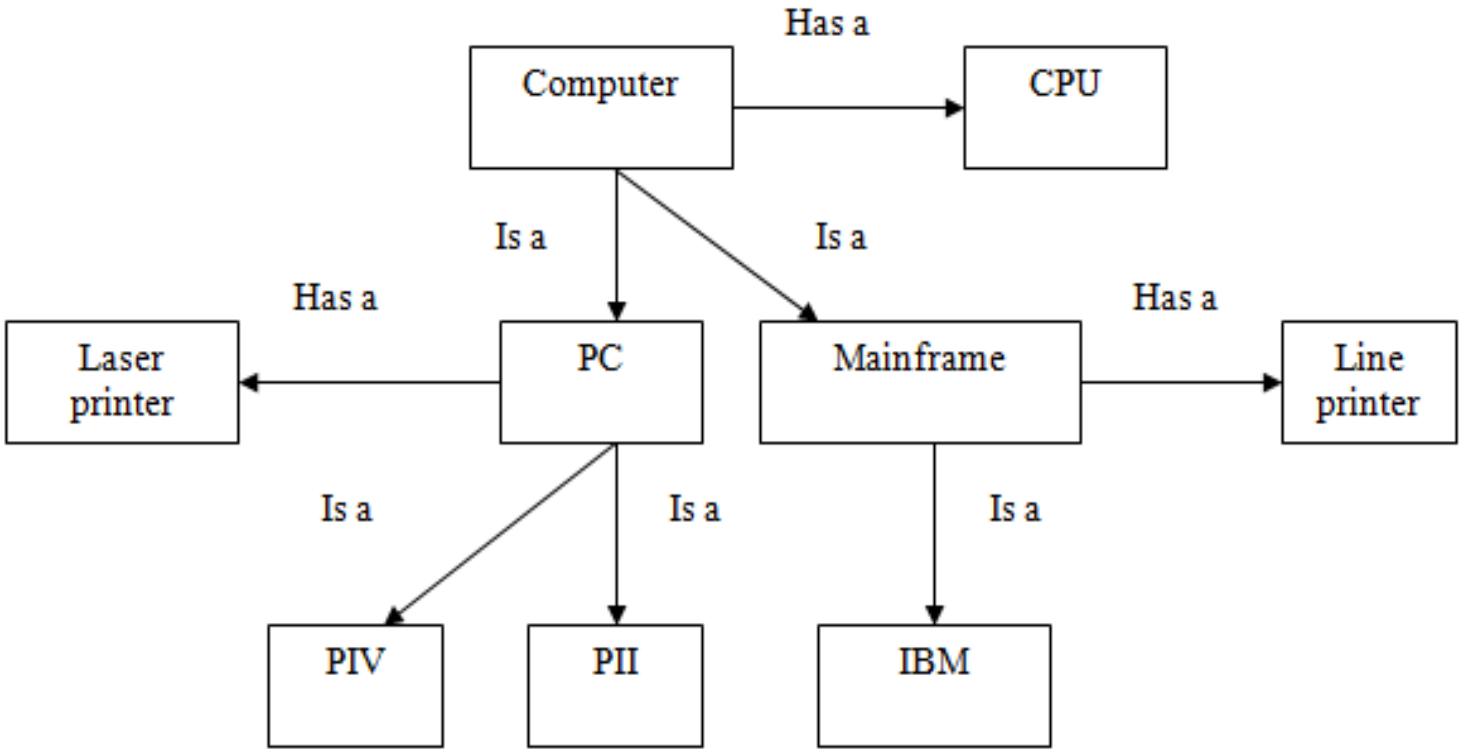
- There are many methods can be used for knowledge representation and they can be described as follows:-
- 1- Semantic net.
- 2- Conceptual graph.
- 3- Frames
- 4- Predicates logic.
- 5-Clause forms

1) Semantic Net

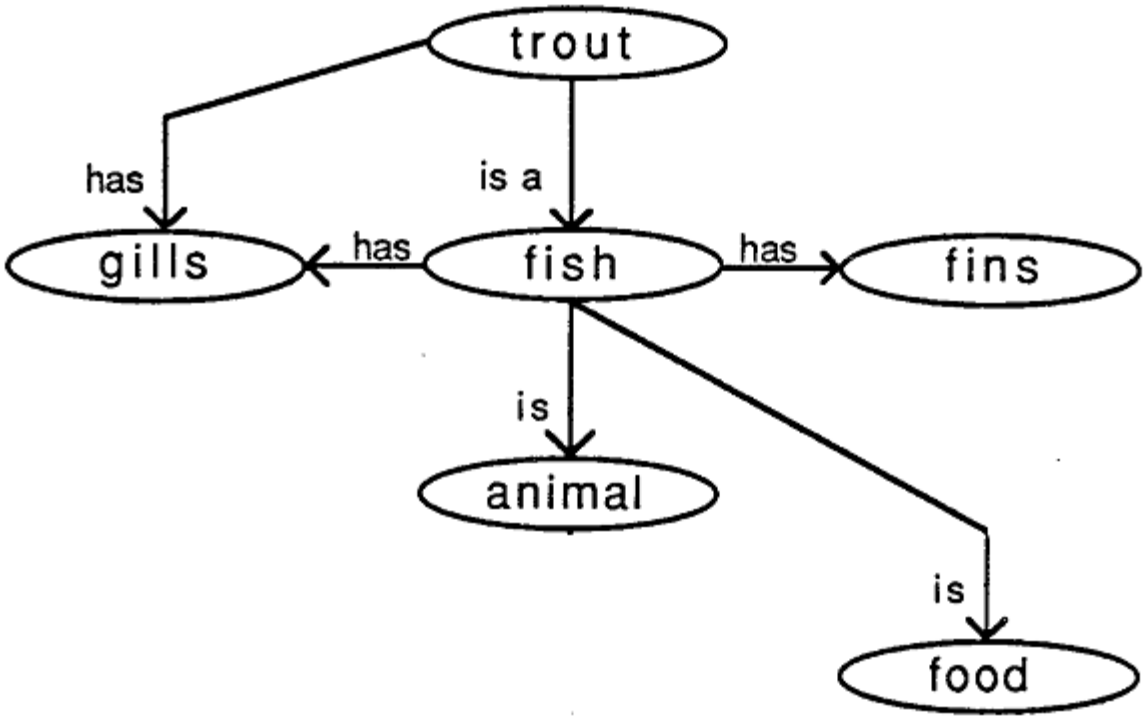
- It is consist of a set of nodes and arcs , each node is represented as a rectangle to describe the objects, the concepts and the events. The arcs are used to connect the nodes and they divided to three parts:-
- Is a:  for objects & types
- Is :  To define the object or describe it
- Has a
- can  To describe the properties of objects or the actions that the object can do
- **To represent the actions, events and objects** 

To represent the relation among objects 

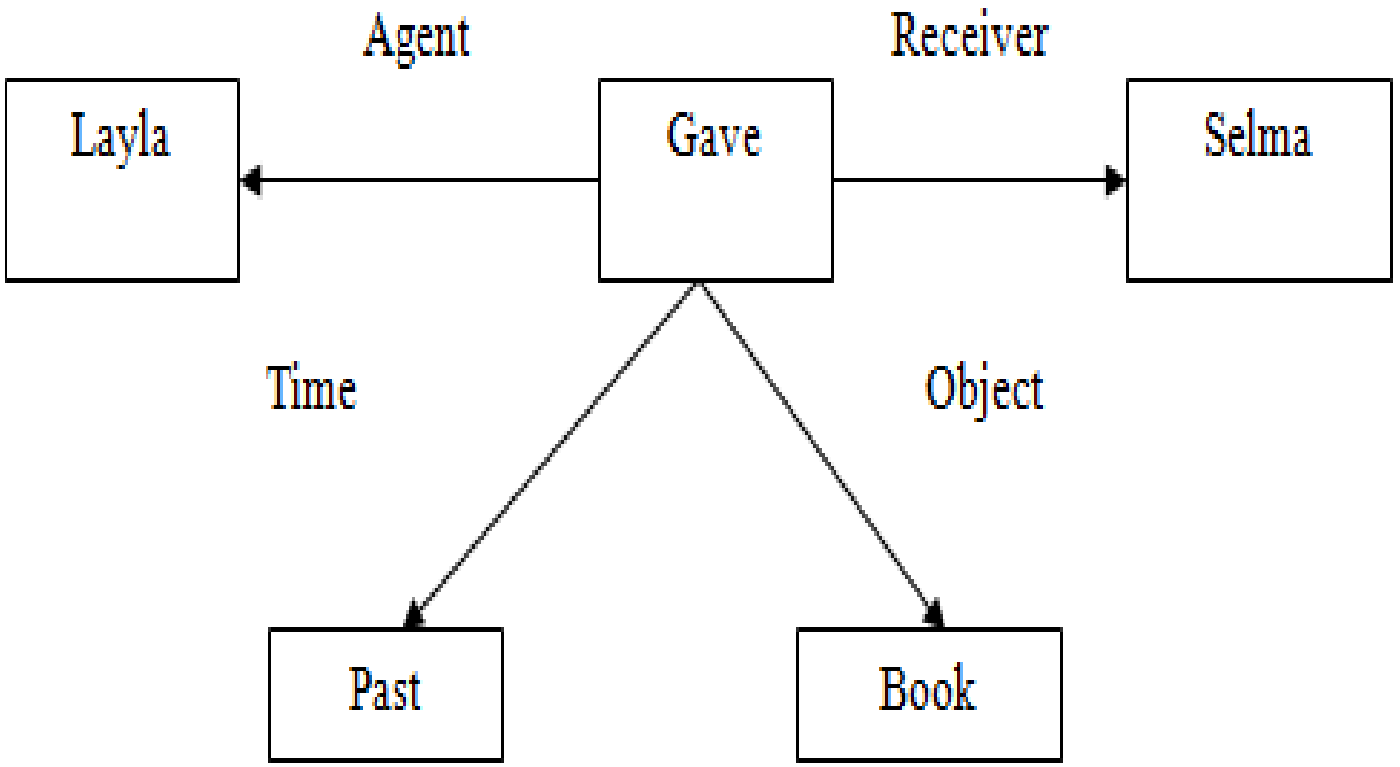
Example1: Computer has many part like a CPU and the computer divided into two type, the first one is the mainframe and the second is the personal computer ,Mainframe has line printer with large sheet but the personal computer has laser printer , IBM as example to the mainframe and PIII and PIV as example to the personal computer.



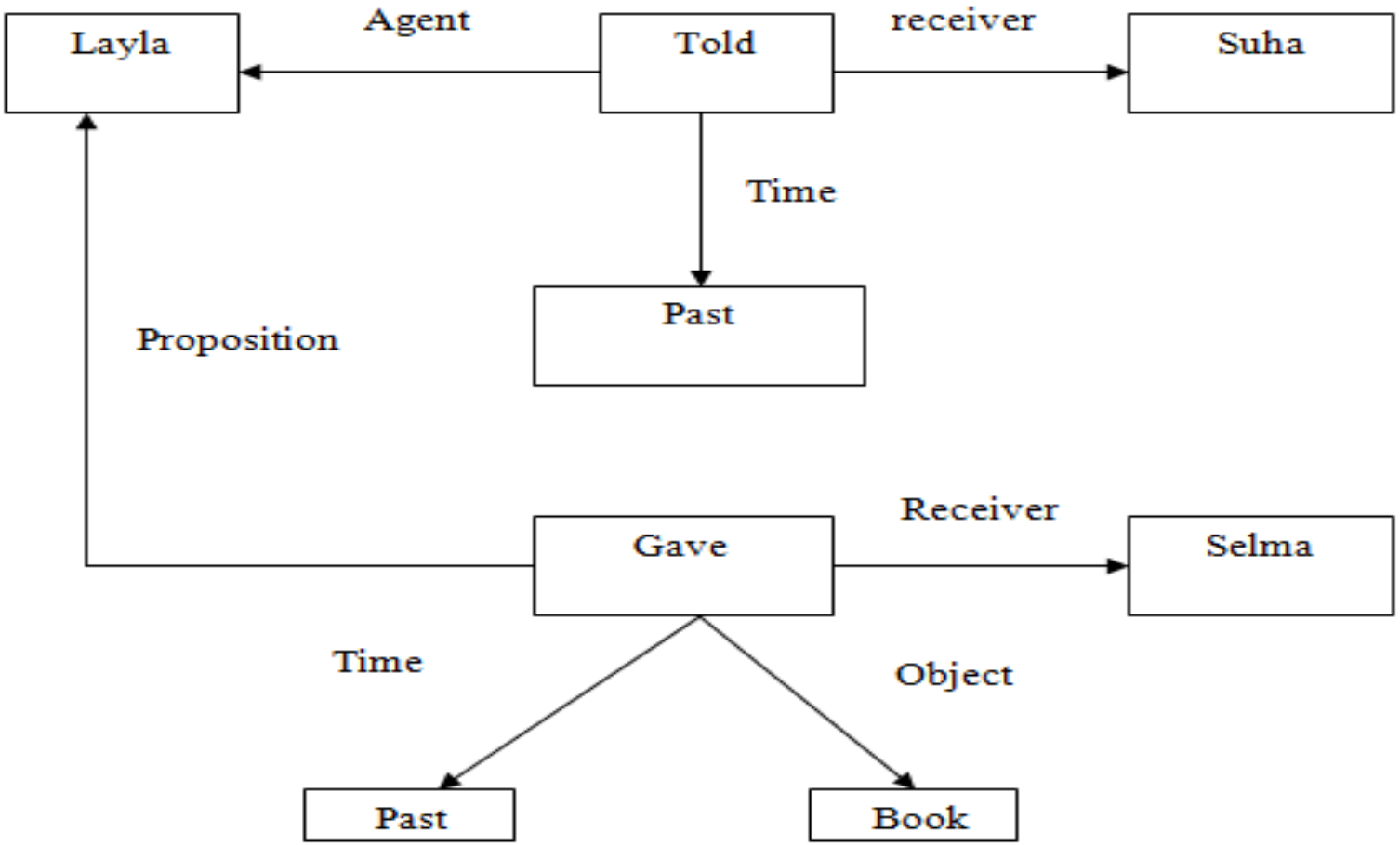
- **Example2:** Create the semantic network for the following facts (Note: You must append new indirect facts if they exist):
- A trout is a fish.
- A fish has gills.
- A fish has fins.
- Fish is food.
- Fish is animal.
- Solusion:
- There is a fact must be added that is "A trout has gills" because all the fishes have gills. The semantic network is shown below:


trout
has
is a
gills
has
fish
has
fins
is
animal
is
food

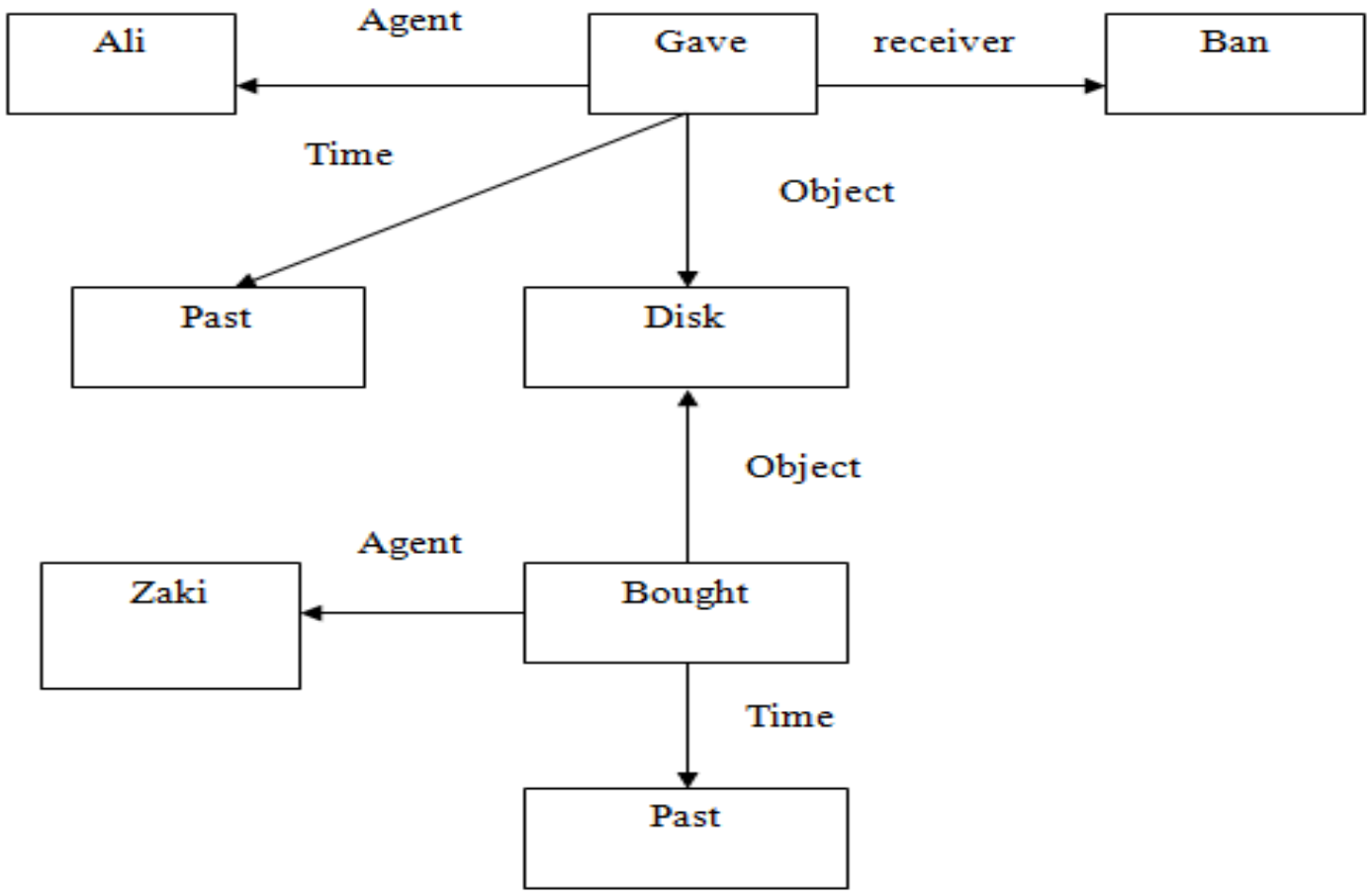
Example 2: Layla gave Selma a book



Example 3: Layla told Suha that she gave Selma a book



Example 4: Ali gave Ban a disk which is Zaki bought

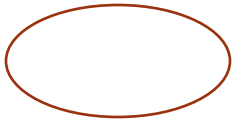


2) The Conceptual Graph

- Conceptual Graphs is a logical formalism that includes classes, relations, individuals and quantifiers. This formalism is based on semantic networks, but it has direct translation to the language of first order predicate logic, from which it takes its semantics. The main feature is standardized graphical representation that like in the case of semantic networks allows human to get quick overview of what the graph means. Conceptual graph is a bipartite orientated graph where instances of concepts are displayed as *rectangle* and conceptual relations are displayed as *ellipse*. Oriented edges then link these vertices and denote the existence and orientation of relation. A relation can have more than one edges, in which case edges are numbered.

