2/1/18

Artificial intelligence (AI)

"There can't be any definition for intelligence nor any parameter but still we can categorize one as more intelligent than other and this fact is intelligence."

AI is the study of how to make computers do things, thing at the moment people do better.

Domains of AI

① Mundane Tasks : day-to-day work
perception - vision, speech
Natural language - understanding, translation
common sense reasoning.

② formal Tasks : game playing [chess]
mathematics - geometry, logic, integral calc
proofs & programs

③ Expert Tasks : Engineering - design, fault detection, planning
Scientific Analysis
medical diagnosis
financial analysis

AI Experiments

1> Alan Turing - The Imitation Game (circa 1950)
2> MIT Experiment - The psychotherapist chat bot
3> John Searl - Chinese Room Experiment

Scanned with CamScanner

Richt Knight* → Not all topics only algos but not complete.

"Well, I'am studying engg. That ought to give you an idea about what am but soon ill be working @ cabbing factor" → DomuchNore 2009

Topics in AI

→ Perception / Sensing → Computing/Realising → Action

- Speech Recog
- Image Processing
  - CV
  - NNs
  - Knowledge discovery
  - Pattern Recognition
- Natural Language Generation & understanding
- qualitative reasoning
- fuzzy Sets & Logic
- Knowledge Representation
- ontology
- ML
- D Mining
- Problem Solving
- Planning
- Robotics

Scanned with CamScanner

3/1/17

Artificial Intelligence

Complexities

→ Voluminous.
→ Hard to catigorize accurately.
→ Constantly changing.
→ differs from the data being organized in a way that corresponds to the way it'll be used.

"What is AI technique"

We are forced to conclude that an AI technique is a method that exploits knowledge that should be represented in such a way that

- The knowledge should capture generalization i.e if situation share imp properties should be grouped together.
- It can be understood by the people who provide it.
- It can easily be modified to correct errors + reflect changes in the world.
- It can be used in great many situations, even if it is not totally accurate or complete.
- It can be used to help overcome its own sheer bulk by helping to narrow the range of possibilities that must usually be considered.

The Tic-Tac-Toe Game

→ Algorithm / Program 1

This contains two data structures

Scanned with CamScanner

1) **Board**

1	2	3
4	5	6
7	8	9

0 → Black

1 → X

2 → O

2) **Movetable**

→ contain 3^9 moves i.e 19683 move

→ each move is a vector of 9 elements.

Algorithm

1. Views the vector board as base 3 No, convert to a decimal No.
2. Use the No computed in 1. as an index into Move table and access the vector stored there

0	0	1
0	0	2
1	0	0

3. The vect from 2. represents the way the board will look after the move is made, So set board = that vector

= 001002100 $[Y]_3$

≈ ..... $0 \times 3 = X_{10}$

49 || ——— vector

⇒ New board layout

1. $[Y]_3 \rightarrow [X]_{10} \rightarrow$ Move$[X_{10}] \rightarrow$ Newtable

drawbacks

→ Size, predicting all 3^9 moves, Not generalized.

Scanned with CamScanner

→ The approach lacks extisibility, eg in 3D it'll be 3rd moves.

Algorithm 2/ Program 2

2 Data structures

Board

1	2	3
4	5	6
7	8	9

2 → blank
3 → X
5 → O

Turn

An int indicating which move of the game is about to be played, 1 is the 1st move, 9 the last.

This used 3 sub procedure

1 → Make 2

- It returns 5 if the center square is blank other wise return any blank, non corner square.

2 → PossWin(P)

- if P can win then returns yes & the move to be made otherwise returns zero. (NO)

For O → Scan each row/col/diag & if the product of the scan is 50 i.e 2x5x5 then yes O can wins & the 2 will be set to 5 as the step. But if P is losing then find the winning slot of X & set it to 3.

Scanned with CamScanner

3 → Go(n)

- Make a move in square n, this sub Board(n) to 3 if turn is odd & 5 if turn is even.

4/1/18

Game Play Example

(3) X	(5) O	(9) X
	(5) (O)	
(5) O	3 (X)	(8) X

Go(1)

4) if Pauseswin(x) ≠ 0 :
    Go(Pauseswin(x))
else go(make2)

5) if Poswin (x) ≠ 0 :
    Go(Pos win(x))
else if poswin(o) ≠ 0 :
    Go(PosWin(o))
else : if [board7] == blank : Go(7)
    else go(3)

6) if PosWin (O) ≠ 0 :
    go(Pos win (o))
elseif Poswin(x) ≠ 0
    Go (PosWin (X))
else make 2

7) if Pos win (x) ≠ 0 :
    Go Poswin (x)
else : GoPoswin (o)
else go(blank)

8) check o
    → o
check x
    → block X

9) (7)

Scanned with CamScanner

(8, 3, 9, 6)

8, 3, 9, 6

Optimized Algo 2

8	3	4
1	5	9
6	7	2

optimized the PossWin method

[magic square]

winning = 15 Placed : if <0 4 6> = 1 → X

Algorithm 3 / Program 3

→ To decide on the next move look ahead at the board position that results from each possible move.

→ decide which posi is best, and make the move that leads to that position and assign the rating of that best move to the current pos.

1) See if it is a win, call it the best by giving it the highest possible rating

2) otherwise, consider all moves the opponent can make next & see which of them is worst for us, & assume opponent will make that move. whatever rating that move has, assign it to the node we are considering

3) The best node is then the one with highest rating

Using Algo 1, 2, 3 we can define the AI technique

→ Search

- Provides a way of solving problem for which no more direct approach is available, as well as, a frame work into which direct technique that are available can be embedded

Scanned with CamScanner

→ **Use of knowledge**

- It provides a way of solving complex probs by exploiting the structs of objects that are involved.

→ **Abstraction**

- It provides a way of separating imp features & variation ~~and~~ from many un imp ones that would otherwise overwhelm any process.

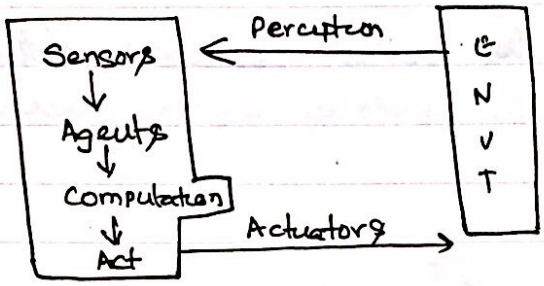
- The AI Technique
  Search to see the future
  Use knowledge to solve the problem
  Abstract the features.

Scanned with CamScanner

8/1/18

Agents