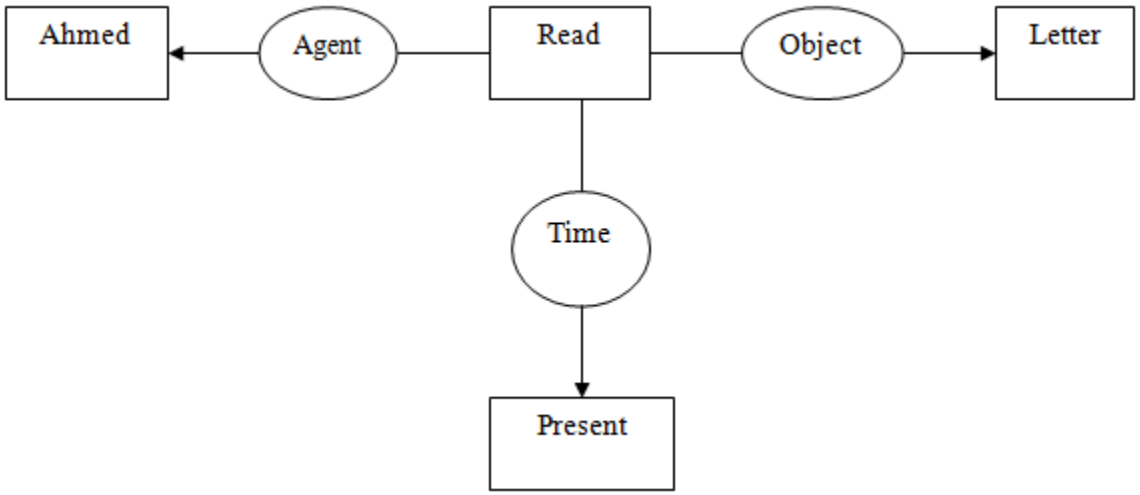
- It is similar to semantic net with two parts

- Is used to describe the nouns, the adjectives , the verbs(actions) and the objects.

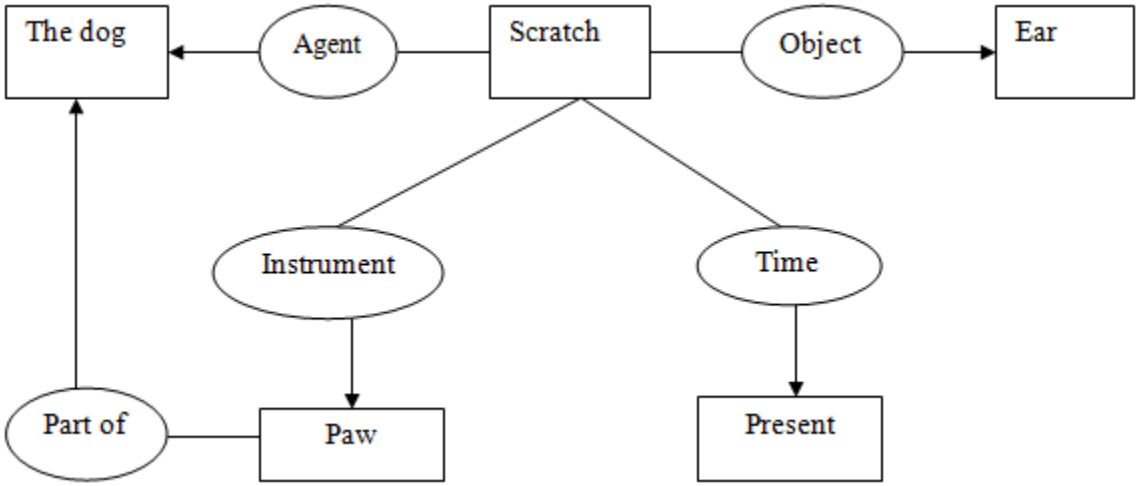


Is used to represent the relations among objects

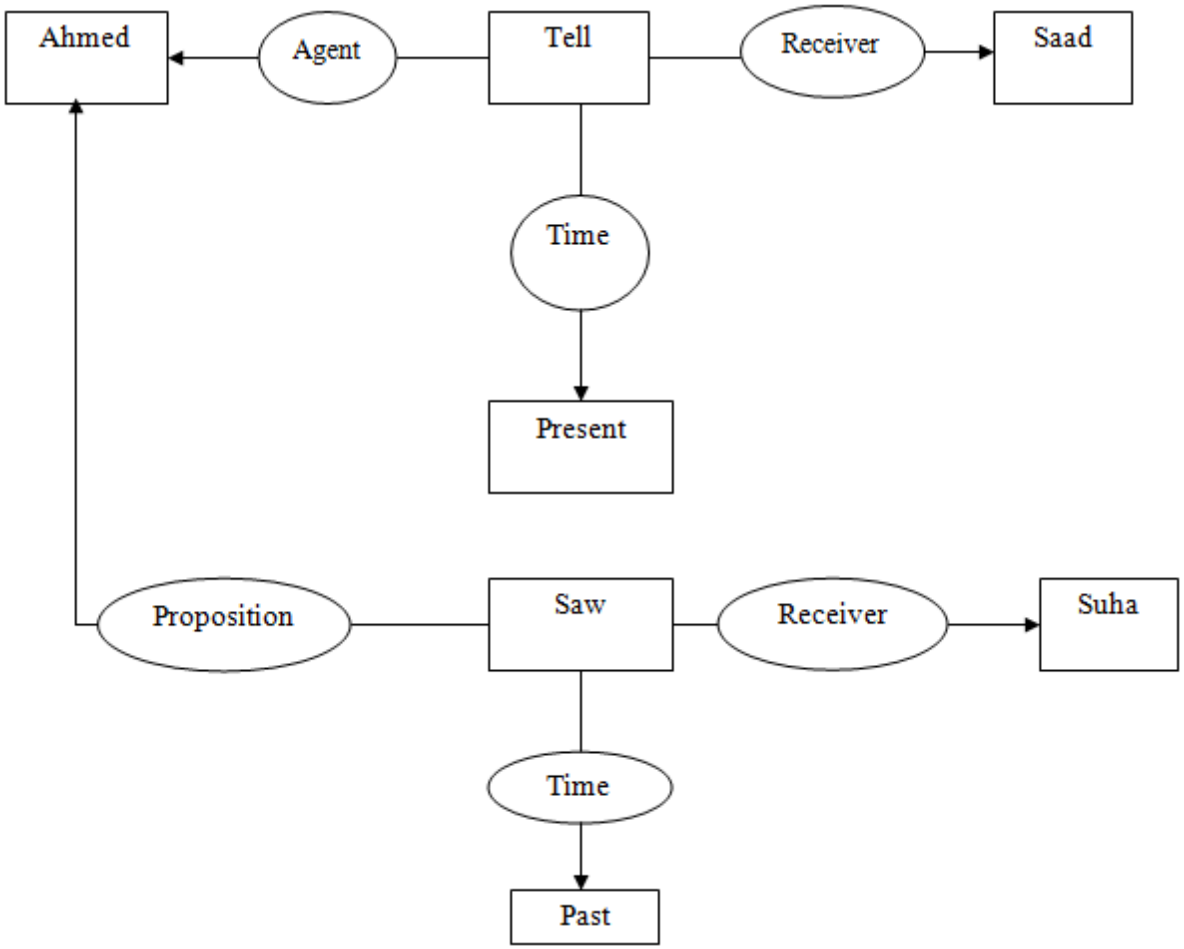
Example 1: Ahmed read a letter yesterday



Example 2:- The dog Scratch it ear with is paw



Example 3: Ahmed tell Saad that he saw Suha

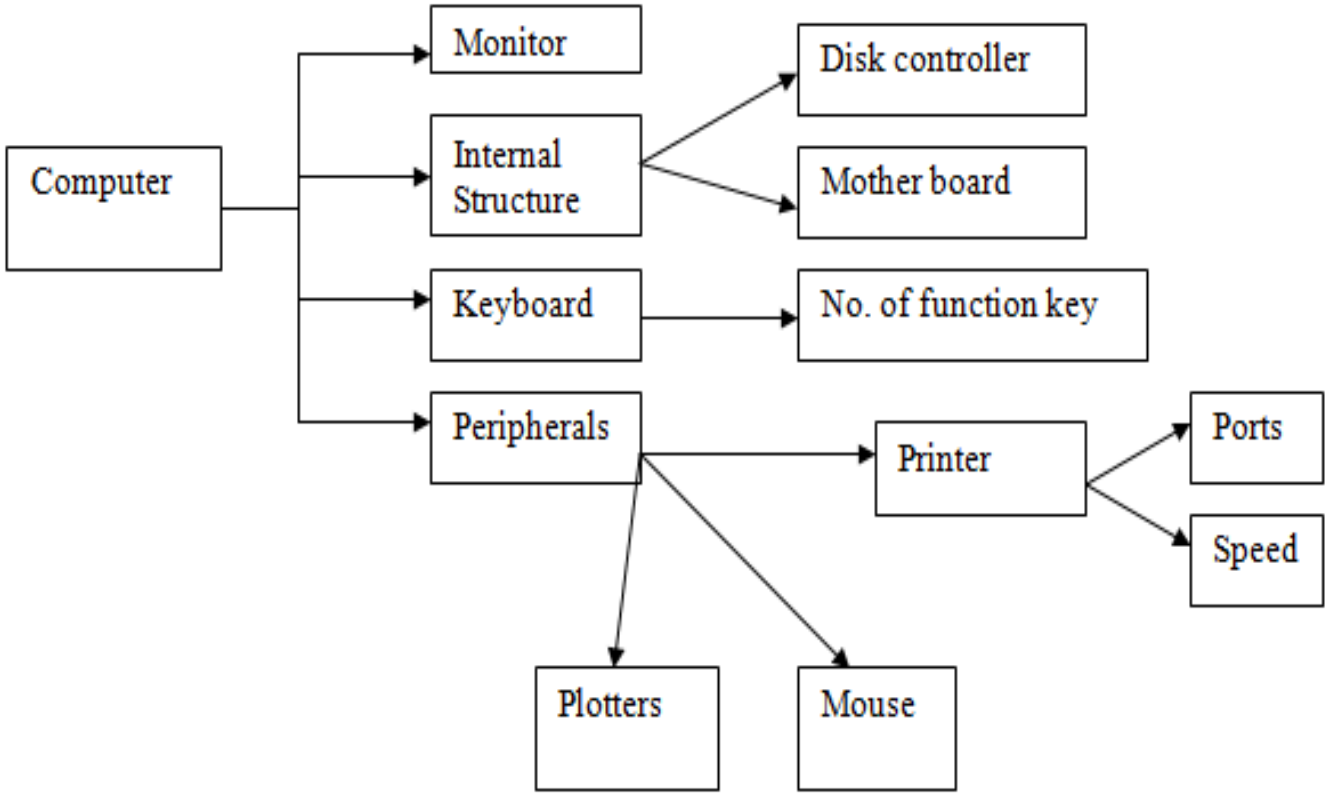


3) Frame

- **Frame can be divided to frame list and slot list.**

- **Frame-list(node-name, parent, [child]).**
- **Slot-list(node-name, parent).**

Example: Represent with frame list



Solution

- Frame –list(computer,_ ,[Internal structure, monitor, keyboard , perphilas]).
- Frame-list(Internal structure, computer, [disk controller, mother board]).
- Frame- list(printer, peripheral, [speed, ports]).
- .
- .
- Slot-list(motherboard, Internal structure).
- Slot-list(mouse, peripheral).
- .
- .
- .

- In the above example we have 5 frame lists and 8 slot lists.

Example: Convert the following statements to frame representation.

- ☒ Birds and fish are animals. Birds are moving by fly and they lay eggs. They have wings and their active at day light. kiwi and alberto are birds. The color of kiwi is black and white and the color of alberto is brown. Fish moves by swimming and has a skin.

The Prepositional and Predicates Calculus

The Prepositional and Predicates Calculus:

- The propositional calculus and predicate calculus are first of all languages. Using their words, phrases, and sentences, we can represent and reason about properties and relationships in the world. The first step in describing a language is to produce the pieces that make it up: its set of symbols.
- Propositional Calculus Symbols
- 1- The symbols of propositional calculus are: {P, Q, R, S, …}
- 2- Truth symbols: **{True, false}**
- 3- Connectives: **{∧ ,∨ , ¬ , → , ≡}**
- 4- Propositional symbols denote ***propositions***, or statements about the world that may be either true or false, Propositions are denoted by uppercase letters near the end of the English alphabet Sentences

For example:

- **P: It is sunny today.**
- **Q:The sun shines on the window.**
- **R:The blinds are down.**
- **(P→Q): If it is sunny today, then the sun shines on the window**
- **(Q→R): If the sun shines on the window, the blinds are brought down.**
- **(¬R):The blinds are not yet down.**

Propositional Calculus Sentence

- Every propositional symbol and truth symbol is a sentence.
- For example: **true, P, Q,** and **R** are sentences**.**

- The ***negation*** of a sentence is a sentence.
- For example: **¬P** and **¬false** are sentences.

- The ***conjunction***, **AND**, of two sentences is a sentence.
- For example: **P** $\wedge$ **¬P** is a sentence.

- The ***disjunction***, **OR** of two sentences is a sentence.
- For example: **P** $\vee$ **¬P** is a sentence.

- The ***implication*** of one sentence from another is a sentence.
- For example: **P** $\rightarrow$ **Q** is a sentence.

- The ***equivalence*** of two sentences is a sentence.
- For example: **P** $\vee$ **Q** $\equiv$ **R** is a sentence.
- Legal sentences are also called ***well-formed formulas*** or ***WFFs***.

- In expressions of the form **P** $\wedge$ **Q, P** and **Q** are called the ***conjuncts***. In **P** $\vee$ **Q**, **P** and **Q** are referred to as ***disjuncts***.
- In an implication, **P** $\rightarrow$ **Q, P** is the ***premise*** and ***Q***, the ***conclusion*** or ***consequent.***
- In propositional calculus sentences, the symbols **( )** and **[ ]** are used to group symbols into sub-expressions and so to control their order of evaluation and meaning.

- **For Example: (P ∨ Q) ≡ R** is quite different from **P ∨ (Q ≡ R)** as can be demonstrated using truth tables. An expression is a sentence, or well-formed formula, of the propositional calculus if and only if it can be formed of legal symbols through some sequence of these rules.

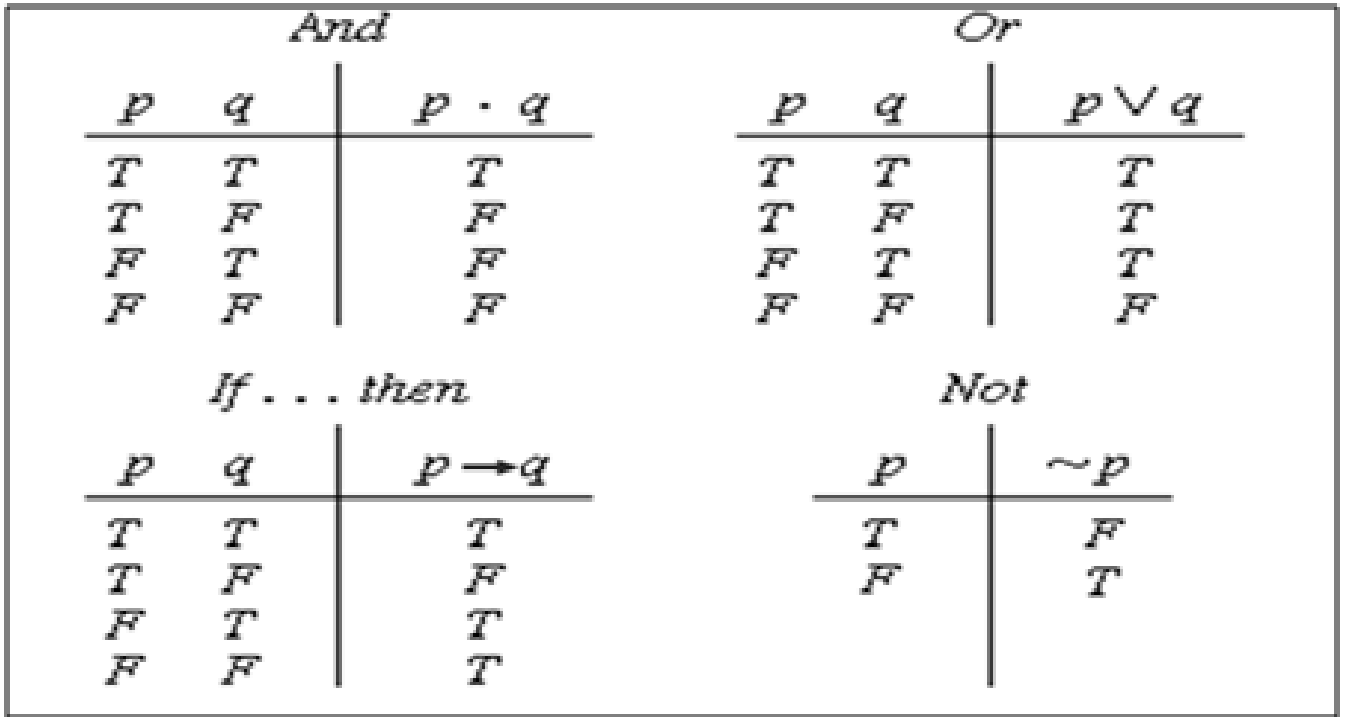
For Example, the truth table for $P \wedge Q$, **Fig.(a)** , lists truth values for each possible truth assignment of the operands.

P	Q	$P \wedge Q$
T	T	T
T	F	F
F	T	F
F	F	F

P	Q	¬P	¬PvQ	P→Q	(¬PvQ) ≡(P→Q)
T	T	F	T	T	T
T	F	F	F	F	T
F	T	T	T	T	T
F	F	T	T	T	T

- By demonstrating that two different sentences in the propositional calculus have identical truth tables, we can prove the following equivalences. For propositional expressions **P, Q,** and **R**:
- 1) $\neg (\neg P) \equiv P$
- 2) $P \rightarrow Q \equiv \neg P \vee Q$
- 3) The contra positive law: $(P \rightarrow Q) \equiv (Q \rightarrow P)$
- 4) De Morgan's law:
- $\neg(P \vee Q) \equiv (\neg P \wedge \neg Q)$ and $\neg(P \wedge Q) \equiv (\neg P \vee \neg Q)$
- 5) The commutative laws:
- $(P \wedge Q) \equiv (Q \wedge P)$ and $(P \vee Q) \equiv (Q \vee P)$
- 6) The associative law: $((P \wedge Q) \wedge R) \equiv (P \wedge (Q \wedge R))$
- 7)The associative law: $((P \vee Q) \vee R) \equiv (P \vee (Q \vee R))$
- 8)The distributive law: $P \vee (Q \wedge R) \equiv (P \vee Q) \wedge (P \vee R)$
- 9)The distributive law: $P \wedge (Q \vee R) \equiv (P \wedge Q) \vee (P \wedge R)$

- Identities such as these can be used to change propositional calculus expressions into a syntactically different but logically equivalent form. These identities may be used instead of truth tables to prove that two expressions are equivalent: find a series of identities that transform one expression into the other.
- The ability to change a logical expression into a different form with equivalent truth values is also important when using ***inference rules*** (**modus ponens, and resolution**) that require expressions to be in a specific form.
- Truth table then will list all possible truth value assignments to the propositions of an expression, the standard truth tables are shown the figure below:


And
p q p · q
T T T
T F F
F T F
F F F
Or
p q p ∨ q
T T T
T F T
F T T
F F F
If . . . then
p q p → q
T T T
T F F
F T T
F F T
Not
p ~p
T F
F T

Example: Use a truth table to list all possible truth value assignments to the propositions of the expression $(P \wedge Q) \vee (\neg Q \vee P)$.

P	Q	$P \wedge Q$	$\neg Q$	$\neg Q \vee P$	$(P \wedge Q) \vee (\neg Q \vee P)$
T	T	T	F	T	T
T	F	F	T	T	T
F	T	F	F	F	F
F	F	F	T	T	T

Example: Prove that $(P \wedge Q)$ is not equivalent to $(P \rightarrow Q)$, in other word prove $(P \wedge Q) \ (P \rightarrow Q)$

P	Q	$(P \wedge Q)$	$(P \rightarrow Q)$
T	T	T	T
T	F	F	F
F	T	F	T
F	F	F	T

≠

- **Example:** True or false**: (P∧Q)** not equivalent **(P→Q).**
- **Answer:** true**.**

- **H.W:** True or false**: ((P→Q)∧Q)→P. Answer:** ???**.**

Example: Convert the following English sentences to propositional calculus sentences

- **1- It is hot.**
- **2- It is not hot.**
- **3- If it is raining, then will not go to mountain.**
- **4- The food is good and the service is good.**
- **5- If the food is good and the service is good then the restaurant is good.**

Answer:

- It is hot
  p
- It is not hot
  ¬p
- If it is raining, then will not go to mountain
  p → ¬q
- The food is good and the service is good
  x ∧ y
- If The food is good and the service is good then the restaurant is good
  x ∧ y → z

4.2 The Predicate Calculus (Also known as First-Order Logic):

- To solve the limitations in the prepositional calculus, you need to analyze propositions into predicates and arguments, and deal explicitly with quantification. Predicate calculus provides formalism for performing this analysis of prepositions and additional methods for reasoning with quantified expressions.
- For example, instead of letting a single prepositional symbol, **P**, denotes the entire sentence "**it rained on Tuesday**," we can create a predicate weather that describes a relationship between a date and the weather:

weather (rain, Tuesday)

through inference rules we can manipulate predicate calculus expression accessing their individual components and inferring new sentences.

Predicate calculus also allows expressions to contain variables. Variables let us create general assertions about classes of entities. For example, we could state that for all values, of X, where X is a day of the week, the statement:

weather (rain, X) is true ;

I,e., it rains it rains every day. As with propositional calculus, we will first define the syntax of the language and then discuss its semantics.

Example: Convert the following engilsh sentences to predicate calculus sentences:

- 1. If it is raining, tom will not go to mountain
- 2. If it doesn't rain tomorrow, Tom will go to the mountains.
- 3. All basketball players are tall.
- 4. Some people like anchovies.
- 5. John like anyone who likes books.
- 6. Nobody likes taxes.
- 7. There is a person who writes computer class.
- 8. All dogs are animals.
- 9. All cats and dogs are animals.
- 10. John did not study but he is lucky.
- 11. There are no two adjacent countries have the same color.
- 12. All blocks supported by blocks that have been moved have also been moved. Note: you can use the following predicates:
- • block(X) means X is a block
- • supports(X, Y) means X supports Y
- • moved(X) means X has been moved

Answer:

1. weather (rain) → ¬ go(tom,mountain)
2. ¬ weather (rain, tomorrow) →go(tom, mountains).
3. ∀ X (basketball _ player(X) → tall (X))
4. ∃ X (person(X) ∧ likes(X, anchovies)).
5. ∃ X like(X, book) →like(john,X)
6. ¬ ∃ X likes(X,taxes).
7. ∃X write(X, computer_ class)
8. ∀X dogs(X) →animals(X)
9. ∀X∀Y cats(X)∧dogs(Y) →animals(X)∧animals(Y).
10. ¬ study(john)∧ lucky(john)
11. ∀X∀Y (county(X) ∧ county(Y) ∧ adjacent(X,Y)) → ¬ (color(X) ≡ color(Y)). Or we say: ∀X∀Y ¬ county(X) ∨ ¬county(Y) ∨ ¬adjacent(X,Y) ∨ ¬ (color(X) ≡ color(Y)).
12. ∀X∀Y block(X) ∧ block(Y) ∧ supports(X,Y) ∧ moved(X) → moved(Y)

Metaheuristic

Local search

Simulated annealing

Tabu search

Metaheuristics

- **Metaheuristics:** A new kind of approximate algorithm has emerged which tries to combine basic heuristic methods in higher level frameworks aimed at efficiently and effectively exploring a search space. These methods nowadays commonly called metaheuristics.
- The term metaheuristic is derived from composition of two Greek words, the suffix Meta mean beyond , in upper level and the verb heuriskein which means "to find". Metaheuristic were often called modern heuristic.
- Heuristic algorithm typically intends to find a good solution to an optimization problem by 'trail –and- error' in reasonable amount of computing time. There is no guarantee to find the best or optimal solution, though it might be better or improved solution than an educated guess. Metaheuristic algorithms are higher level heuristic algorithms.
-
- **Different between heuristic and metaheuristic**
- heuristic is solving method for special problem (it can benefit the from the properties of the solved problem).
- Metaheuristic is generalized solving method like Genetic Algorithm (GA), Tabu Search (TS), etc.

Metaheuristics

Key ideas of metaheuristics:

- 1- Use the information being gathered to guide the search towards the global optimimum.
- 2- It is capable to escaping from a local optima.
- 3- Examine Neighborhood Structure some solutions are similar to other neighbor, so we need a neighborhood structure.

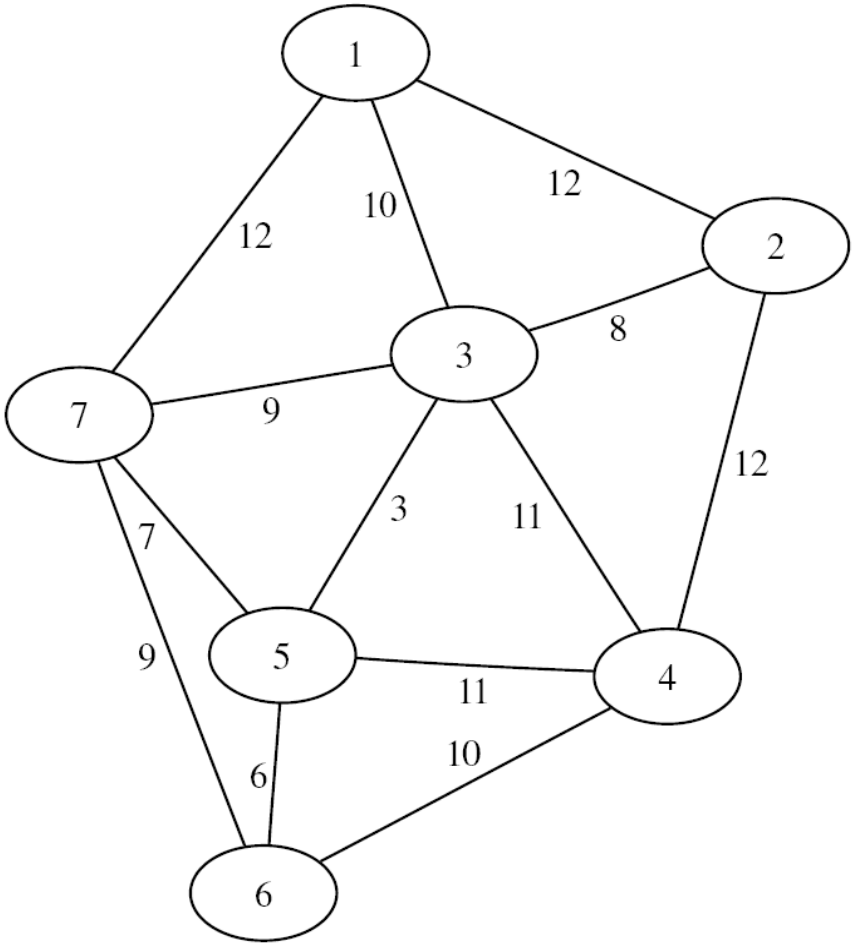
The objectives of metaheuristics are:

- Search using structure without getting stuck.
- Don't keep trying the same solutions.
- Combine one or more properties of good solutions when generating new solutions.

- **Advantage of metaheuristics:**
- It tends to move relatively quickly towards very good solutions, so it provides a very efficient way of dealing with large complicated problems.
- Useful in cases where traditional methods get stuck at local minimas.
- Common area of application is combinatorial optimization problems.
- **Disadvantage of metaheuristics:**

There is no guarantee that the best solution found will be the optimal solution.

Travelling salesman Problem (TSP)



- We will use the travelling salesmen problem (TSP) on the graph as an example problem for the metaheurisics discussed. Travelling Salesman Problem(TSP):
- A salesman spends his time visiting n cities (or nodes) cyclically. Starting from the home city, the salesman wishes to determine which route to follow to visit each city exactly once before returning to the home city so as to minimize the total distance of the tour.
- The difficulty of the travelling salesman problem increases rapidly as the number of cities increases. For a problem with n cities and a link between every pair of cities, the number of feasible routes to be considered is (n-1)!/2. Due to enormous difficulty in solving the TSP, heuristic methods guided by the metaheuristics, address such problems.
- Heuristic methods involve sequence of feasible trial solutions, where each solution is obtained by making certain adjustment in current trial solution.
- Sub tour Reversal: Adjusting a sequence of cities visited in the current solution by selecting a subsequence of the cities and simply reversing the order.

Example

- Initial trial solution is the following sequence of cities visited: 1-2-3-4-5-6-7-1
- with total distance = 69.
- While reversing the sequence 3-4 , we obtain new trial solution: 1-2-4-3-5-6-7-1
- with total distance = 65.
- Neighbors: We say 2 tours/solutions/cycles are neighbors if we can transform one to the other by a subtour reversal.
- Degree of Neighbor: The degree of a neighbor A to B equals the minimum number of sub tour reversals required to get from A to B.
-

Local search:

- **Local search:** A local search named is neighborhood local search is an iterative algorithm that moves from one solution S to another S' according to some neighborhood structure.

- Local search algorithms have 2 key advantages:
  - They use very little memory
  - They can find reasonable solutions in large or infinite (continuous) state spaces.
- Some examples of local search algorithms are:
  - Hill-climbing
  - Random walk
  - Simulated annealing

Local Search Algorithm

$s = s_0$; /* Generate an initial solution s_0 */
While not Termination_Criterion **Do**
  Generate ($N(s)$) ; /* Generation of candidate neighbors */
  If there is no better neighbor **Then** Stop ;
  $s = s'$; /* Select a better neighbor $s' \in N(s)$ */
Endwhile
Output Final solution found (local optima).

Simulated annealing

- Annealing is a thermal process for obtaining low energy states of a solid in a heat bath. the process contains two steps:
  - Increase the temperature of the heat bath to a maximum value at which the solid melts.
  - Decrease carefully the temperature of the heat bath until the particles arrange themselves in the ground state of the solid. Ground state is a minimum energy state of the solid.
- The ground state of the solid is obtained only if the maximum temperature is high enough and the cooling is done slowly.

The Simulated Annealing Algorithm

Input: Cooling schedule.
$s = s_0$; /* Generation of the initial solution */
$T = T_{max}$; /* Starting temperature */
Repeat
  Repeat /* At a fixed temperature */
    Generate a random neighbor s' ;
    $\Delta E = f(s') - f(s)$;
    If $\Delta E \leq 0$ **Then** $s = s'$ /* Accept the neighbor solution */
    Else Accept s' with a probability $e^{\frac{-\Delta E}{T}}$;
  Until Equilibrium condition
  /* e.g. a given number of iterations executed at each temperature T */
  $T = g(T)$; /* Temperature update */
Until Stopping criteria satisfied /* e.g. $T < T_{min}$ */
Output: Best solution found.

Travelling Salesman Example

- Suppose the $T_{max} = 500$
- **Initial trial solution:** 1-2-3-4-5-6-7-1 Distance = 69
- **Iteration 1:** Reverse (3-4) 1-2-4-3-5-6-7-1 Distance = 65 (65-69=-4 <0) accepted
- Accept as new solution.
- **Iteration 2:** Reverse (3-5-6) 1-2-4-6-5-3-7-1 Distance = 64 (64-65=-1 <0) accepted
- Accept as new solution.
- **Iteration 3:** Reverse (3-7) 1-2-4-6-5-7-3-1 Distance = 66
- Since the new distance is "worse," accept as new solution with some probability
- Continue for some fixed number of iterations, or until temperature function falls below a given threshold.
- $66-64=2 \leq 0$ no then taking the probability for the solution
- $\alpha = e^{-\Delta E / T}$
- $\alpha = e^{-2/500} = 0.99$
- $\rightarrow 0.99 > 0$ yes then accept as new solution
- Temperature Update using:
- $T = \alpha T$
- $T = 0.9 * 500$
- $T = 450$ (new temperature)
- then repeat the loop (with 3 iterations) until reach to the goal with low distance and temperature or number of iterations which determined in advance.

- **Tabu Search**
- Tabu search, created by Fred W. Glover in 1986 and formalized in 1989, is a metaheuristic that guides a local search procedure to explore the solution space beyond local optimality.
- One of the features of tabu search is to avoid bad solutions which have already been explored i.e use of memory to guide the search by using tabu list to record recent searches. Essentially, Tabu search makes some moves illegal by maintaining a list of 'tabu' moves.
- For example, if A is a neighbor of B in the TSP then B is a neighbor of A. But if you have already chosen B over A, there might not be any reason to search A again.

- 1. Tabu search is a local search strategy with a flexible memory structure. Simulated annealing has no memory structure(it relies on randomization)
- 2. Tabu search has two prominent features:
- - Adaptive memory
- - Reasonable exploration strategies
- 3. Main features
- -always move to the best available neighborhood solution point, even if it is worse than the current solution point.
- 4. intensification and diversification التكثيف والتنوع
- - Intensify: To intensify the search is to search the local area (portion of feasible region) more thoroughly.
- - Diversify: To diversify the search is to force the search away from the current solution (to unexplored areas of feasible region).
- - Length of Tabu List: The length of the list signifies the balance between
- intensify/diversify.
- 5. Long term memory

The Tabu Search Algorithm

- S=generate initial solution ( ) /Initialization/
- TabuList=0
- **While** termination condition not met **do** / Iteration/
- S=choose best of (N(s)\tabuList) / Compare all possible moves/
- Update (TabuList) / Update List/
- **End while**
- Stop after a fixed time or CPU usage, or there are no feasible moves. The optimal solution is the best solution so far.
-

Travelling Salesman Example

- **Initial trial solution:** 1-2-3-4-5-6-7-1 Distance = 69 Tabu
- **list :** Null
- **Iteration 1:** Reverse 3-4
- **Deleted links:** 2-3 and 4-5
- **Added links :** 2-4 and 3-5
- **Tabu List:** (2-4), (3-5)
-
- **New trials Solution:** 1-2-4-3-5-6-7-1 Distance = 65
- **Iteration 2:** Reverse: 3-5-6
- **Delete links:** 4-3 and 6-7
- **Add links:** 4-6 and 3-7
-
- **Tabu List:** 2-4, 3-5, 4-6, 3-7
- **New Solution:** 1-2-4-6-5-3-7-1 Distance = 64

- **Advantage**:
- This method is easy to integrate with other methods.

- **Disadvantage:**
- Tabu search approach is to climb the hill in the steepest direction and stop at top and then climb downwards to search for another hill to climb. The drawback is that a lot of iterations are spent climbing each hill rather than searching for tallest hill.

GRASP

- The GRASP metaheuristic is an iterative greedy heuristic to solve combinatorial optimization problems. It was introduced in 1989. Each iteration of the GRASP algorithm contains two steps:
- 1- construction and local search:In the construction step, a feasible solution is built using a randomized greedy algorithm.
- 2- a local search heuristic: the next step is applied from the constructed solution.

GRASP

Greedy Randomized Adaptive Search Procedure

Input: Number of iterations.

Repeat

s = Random-Greedy(seed) ; /* apply a randomized greedy heuristic */

s' = $Local - Search(s)$; /* apply a local search algorithm to the solution */

Until Stopping criteria /* e.g. a given number of iterations */

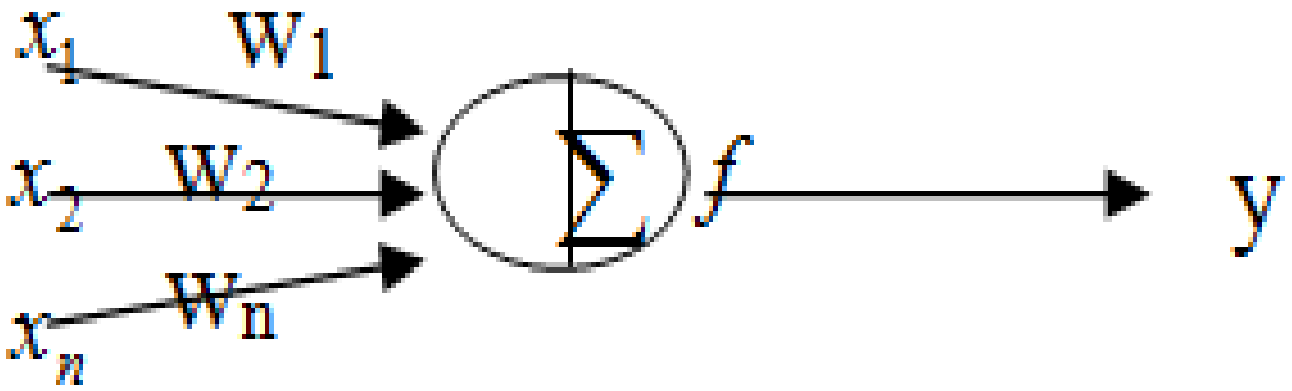
Output: Best solution found.

Artificial Neural Networks (ANN)

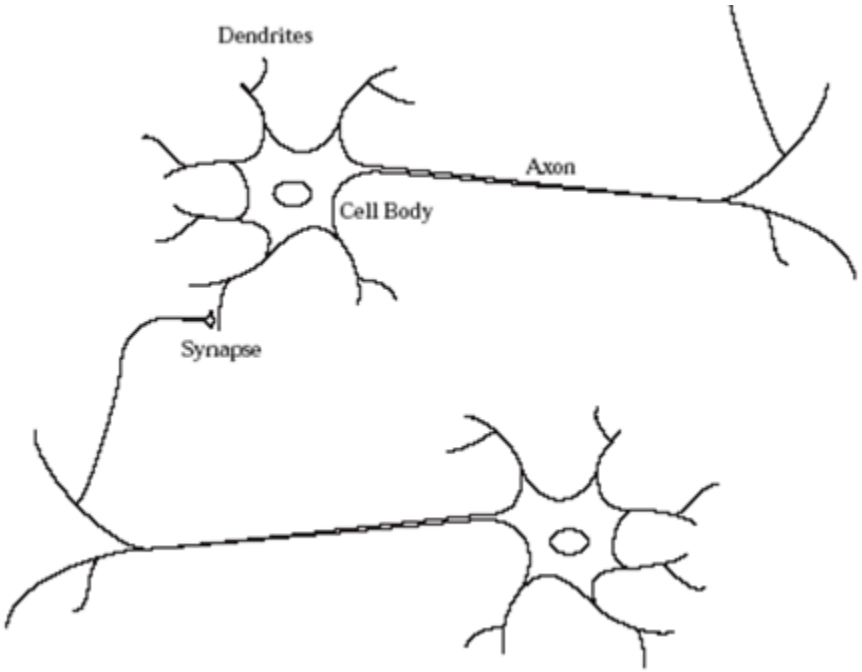
Theory of Neural Networks (NN)

- Human brain is the most complicated computing device known to a human being. The capability of thinking, remembering, and problem solving of the brain has inspired many scientists to model its operations. Neural network is an attempt to model the functionality of the brain in a simplified manner. These models attempt to achieve "good" performance via dense interconnections of simple computational elements. The term (ANN) and the connection of its models are typically used to distinguish them from biological network of neurons of living organism which can be represented systematically as shown in figure below:

Artificial Neural Network



Biological Neural Network



- ***Neclues*** is a simple processing unit which receives and combines signals from many other neurons through input paths called ***dendrites*** if the combined signal is strong enough, it activates the firing of neuron which produces an o/p signal. The path of the o/p signal is called the ***axon***, ***synapse*** is the junction between the (axon) of the neuron and the dendrites of the other neurons. The transmission across this junction is chemical in nature and the amount of signal transferred depends on the synaptic strength of the junction.
- This synoptic strength is modified when the brain is learning.
- **Weights (ANN) = synaptic strength (biological Networks)**

Artificial Neural Networks (ANN)

- An artificial neural network is an information processing system that has certain performance characters in common with biological neural networks.
- Artificial neural networks have been developed as generalizations of mathematical models of human cognition or neural biology, based on the assumptions that:-
- 1-Information processing occurs at many simple elements called neurons.
- 2-Signals are passed between neurons over connection links.
- 3-Each connection link has an associated weight which, in a typical neural net, multiplies the signal transmitted.
- 4-Each neuron applies an action function (usually nonlinear) to its net input
- (sum of weighted input signals) to determine its output signal.

A neural network is characterized by:

- 1- Architecture: - its pattern of connections between the neurons.
- 2- Training Learning Algorithm: - its method of determining the weights on the connections.
- 3- Activation function

Properties of ANN

- 1-Parallelism
- 2-Capacity for adaptation "learning rather programming"
- 3-Capacity of generalization
- 4-No problem definition
- 5- Abstraction & solving problem with noisy data.
- 6- Ease of constriction & learning.
- 7-Distributed memory
- 8- Fault tolerance

Type of learning

- **1:** A logical function to be represented is given. The input vector e1 , e2, e3, …., en are present, whom the output vectors a_1, a_2, a_3, …. , a_n assigned. These functions are to be represented by a network.
- **2:** A topology is to be selected for the network.
- **3:** The weights w1, w2, w3, …wn are to be selected in such a way that the network represents The given function (n) the selected topology. Learn procedures are to be used for determining the weights.
- **4:** After the weights have been learned and the network becomes available, it can be used as after as desired.

- **The learning of weights is generally done as follows:**
- 1- Set random numbers for all weights.
- 2- Select a random input vector e_j.
- 3- Calculate the output vector O_j with the current weights.
- 4- Compare O_j with the destination vector a_j , if $O_j = a_j$ then continue with (2).Else correct the weights according to a suitable correction formula and then continue with (2).

There are *three type* of learning in which the weights organize themselves according to the task to be learnt, these types are:-

- **1- Supervised learning:-**

  The supervised is that, at every step the system is informed about the

  exact output vector. The weights are changed according to a formula (e.g. the delta-rule), if o/p is unequal to a. This method can be compared to learning under a teacher, who knows the contents to be learned and regulates them accordingly in the learning procedure.

- **2-Unsupervised Learning:-**

  Here the correct final vector is not specified, but instead the weights are

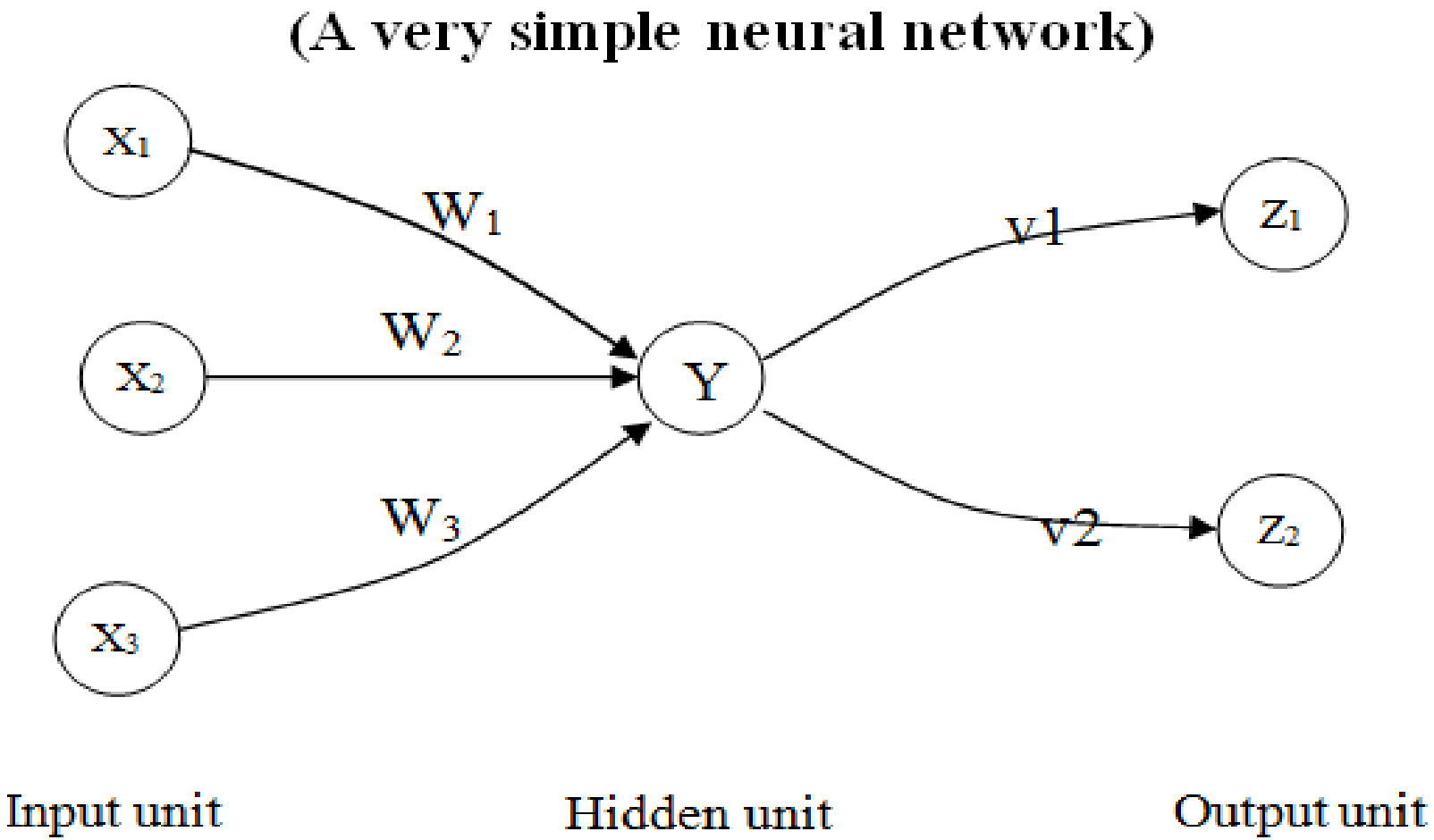
  changed through random numbers. With the help of an evaluation function one can ascertain whether the output calculated with the changed weights is better than the previous one. In this case the changed weights are stored, else forgotten. This type of learning is also called reinforcement learning.

3- Learning through self- organization:-

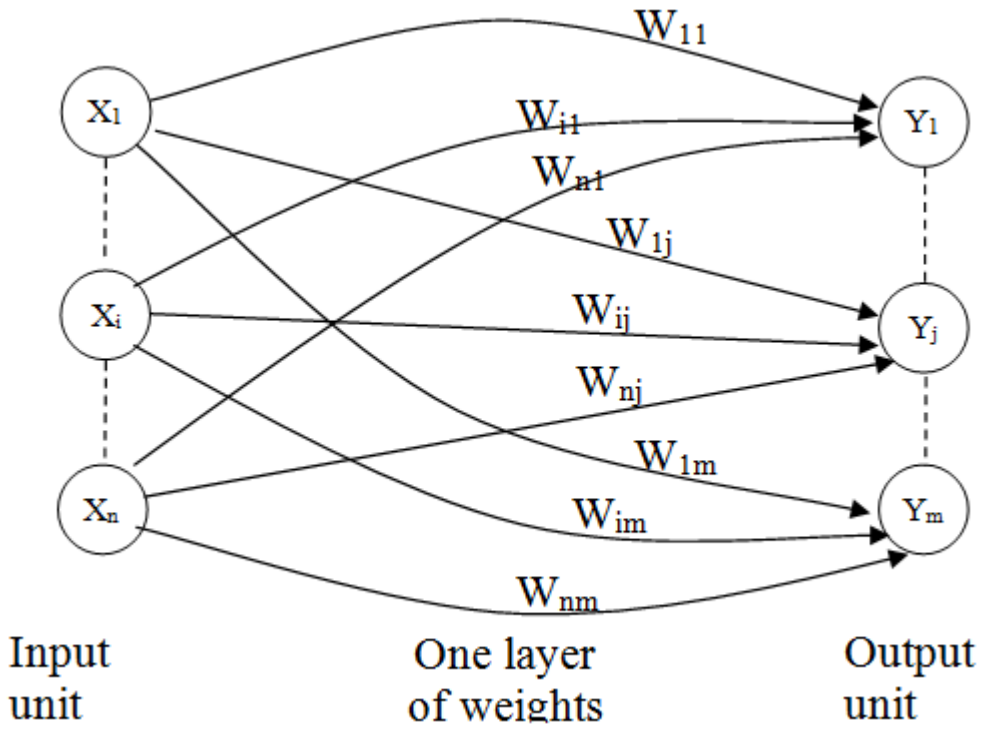
The weights changed themselves at every learning step. The change depends up on:

- 1- The neighborhood of the input pattern.
- 2- The probability pattern, with which the permissible input pattern is offered.

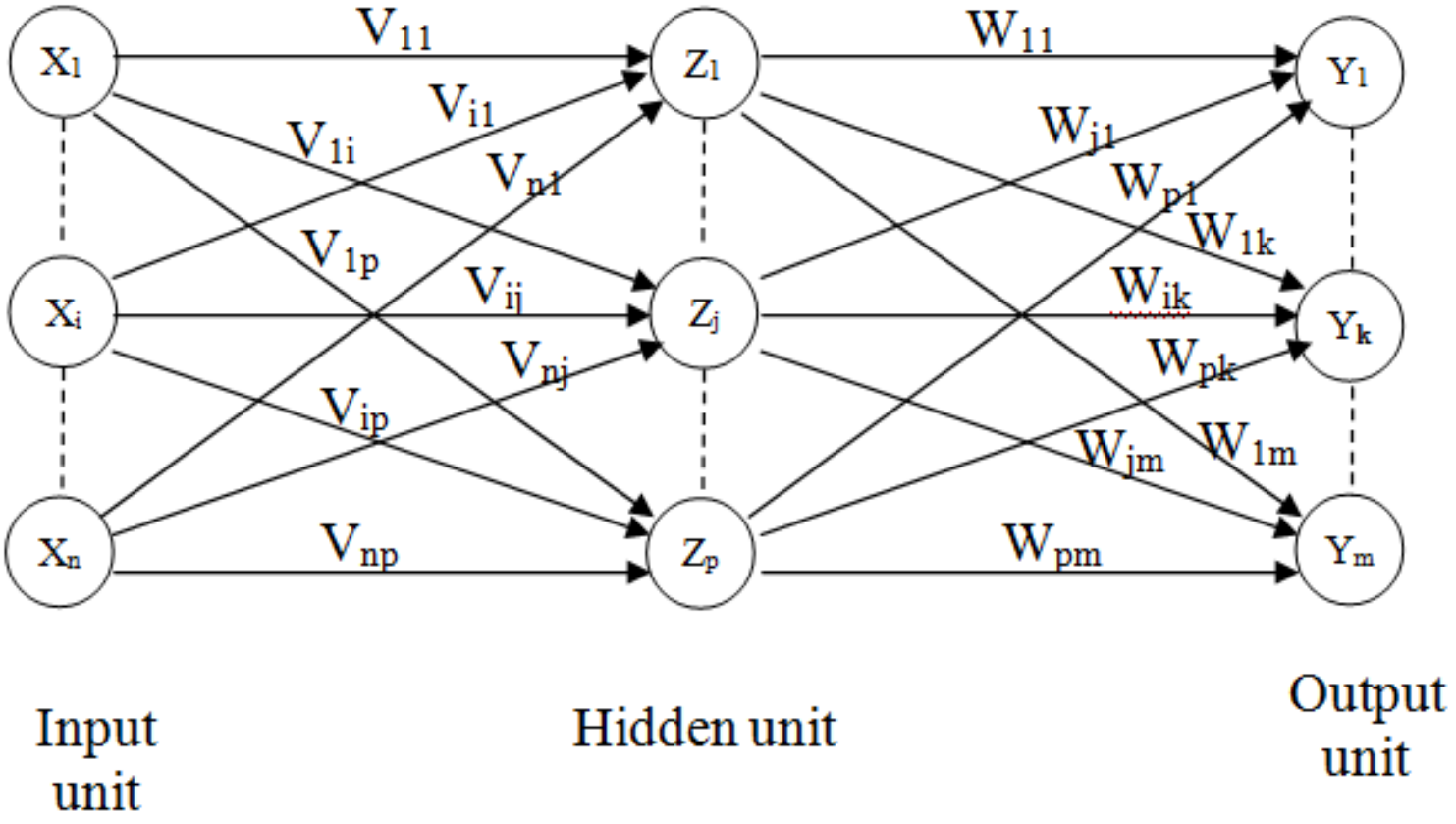
Typical Architecture of NN



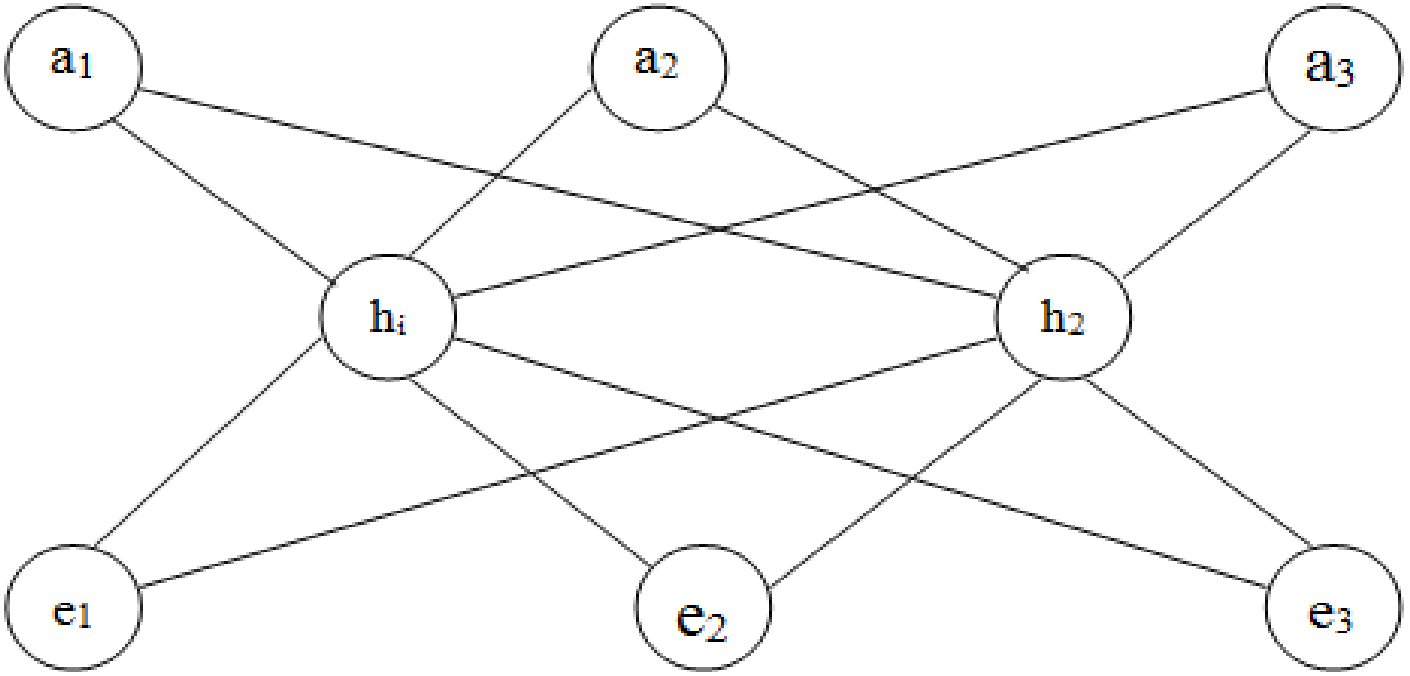
1 - Single-Layer Net



2 - Multilayer net



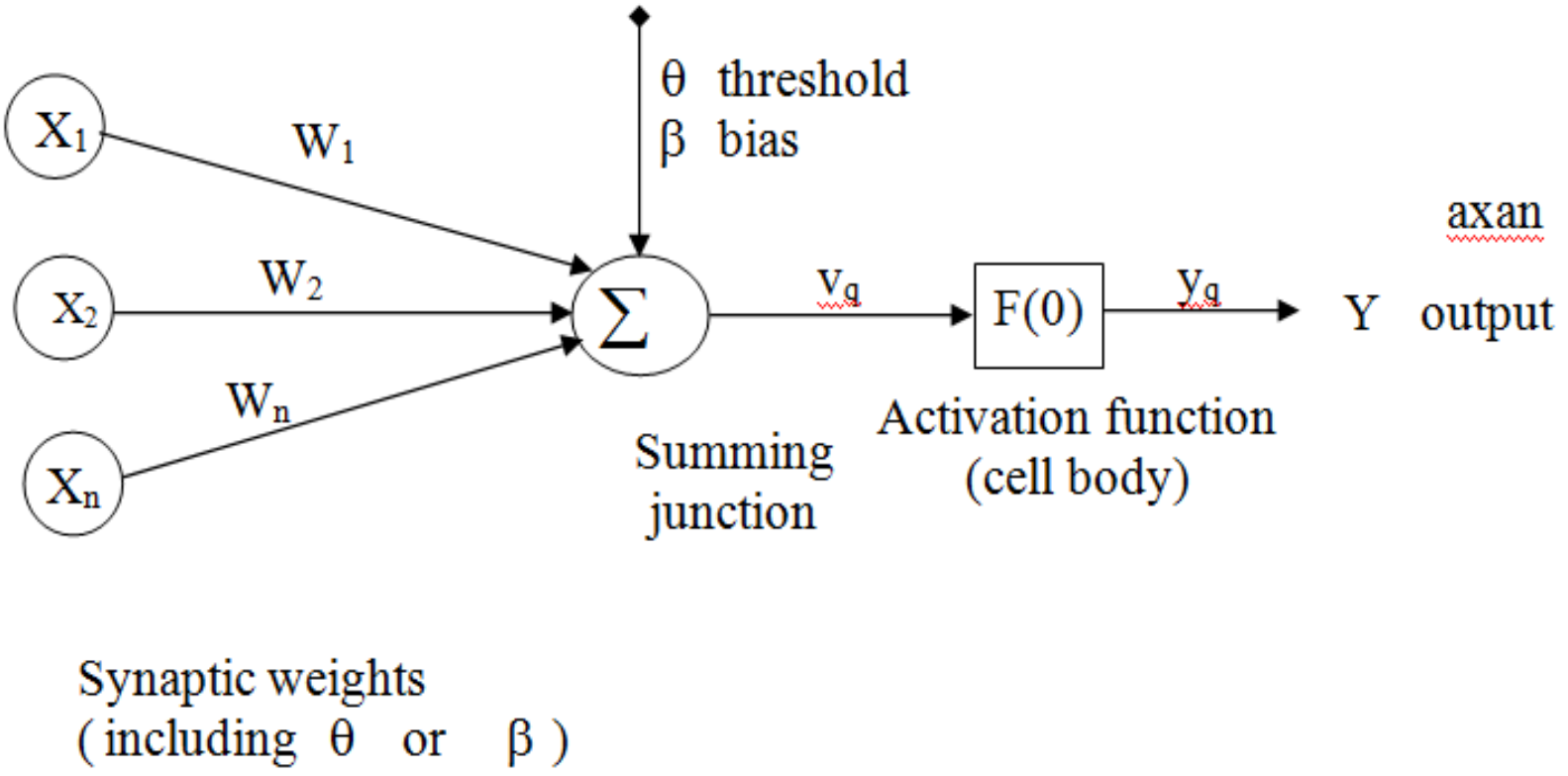
The figure shown bellow is an example of a three-layered neural net work with two hidden neurons.



Basic Activation Functions

The activation function (Sometimes called a transfers function) shown in figure below can be a linear or nonlinear function. There are many different types of activation functions. Selection of one type over another depends on the particular problem that the neuron (or neural network) is to solve. The most common types of activation function are:-

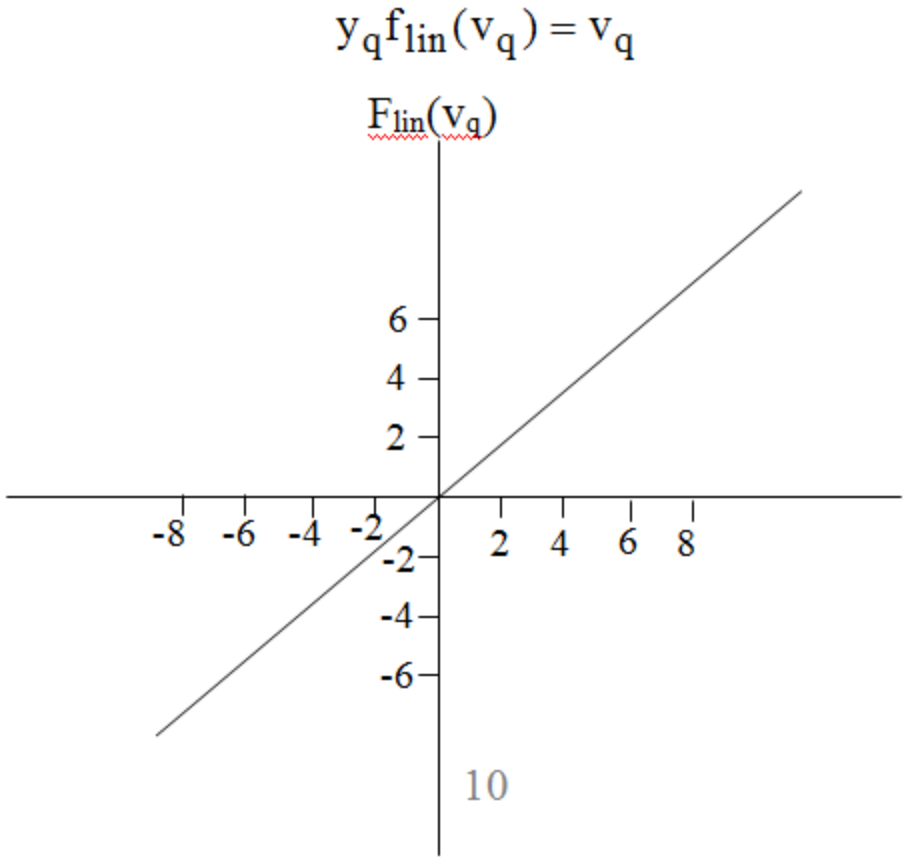
$$V_q = \sum_{v=0}^{n} W_{qj} X_j$$



Alternate nonlinear model of an ANN

ACTIVATION FUNCTION S

1- The first type is *the linear* (or *identity*) function. Ramp



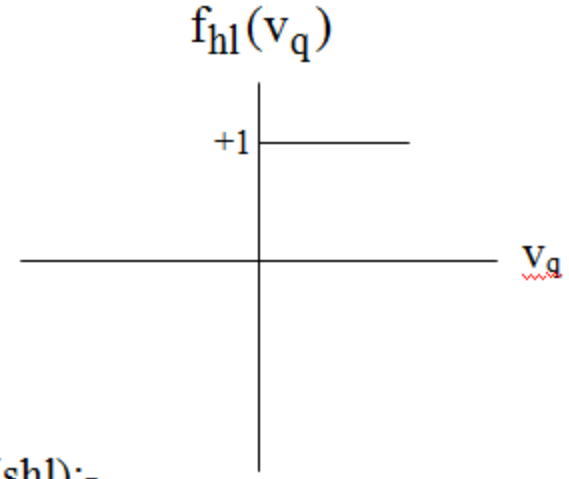
- **2-**The second type of activation function is a ***hard limiter***; this is a binary (or bipolar) function that hard-limits the input to the function to either a 0 or a 1 for the binary type, and a -1 or 1 for the bipolar type.
- The binary hard limiter is sometimes called the threshold function

and

- the bipolar hard limiter is referred to as the symmetric hard limiter.

a- The o/p of the binary hard limiter:-

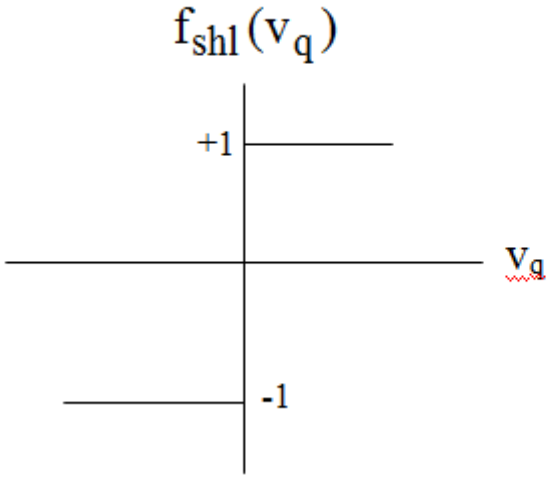
$$y_q = f_{hl}(v_q) = \begin{cases} 0 & \text{if } v_q < 0 \\ 1 & \text{if } v_q >= 0 \end{cases}$$



b-The o/p for the symmetric hard limiter (shl):-

$$y_q = f_{shl}(v_q) = \begin{cases} -1 & \text{if } v_q < 0 \\ 0 & \text{if } v_q = 0 \\ 1 & \text{if } v_q > 0 \end{cases}$$

also called double side



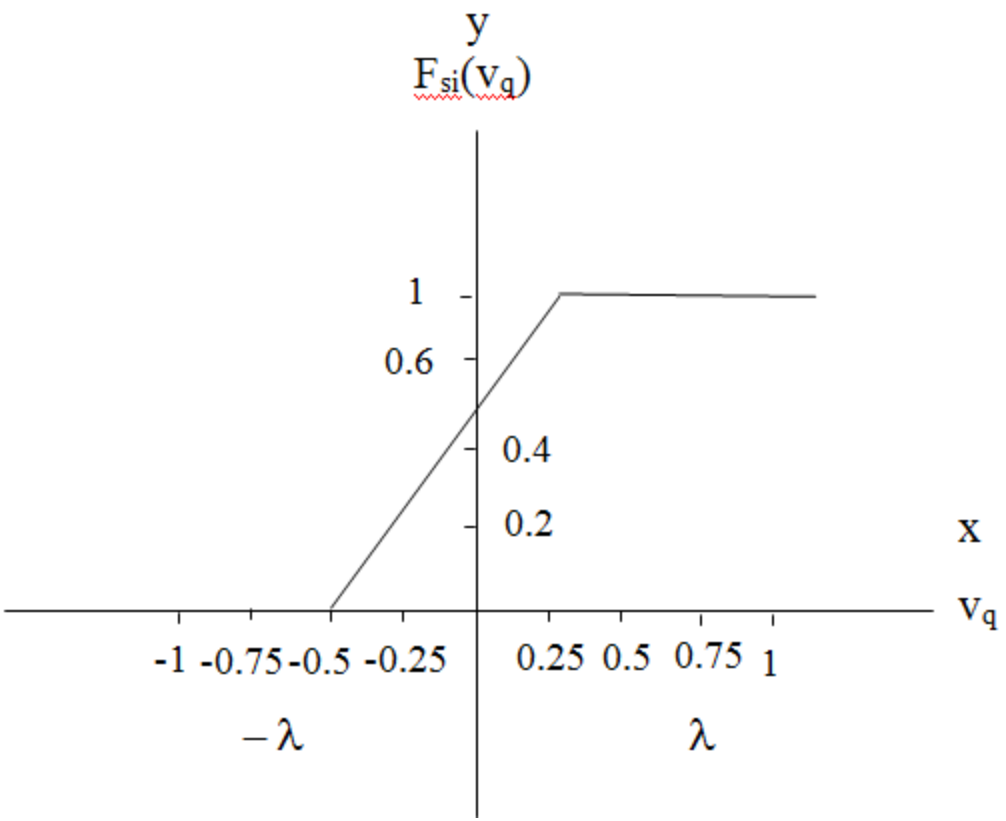
- **3-**The third type of basic activation function is the ***saturating linear*** function or threshold logic Unit (TLU) .
- This type of function can have either a binary or bipolar range for the saturation limits of the output. The bipolar saturating linear function will be referred to as the symmetric saturating linear function.

a- The o/p for the *saturating linear* function (binary o/p)

$$y_q = f_{sl}(v_q) = \begin{cases} 0 & \text{if } v_q < -1/2 \\ v_q + 1/2 & \text{if } -1/2 <= v_q <= 1/2 \\ 1 & \text{if } v_q > 1/2 \end{cases}$$

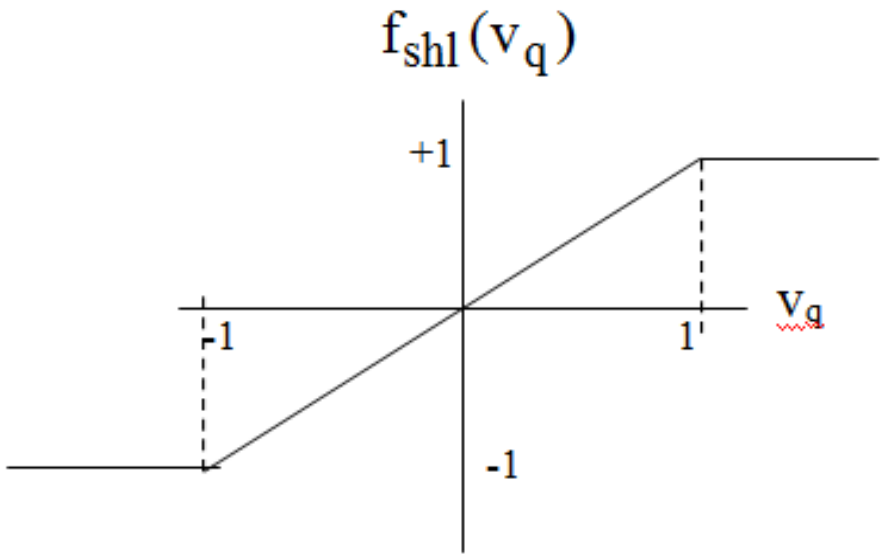
or

$$y = \begin{cases} \lambda & \text{if } x > \lambda \\ x & \text{if } -\lambda <= x <= \lambda \\ -\lambda & \text{if } < -\lambda \end{cases}$$


y
$F_{si}(v_q)$
1
0.6
0.4
0.2
x
v_q
-1
-0.75
-0.5
-0.25
0.25
0.5
0.75
1
$-\lambda$
λ

b- The o/p for the *symmetric saturating linear* function

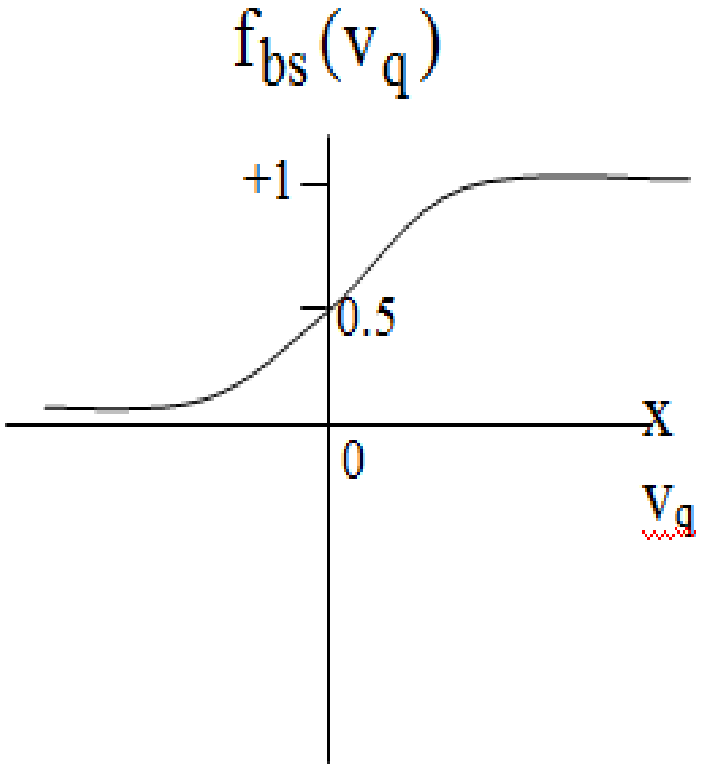
$$y_q = f_{ssl}(v_q) = \begin{cases} -1 & \text{if } v_q < -1 \\ v_q & \text{if } -1 <= v_q <= 1 \\ 1 & \text{if } v_q > 1 \end{cases}$$



4- The fourth type is *sigmoid*. Modern NN's use the sigmoid nonlinearity which is also known as logistic, semi linear, or squashing function.

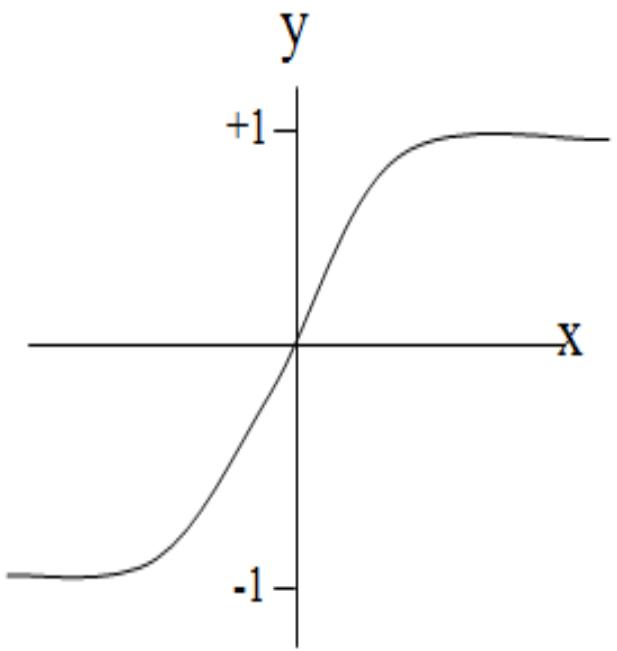
$$y_q = f_{bs}(v_q) = \frac{1}{1 + e^{-\propto v_q}}$$

$$y = \frac{1}{1 + e^{-x}}$$



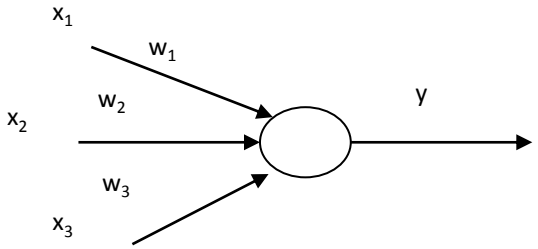
5-*Hyperbolic tangent* function is similar to sigmoid in shape but symmetric about the origin. (Tan h)

$$y = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$



Ex.1 find y for the following neuron if :-

$x_1=0.5,\ x_2=1,\ x_3=-0.7$ $w_1=0,\ w_2=-0.3,\ w_3=0.6.$



- **Sol**
- net =
- =0.5*0+1*-0.3+(-0.7*0.6)= -0.72
- 1- if f is linear
- y = -0.72
- 2- if f is hard limiter (on-off)
- y = -1

3-if f is sigmoid

$$y = \frac{1}{1 + e^{-(-0.72)}} = 0.32$$

4-if f is tan h

$$y = \frac{e^{-0.72} - e^{0.72}}{e^{-0.72} + e^{+0.72}} = -0.6169$$

5-if f is (TLU) with b=0.6, a=3 then y=-3

$$f(y) = \begin{cases} a & y > b \\ ky & -b < y < b \\ -a & y < -b \end{cases} \quad \Bigg| \quad f(y) = \begin{cases} a & y > b \\ ky & 0 < y < b \end{cases}$$

Ex2:- (H.W)

Find y for the following neuron if

$x_1 = 0.5,\ x_2 = 1, x_3 = -0.7$

$w_1 = 0,\ w_2 = -0.3, w_3 = 0.6$

$\theta = 1$

$$\text{Net} = \sum W_i X_i + \theta$$

$$= -0.72 + 1 = 0.28$$

1- if f is linear

$$y = 0.28$$

2- if f is hard limiter

$$y = 1$$

3-if f is sigmoid

$$y = \frac{1}{1 + e^{-0.28}} = 0.569$$

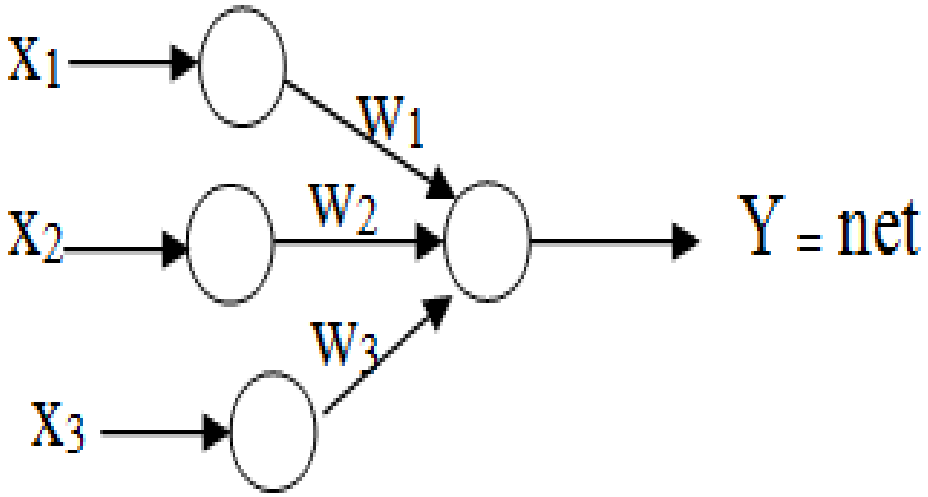
4-if f is tan sh

$$y = \frac{e^{0.28} - e^{-0.28}}{e^{0.28} + e^{0.28}} = 0.272$$

5-if f is TLU with b=0.6, a=3 , y=0.28

y=0.28 $y \leftarrow -b < y < b$

EX: The output of a simulated neural using a sigmoid function is 0.5 find the value of threshold when the input $x_1 = 1$, $x_2 = 1.5$, $x_3 = 2.5$. and have initial weights value = 0.2.



Solution

$$\text{Output} = F\ (\text{net} + \theta\ )$$

$$F(\text{net}) = \frac{1}{1 + e^{-\text{net}}}$$

$$\text{Net} = \sum W_i X_i$$

$$= X_1 W_1 + X_2 W_2 + X_3 W_3$$

$$= (1*0.2) + (1.5*0.2) + (2.5*0.2) = 0.2 + 0.30 + 0.50 = 1$$

$$0.5 = \frac{1}{1 + e^{-(1+\theta)}}$$

$$0.5\ (1 + e^{-(1+\theta)}) = 1$$

$$0.5 + 0.5\ e^{-(1+\theta)} = 1$$

$$0.5\ e^{-(1+\theta)} = 0.5$$

$$e^{-(1+\theta)} = 1$$

$$-(1+\theta) = \ln 1 \Rightarrow -1 - \theta = \theta \Rightarrow -\theta = 1 \Rightarrow \therefore \theta = -1$$

The Bias

Some networks employ a bias unit as part of every layer except the output layer.

This units have a constant activation value of 1 or -1, it's weight might be adjusted during learning. The bias unit provides a constant term in the weighted sum which results in an improvement on the convergence properties of the network.

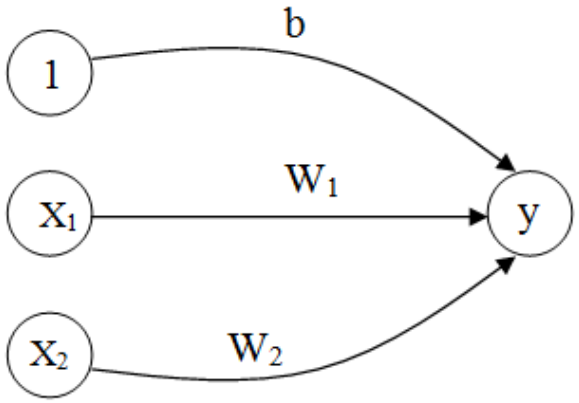
A bias acts exactly as a weight on a connection from a unit whose activation is always 1. Increasing the bias increases the net input to the unit. If a bias is included, the activation function is typically taken to be:

$$f(net)\begin{cases}1 & \text{if } net >= 0;\\ -1 & \text{if } net < 0;\end{cases}$$

Where

$$net = b + \sum_i X_i W_i$$

Figure: - single –layer NN for logic function



Some authors do not use a bias weight, but instead use a fixed threshold for the activation function.

$$f(net)\begin{cases} 1 & \text{if } net >= 0; \\ -1 & \text{if } net < 0; \end{cases}$$

Where

$$net = b + \sum_i X_i W_i$$

Learning Algorithms

- The NN's mimic the way that a child learns to identify shapes and colors NN algorithms are able to adapt continuously based on current results to improve performance. Adaptation or learning is an essential feature of NN's in order to handle the new "environments" that are continuously encountered. In contrast to NN's algorithms, traditional statistical techniques are not adoption but typically process all training data simultaneously before being used with new data. The performance of learning procedure depends on many factors such as:-
- 1- The choice of error function.
- 2- The net architecture.
- 3- Types of nodes and possible restrictions on the values of the weights.
- 4- An activation function.

The convergent of the net:-

- The convergent of the net depends on the:-

1-Training set

2-The initial conditions

3-Learning algorithms.

- **Note:-**

  The convergence in the case of complete information is better than in the case of incomplete information.

  Training a NN is to perform weights assignment in a net to minimize the o/p error. The net is said to be trained when convergence is achieved or in other words the weights stop changing.

- The learning rules are considered as various types

Hebbian Learning Rule

- The earliest and simplest learning rule for a neural net is generally known as the Hebb rule. Hebbian learning rule suggested by Hebb in 1949. Hebb's basic idea is that if a unit U_j receives an input from a unit U_i and both unite are highly active, then the weight W_{ij} (from unit i to unit j) should be strengthened.
- This idea is formulated as:-

$$\Delta w_{ij} = \zeta \; x_i y_j$$

Where ζ is the learning rate $\zeta = \propto = 1$, Δw is the weight change

$$w(new) = w(old) + xy$$

$$\therefore \quad w(new) = w(old) + \Delta w$$

Advantage of Hebbian learning

- تسمح بتقليل قيم الـ weight عندما الـ I/p يكون active والـ o/p active وهذه من السهولة يتم انجازها باستخدام 1- and +1 بدلا من 0 and 1 .

Disadventage of Hebbian learning -

- Hebbian learning takes no account of the actual value of the output, only the desired value. This limitation can be overcome if the weights are adjusted by amount which depends upon the error between the desired and actual output. This error is called delta, S, and the new learning rule is called the delta rule.

(Hebbian learning Rule) Algorithm

- **Step 0:** Initialize all weights
- $w_i = 0$ (i =1 to n)
- **Step 1:** for each I/p training vector target o/p
- Pair. S : t do steps 2- 4.
- **Step 2 :** Set activations for I/P units:
- $w_i = s_i$ (i =1 to n)
- **Step 3 :** set activation for O/P unit :
- $y = t$
- **Step 4 :** Adjust the weights for
- w_i (new) = w_i(old) + $x_i y$ (i =1 to n)
- Adjust the bias:
- b(new) = b(old) + y
- Note that the bias is adjusted exactly like a weight from a "unit" whose output signal is always 1.

Ex : A Hebbian net for the AND function: binary input and targets

Input		Target	
1	1	1	1
1	0	1	0
0	1	1	0
0	0	1	0

$\Delta w_1 = x_1 y, \quad \Delta w_2 = x_2 y, \quad \Delta b = y$

Initial weights = 0, $w_1 = 0$, $w_2 = 0$, $w_3 = 0$

	x_1	x_2	b	y	Δw_1	Δw_2	Δb	w_1 0	w_2 0	b 0
1	1	1	1	1	1	1	1	1	1	1
2	1	0	1	0	0	0	0	1	1	1
3	0	1	1	0	0	0	0	1	1	1
4	0	0	1	0	0	0	0	1	1	1

- The first input pattern shows that the response will be correct presenting the second, third, and fourth training i/p shows that because the target value is 0, no learning occurs. Thus, using binary target values prevents the net from learning only pattern for which the target is “off”.

- The AND function can be solved if we modify its representation to express the inputs as well as the targets in bipolar form. Bipolar representation of the inputs and targets allows modifications of a weight when the input unit and the target value are both "on" at the same time and when they are both "off" at the same time and all units will learn whenever there is an error in the output. The Hebb net for the AND function: bipolar inputs and targets are:

Replace the 0 with -1

$$\Delta w_1 = x_1 * y$$
$$= 1 * 1 = 1$$
$$w_1(\text{new}) = w_1(\text{old}) + \Delta w_1$$
$$= 0 + 1 = 1$$

x_1	x_2	b	y
1	1	1	1
1	-1	1	-1
-1	1	1	-1
-1	-1	1	-1

Presenting the first input:-

x_1	x_2	b	y	Δw_1	Δw_2	Δb	w_1 0	w_2 0	b 0
1	1	1	1	1	1	1	1	1	1

Presenting the second input:-

x_1	x_2	b	y	Δw_1	Δw_2	Δb	w_1 1	w_2 1	b 1
1	-1	1	-1	-1	1	-1	0	2	0

Presenting the third input:-

x_1	x_2	b	y	Δw_1	Δw_2	Δb	w_1 0	w_2 2	b 0
-1	1	1	-1	1	-1	-1	1	1	-1

Presenting the fourth input:-

x_1	x_2	b	y	Δw_1	Δw_2	Δb	w_1 1	w_2 1	b -1
-1	-1	1	-1	1	1	-1	2	2	-2

The first iteration will be:-

Input			target	Weight change			weights		
x_1	x_2	b	y	Δw_1	Δw_2	Δb	w_1 0	w_2 0	b 0
1	1	1	1	1	1	1	1	1	1
1	-1	1	-1	-1	1	-1	0	2	0
-1	1	1	-1	1	-1	-1	1	1	-1
-1	-1	1	-1	1	1	-1	2	2	-2

Second Method

$$W_{ij} = \sum X_i Y_j \qquad \textit{or} \qquad [W] = [X]^T [Y]$$

Basic Delta Rule (BDR)

The idea of Hebb was modified to produce the widrow-Hoff (delta) rule in 1960 or least Mean Square (LMS). The BDR is formulated as:-

$$\Delta w_{ij} = \xi(d_j - y_i)\ x_i$$

$$\Delta w_{ij} = \xi\ \delta_j\ x_i \quad \text{(Delta rule)}$$

Δw : - is the weight change

ξ : - is the learning rate

d :- desired output

y :- actual output

δ : - error between d and y

Note

- Before training the net, a decision has to be made on the setting of the learning rate. Theoretically, the larger the faster training process goes. But practically, may have to be set to a small value for example (0.1) in order to prevent the training process from being trapped at local minimum resulting at oscillatory behavior.

Back Propagation

The determination of the error is a recursive process which start with the o/p units and the error is back propagated to the I/p units. Therefore the rule is called error Back propagation (EBP) or simply Back Propagation (BP). The weight is changed exactly in the same form of the standard DR.

$$\Delta w_{ij} = \xi \ \delta_j \ x_i$$

$$w_{ij}(t+1) = w_{ij}(t) + \xi \ \delta_j \ x_i$$

There are two other equations that specify the error signal. If a unite is an o/p unit, the error signal is given by:-

Where

$$\delta = (d_j - y_j)\ f_j(\text{net}\ \ j)$$

$$\text{Where}\quad \text{net}\quad j\ = \sum w_{ij}\, x_i + \theta$$

The GDR minimize the squares of the differences between the actual and the desired o/p values summed over the o/p unit and all pairs of I/p and o/p vectors. The rule minimize the overall error by implementing a gradient descent in E: -

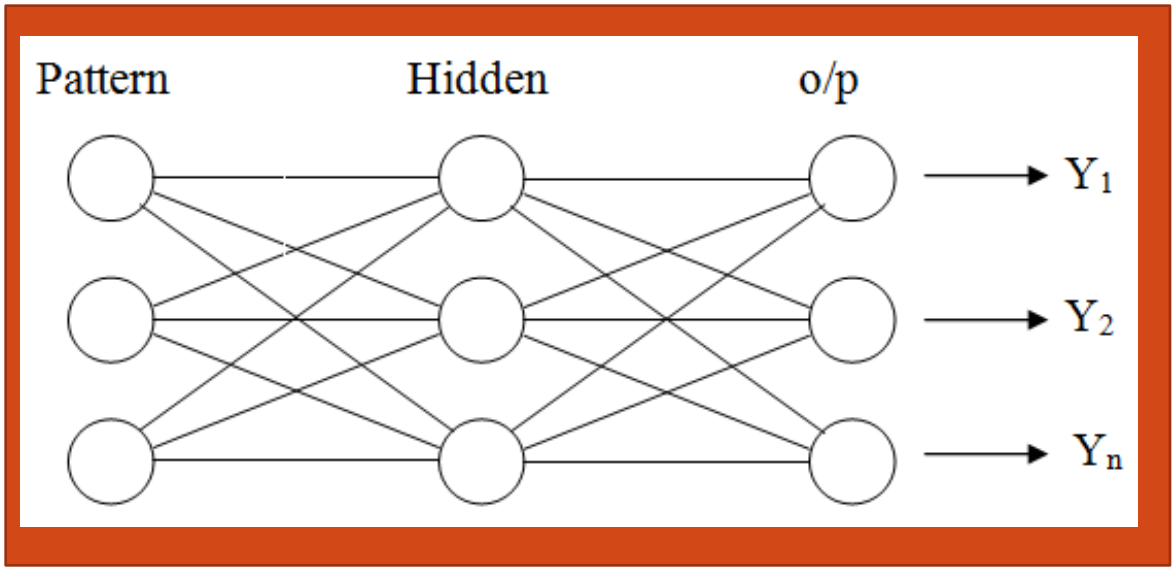
- Where

$$E_p = 1/2 \sum_j (d_j - y_j)^2$$

The BP consists of two phases:-

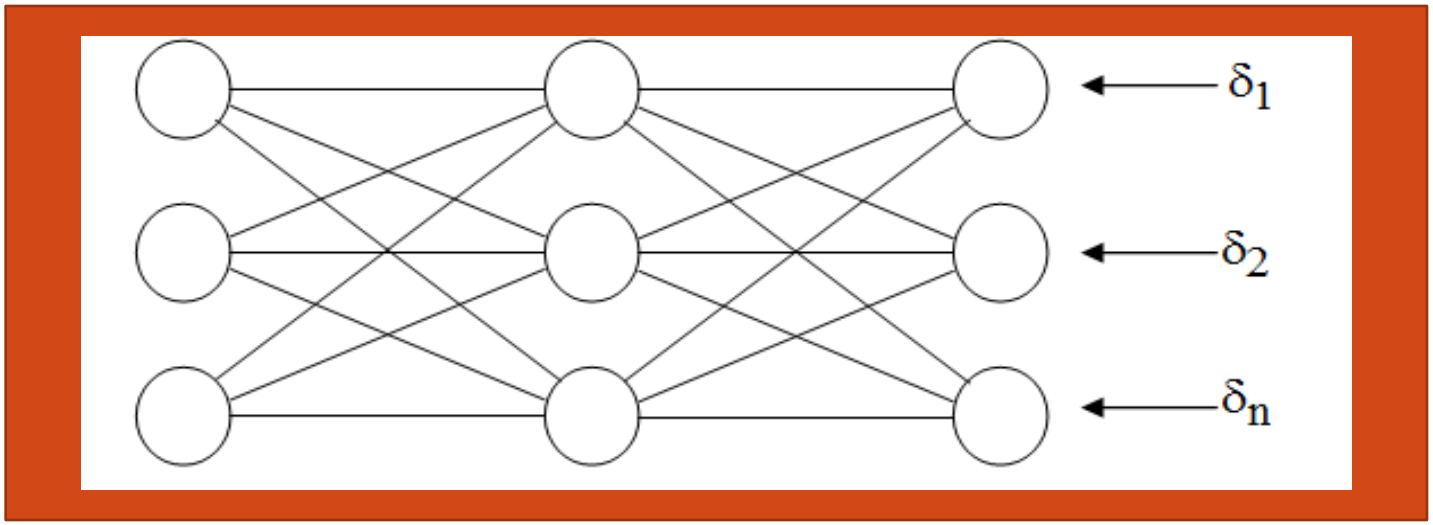
1- Forward Propagation:-

During the forward phase, the I/p is presented and propagated towards the o/p.



2- Backward Propagation:-

- During the backward phase, the *errors* are formed at the o/p and propagated towards the I/p.



3- Compute the error in the hidden layer.

$$\text{If } y = f(x) = \frac{1}{1+e^{-x}}$$

$$f' = y(1-y)$$

Equation is can rewrite as:-

$$\delta_j = y(1-y)(d_j - y_j)$$

The error signal for hidden units for which there is no specified target (desired o/p) is determined recursively in terms of the error signals of the units to which it directly connects and the weights of those connections:-

That is

$$\delta_j = f'(net_j)\sum_k \delta_k w_{ik}$$

Or

$$\delta_j = y_j(1-y_j)\sum_k \delta_k w_{ik}$$

Computation of error

Back Propagation Training Algorithm

Training a network by back propagation involves three stages:-

1-the feed forward of the input training pattern

2-the back propagation of the associated error

3-the adjustment of the weights

let n = number of input units in input layer,

let p = number of hidden units in hidden layer

let m = number of output units in output layer

let V_{ij} be the weights between i/p layer and the hidden layer,

let W_{ij} be the weights between hidden layer and the output layer,

we refer to the i/p units as X_i , i=1, 2, ….,n. and we refer to the hidden units as Z_j , j=1,….,p. and we refer to the o/p units as y_k, k=1,….., m.

δ_{1j} is the error in hidden layer,

δ_{2k} is the error in output layer,

ζ is the learning rate

$\propto$ is the momentum coefficient (learning coefficient, $0.0 < \propto < 1.0$,

y_k is the o/p of the net (o/p layer),

Z_j is the o/p of the hidden layer,

X_i is the o/p of the i/p layer.

η is the learning coefficient.

The algorithm is as following :-

Step 0 : initialize weights (set to small random value).

Step 1 : while stopping condition is false do steps 2-9

Step 2: for each training pair, do steps 3-8

Feed forward :-

Step 3:- Each i/p unit (X_i) receives i/p signal X_i & broad casts this signal to all units in the layer above (the hidden layer)

Step 4:- Each hidden unit (Z_j) sums its weighted i/p signals,

$$Z - inj = Vaj + \sum_{i=1}^{n} x_i v_{ij} \quad (Vaj \text{ is abias})$$

and applies its activation function to compute its output signal (the activation function is the binary sigmoid function),

$$Z_j f(Z - inj) = 1 \;/\; (1 + \exp - (Z - inj))$$

Hidden layer law

and sends this signal to all units in the layer above (the o/p layer).

Step 5:- Each output unit (Yk)sums its weighted i/p signals,

$$y - ink = wok + \sum_{j=1}^{p} Zjwjk \quad (\text{where wok is abias})$$

and applies its activation function to compute its output signal.

$$y_k = f(y - ink) = 1/(1 + \exp - (y - ink))$$

Output layer law

back propagation of error:-

step 6 : Each output unit (y_k , k= 1 tom) receive a target pattern corresponding to the input training pattern, computes its error information term and calculates its weights correction term used to update W_{jk} later,

$$\delta_{2k} = y_k(1-y_k)*(T_k - y_k),$$

Error computation at output layer

where T_k is the target pattern & k=1 to m .

step 7 : Each hidden unit (Z_j, j= 1 top) computes its error information term and calculates its weight correction term used to update Vij later,

$$\delta_{1j} = Zj*(1-Zj)*\sum_{k=1}^{m} \delta 2kWjk$$

Error computation at hidden layer

step 8: Each output unit (y_k, k =1 tom) upda

$$Wjk(new) = \eta * \delta 2k * Zj + \propto *[Wjk(dd)],$$

j= 1 to p

Each hidden unit (Z_j, j= 1 to p) update its bias and weights:

$$Vij(new) = \eta * \delta 1j * Xi + \propto [vij(dd)],$$

I = 1 to n

Step 9 : Test stopping condition.

New weight computation

New vector computation

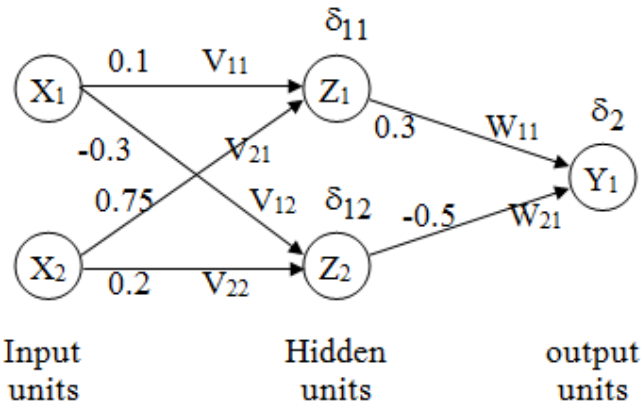
EX:

Suppose you have BP- ANN with 2-input , 2-hiddden , 1-output nodes with sigmoid function and the following matrices weight, trace with 1-iteration.

$$V = \begin{bmatrix} 0.1 & -0.3 \\ 0.75 & 0.2 \end{bmatrix} \qquad w = \begin{bmatrix} 0.3 & -0.5 \end{bmatrix}$$

Where $\propto = 0.9$, $\eta = 0.45$, $x = (1,0)$, and $T_k = 1$

Solution:-



1-Forword phase :-

$$Z-in1 = X_1V_{11} + X_2V_{21} = 1*0.1 + 0*0.75 = 0.1$$

$$Z-in2 = X_1V_{12} + X_2V_{22} = 1*-0.3 + 0*0.2 = -0.3$$

$$Z_1 = f(Z-in1) = 1/(1+\exp-(Z-in1)) = 0.5$$

$$Z_2 = f(Z-in2) = 1/(1+\exp-(Z-in2)) = 0.426$$

$$y-in1 = Z_1W_{11} + Z_2W_{21}$$

$$= 0.5*0.3 + 0.426*(-0.5) = -0.063$$

$$y_1 = f(y-in1) = 1/(1+\exp-(y-in1) = 0.484$$

2-Backward phase :-

$$\delta_2 k = yk(1-yk)*(Tk-yk)$$

$$\delta_{21} = 0.484(1-0.484)*(1-0.484)0.129$$

$$\delta_{1j} = Z_j*(1-Z_j)*\sum_{k=1}^{m}\delta_{2k}W_{jk}$$

$$\delta_{11} = Z_1(1-Z_1)*(\delta_{21}W_{11})$$

$$= 0.5\,(1-0.5)*(0.129*0.3) = 0.0097$$

$$\delta_{12} = Z_2(1-Z_2)*(\delta_{21}W_{21})$$

$$= 0.426(1-0.426)*(0.129*(-0.5)) = -0.015$$

3-Update weights:-

$$W_{jk}(new) = \eta * \delta_{2k} * Z_j + \propto * [W_{jk}(old)]$$

$$W_{11} = \eta * \delta_{21} * Z_1 + \propto * [W_{11}(old)]$$

$$= 0.45 * 0.129 * 0.5 + 0.9 * 0.3 = 0.299$$

$$W_{21} = \eta * \delta_{21} * Z_2 + \propto * [W_{21}(old)]$$

$$= 0.45 * 0.129 * 0.426 + 0.9 * -0.5 = -0.4253$$

$$V_{ij}(new) = \eta * \delta_{1j} * X_i + \propto * [V_{ij}(old)]$$

$$V_{11} = \eta * \delta_{11} * X_1 + \propto * [V_{11}(old)]$$

$$= 0.45 * 0.0097 * 1 + 0.9 * 0.1 = 0.0944$$

$$V_{12} = \eta * \delta_{12} * X_1 + \propto * [V_{12}(old)]$$

$$= 0.45 * 0.0158 * 1 + 0.9 * -0.3 = -0.2771$$

$$V_{21} = \eta * \delta_{11} * X_2 + \propto * [V_{21}(old)]$$

$$= 0.45 * 0.0097 * 0 + 0.9 * 0.75 = 0.675$$

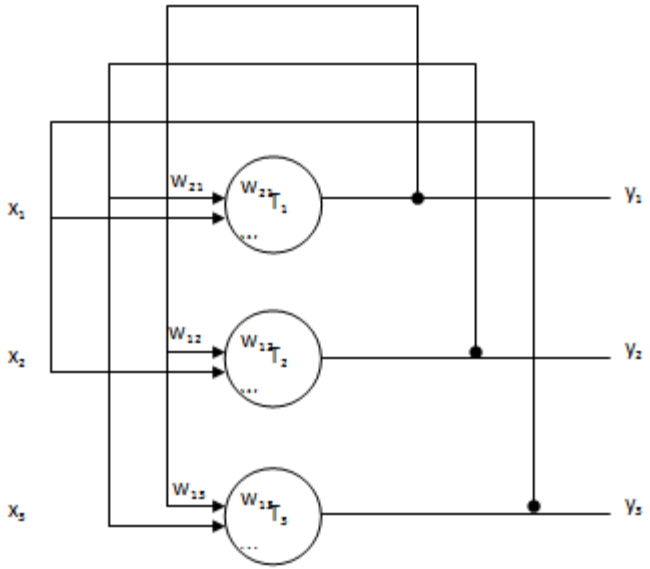
$$V_{22} = \eta * \delta_{12} * X_2 + \propto * [V_{22}(old)]$$

$$= 0.45 * -0.0158 * 0 + 0.9 * 0.2 = 0.18$$

$$\therefore V = \begin{bmatrix} 0.0944 & -0.2771 \\ 0.675 & 0.18 \end{bmatrix} \qquad W = [0.299 \quad -0.4253]$$

The Hopfield Network

- The Nobel prize winner (in physics) John Hopfield has developed the discrete Hopfield net in (1982-1984). The net is a fully interconnected neural net, in the sense that each unit is connected to every other unit. The discrete Hopfield net has symmetric weights with no self-connections, i.e,
- And
- In this NN, inputs of 0 or 1 are usually used, but the weights are initially calculated after converting the inputs to -1 or +1 respectively.



“The Hopfield network“

- The outputs of the Hopfield are connected to the inputs as shown in Figure, Thus feedback has been introduced into the network. The present output pattern is no longer solely dependent on the present inputs, but is also dependent on the previous outputs. Therefore the network can be said to have some sort of memory, also the Hopfield network has only one layer of neurons.
- The response of an individual neuron in the network is given by :-
- This means that for the jth neuron, the inputs from all other neurons are weighted and summed.

- **Note** , which means that the output of each neuron is connected to the input of every other neuron, but not to itself. The output is a hard-limiter which gives a 1 output if the weighted sum is greater than T_j and an output of 0 if the weighted sum is less than T_j. it will be assumed that the output does not change when the weighted sum is equal to T_j.
- Thresholds also need to be calculated. This could be included in the matrix by assuming that there is an additional neuron, called neuron 0, which is permanently stuck at 1. All other neurons have input connections to this neuron's output with weight W01, W02, W03,…etc. this provides an offset which is added to the weighted sum. The relation ship between the offset and the threshold T_j is therefore:-
- The output [y] is just the output of neuron 0 which is permanently stuck at 1, so the formula becomes:-

Ex1: if the patterns x1=[1100] and [0101] are to be stored, first convert them to
x1=[-1 -1 1 1]
x2=[-1 1 -1 1]

To find the threshold:-

1- The matrix $\begin{bmatrix} -1 & -1 & 1 & 1 \\ -1 & 1 & -1 & 1 \end{bmatrix}$

2-The transpose of the matrix is $\begin{bmatrix} -1 & -1 \\ -1 & 1 \\ 1 & -1 \\ 1 & 1 \end{bmatrix}$

3- y_0 is permanently stuck at +1 , so the offsets are calculated as follows

$$W_0 = \begin{bmatrix} -1 & -1 \\ -1 & +1 \\ +1 & -1 \\ +1 & +1 \end{bmatrix} \begin{bmatrix} +1 \\ +1 \end{bmatrix} = \begin{bmatrix} -2 \\ 0 \\ 0 \\ +2 \end{bmatrix}$$

4-These weights could be converted to thresholds to give:-

$$T_1 = 2$$
$$T_2 = 0$$
$$T_3 = 0$$
$$T_4 = -2$$

$$Tj = -W0j$$

EX2:-

Consider the following samples are stored in a net:-

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} -1 & +1 & -1 & -1 \\ +1 & +1 & -1 & -1 \\ -1 & -1 & +1 & +1 \end{bmatrix}$$

binary $\rightarrow$ convert $\rightarrow$ bipolar

The binary input is (1110). We want the net to know which of samples is the i/p near to?

Note :-

A binary Hopfield net can be used to determine whether an input vector is a "known" vector (i.e., one that was stored in the net) or "unknown" vector.

Solution:- 1-use Hebb rule to find the weights matrix

$$W = \begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \\ w_{31} & w_{32} & w_{33} & w_{34} \\ w_{41} & w_{42} & w_{43} & w_{44} \end{bmatrix}$$

$w_{ii}=0$ (diagonal)

$w_{ij}=w_{ji}$

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & w_{12} & w_{13} & w_{14} \\ w_{21} & 0 & w_{23} & w_{24} \\ w_{31} & w_{32} & 0 & w_{34} \\ w_{41} & w_{42} & w_{43} & 0 \end{bmatrix} \end{array}$$

$$w_{12} = (-1*1) + (1*1) + (-1*-1) = 1$$
$$w_{13} = (-1*-1) + (1*-1) + (-1*1) = -1$$
$$w_{14} = (-1*-1) + (1*-1) + (-1*1) = -1$$

$$w_{21} = w_{12} = 1$$
$$w_{23} = (1*-1) + (1*-1) + (-1*1) = -3$$
$$w_{24} = (-1*-1) + (1*-1) + (-1*1) = -3$$

$$w_{31} = w_{32} = 1$$
$$w_{32} = w_{23} = -3$$
$$w_{34} = (-1*-1) + (-1*-1) + (1*1) = 3$$

$$w_{41} = w_{14} = -1$$
$$w_{42} = w_{24} = -3$$
$$w_{43} = w_{34} = 3$$

$$\therefore W = \begin{bmatrix} 0 & 1 & -1 & -1 \\ 1 & 0 & -3 & -3 \\ -1 & -3 & 0 & 3 \\ -1 & -3 & 3 & 0 \end{bmatrix}$$

2-The i/p vector x = (1 1 1 0). For this vector, y=(1 1 1 0)

Choose unit y_1 to update its activation

$$y\_in1 = x_1 + \sum_j y_j w_{j1}$$

$$y\_in1 = 1 + [(0*1) + (1*1) + (-1*1) + (-1*0)]$$
$$= 1 + 0 = 1$$
$$\therefore y = (1110)$$

Choose unit y_2 to up date its activation:-

$$y\_in2 = x_2 + \sum_j y_j w_{j2}$$

$$= 1 + [(1*1) + (1*0) + (1*-3) + (0*-3)]$$
$$= 1 + (-2) = -1$$
$$y\_in2 < 0 \qquad \therefore y_2 = 0$$
$$\therefore y = (1010)$$

Choose unit y_3 to update its activation:-

$$y\_in3 = x_3 + \sum_j y_j w_{j3}$$

$= 1 + [(1 * -1) + (1 * -3) + (1 * 0) + (0 * 3)]$

$= 1 + (-4) = -3$

$y\_in3 < 0 \quad \therefore y_3 = 0$

$\therefore y = (1000)$

Choose unit y_4 to update its activation:-

$y\_in4 = x_4 + \sum_i y_i w_{i4}$

$= 0 + [(1 * -1) + (1 * -3) + (1 * 3) + (0 * 0)]$

$= 0 + (-1) \quad = -1$

y_in4 < 0 y4=0

y = (1000)

3- Test for convergence, false

∴ The input vector x = (1000), for this vector,

Y= (1 0 0 0)

$y\_in1 = 1$

$y\_in2 = 1$

$y\_in3 = -1 = 0$

$y\_in4 = -1 = 0$

$\therefore y = (1100)$

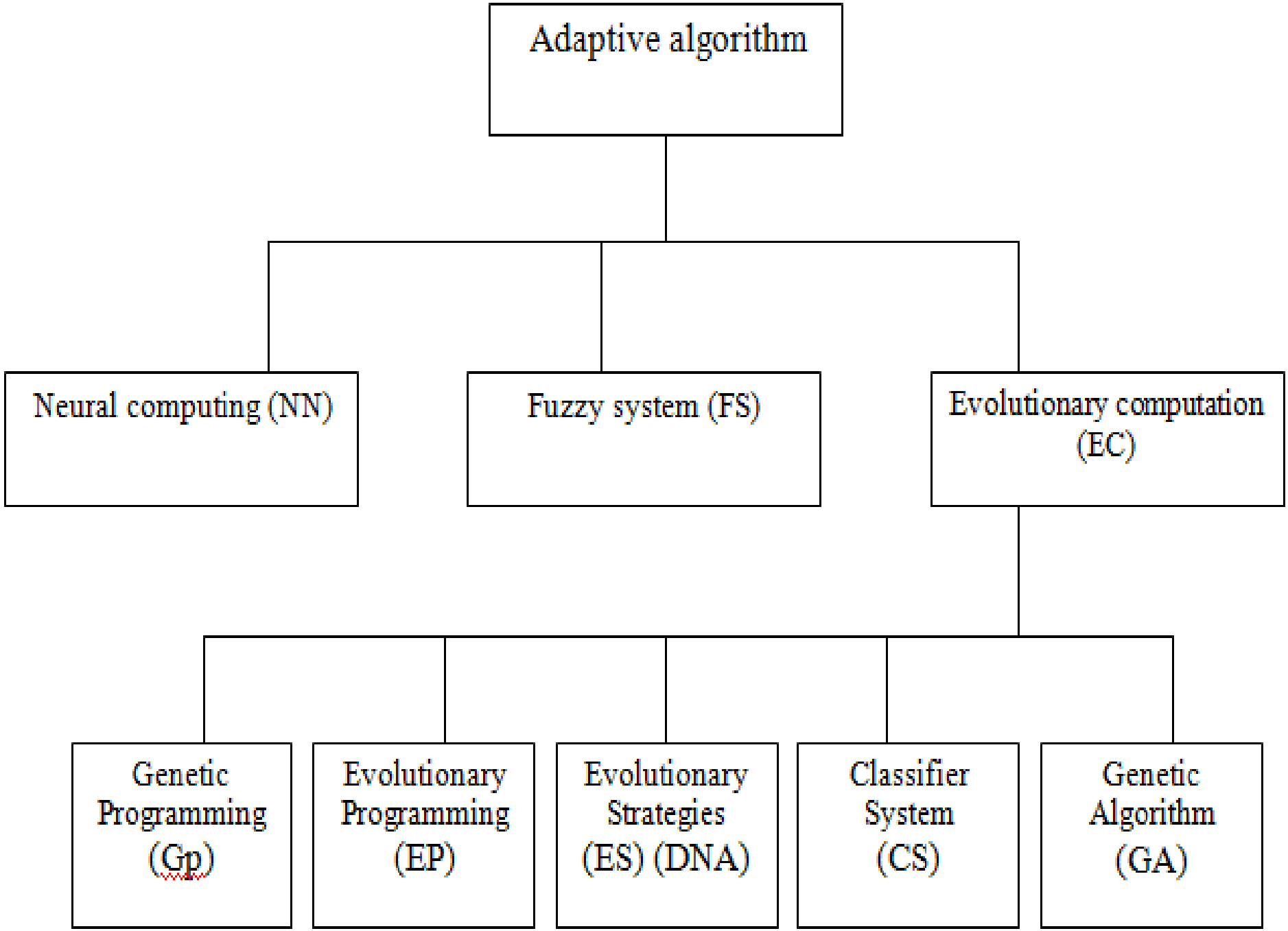
∴ The input vector x= (1 1 0 0)

Y= (1 1 0 0)

$$y - in1 = 2 = 1$$
$$y - in2 = 2 = 1$$
$$y - in3 = -4 = 0$$
$$y - in4 = -4 = 0$$
$$\therefore y = (1100)$$

- The input is near to the second sample.
- True.
- Stop.

Genetic Algorithms (GA)


Adaptive algorithm
Neural computing (NN)
Fuzzy system (FS)
Evolutionary computation (EC)
Genetic Programming (Gp)
Evolutionary Programming (EP)
Evolutionary Strategies (ES) (DNA)
Classifier System (CS)
Genetic Algorithm (GA)

Genetic Algorithms (GA)

- A genetic algorithm is a search procedure modelled on the mechanics of natural selection rather than a simulated reasoning process. Domain Knowledge is embedded in the abstract representation of a candidate solution termed an organism. Organisms are grouped into sets called populations. Successive population are called generation. The aim of GA is search for goal.
- A generational GA creates an initial generation G(0) , and for each generation ,G(t) , generates a new one ,G(t+1) . An abstract view of the algorithm is:-

Genetic Algorithms (GA)

- Generate initial population, G(0);
- Evaluate G(0);
- t:=0;
- Repeat
- t:=t+ 1
- Generate G(t) using G(t-1);
- Evaluate G(t);
- Until solution is found

Genetic Operators

- The process of evolving a solution to a problem involves a number of operations that are loosely modeled on their counterparts from genetics .

- Modeled after the processes of biological genetics , pairs of vectors in the population are allowed to “ mate” with a probability that is proportional to their fitness . the mating procedure typically involves one or more genetic operators .

- The most commonly applied genetic operators are :-
- 1-Crossover.
- 2-Mutation.
- 3-Reproduction.

1- Crossover

- Is the process where information from two parents is combined to form children. It takes two chromosomes and swaps all genes residing after a randomly selected crossover point to produce new chromosomes.
- This operator does not add new genetic information to the population chromosomes but manipulates the genetic information already present in the mating pool (MP).
-

- The hope is to obtain new more fit children It works as follows :-
- 1- Select two parents from the MP (The best two chromosomes) .
- 2- Find a position K between two genes randomly in the range (1, M-1) M = length of chromosome
- 3- Swap the genes after K between the two parents.
- The output will be the both children or the more fit one.

2- Mutation

- The basic idea of it is to add new genetic information to chromosomes. It is important when the chromosomes are similar and the GA may be getting stuck in Local maxima. A way to introduce new information is by changing the allele of some genes. Mutation can be applied to :-
- Chromosomes selected from the MP.
- Chromosomes that have already subject to crossover.

3- Reproduction

- After manipulating the genetic information already present in the MP by fitness function the reproduction operator add new genetic information to the population of the chromosomes by combining strong parents with strong children , the hope is to obtain new more fit children . Reproduction imitate to the natural selection.

Genetic Programming (GP)

- Genetic programming (GP) is a domain – independent problem – solving approach in which computer programs are evolved to solve, or approximately solve problems. Thus, it addresses one of the central goals of computer science namely automatic programming. The goal of automatic programming is to create, in an automated way, a computer program that enables a computer to solve a problem.
- GP is based on reproduction and survival of the fittest genetic operations such as crossover and mutation. Genetic operation are used to create new offspring population of individual computer programs from the current population of programs .

- GP has several properties that make it more suitable than other paradigms (e.g. . best – first search , heuristic search , hill climbing etc .) , these properties are :-
- 1- GP produces a solution to a problem as a computer program. Thus GP is automatic programming.
- 2- Adaptation in GP is general hierarchical computer programs of dynamically varying size & shape.
- 3- It is probabilistic algorithm.
- 4- Another important feature of GP is role of pre processing of inputs and post processing of outputs .

To summarize, genetic programming includes six components, many very similar to the requirements fo GAs:

- 1- A set of structures that undergo transformation by genetic operators.
- 2- A set of initial structures suited to a problem domain.
- 3- A fitness measure, again domain dependent, to evaluate structures.
- 4- A set of genetic operators to transform structures.
- 5-Parameters and state descriptions that describe members of each generation.
- 6- A set of termination conditions.

EX:-

By using GA step by step, find the maximum number in 0 to 31.let k=3 and population size=4 ,and the initial population is:-

14 ⟶ 01110
3 ⟶ 00011
25 ⟶ 11001
21 ⟶ 10101

} population

Fitness function will be:-

25&21

3&14

25&21

1 1 | 0 0 1 ⟶ 25
1 0 | 1 0 1 ⟶ 21

1 1 1 0 1 ⟶ 29
1 0 0 0 1 ⟶ 17

14&3

0 1 | 1 1 0 ⟶ 14
0 0 | 0 1 1 ⟶ 3

0 1 0 1 1 ⟶ 11
0 0 1 1 0 ⟶ 6

the new population will be an array and we choose position [16] randomly to do mutation on it:-

1	1	1	0	1
1	0	0	0	1
0	1	0	1	1
0	0	1	1	0

Mutation ⟶

1	1	1	0	1
1	0	0	0	1
0	1	0	1	1
1	0	1	1	0

Mutation 0 ⟶ 1

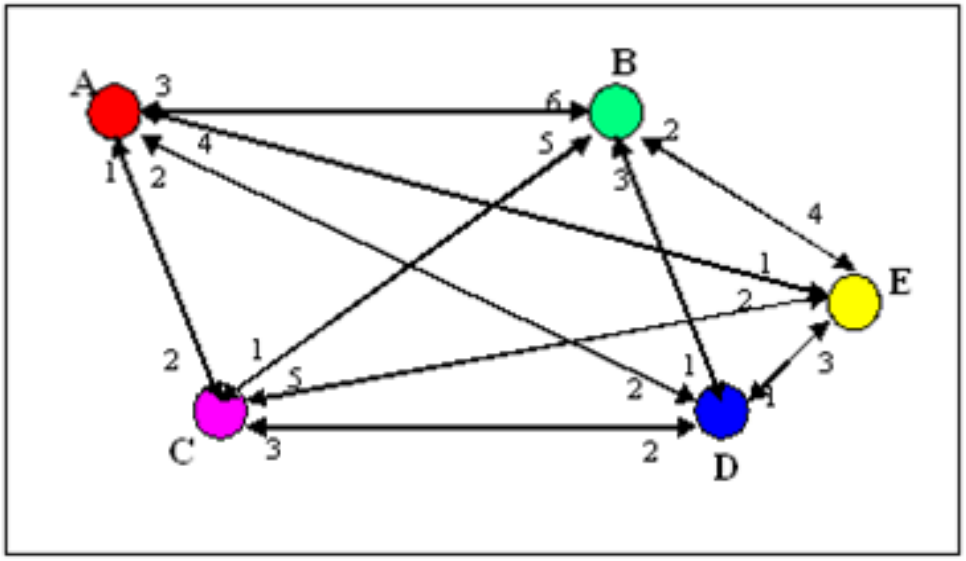
After Mutation the new population will be:

1 1 1 0 1 = 29	reproduction
1 1 0 0 1 = 25	
1 0 1 0 1 = 21	
1 0 1 1 0 = 22	

Because the mutation we replace 17 with 22 in the new population.

EX :-

Apply GA in travelling salesman to find the shortest path | let k=2 and the initial population is:-



A B C D E =12
B C D E A =10
A C D B E =11
E C A D B =11
B A D C E =10
D E B A C =10

} initial population

B C D | E A =10
B A D | C E =10

B C D A E = 6
B A D E C =13

D E B | A C =10
A C D | B E =11

D E B A C = 10
A C D E B =9

E C A | D B =11
A B C | D E =12

E C A D B = 11
A B C D E =12

THE NEW POPULATION IS:-

B C D A E = 6
B A D E C =13
D E B A C = 10
A C D E B = 9
E C A D B =11
A B C D E =12

H.W

Q1: Can the bit string 0 1 0 1 0 1 0 1 be the result of crossing over the following pairs of parents?:-

a- 11111111 and 00000000

b-01010101 and 11111111

c-10100101 and 01011010

Q2: What is genetic algorithm (GA). Explain its algorithm.

Q3: What are the most commonly operators used in GA, list it only, then draw the figure which illustrates schematically the GA approach.

Q4: Adaptive algorithm includes GA and GP in one port of it. Illustrates schematically the main structure of adaptive algorithm.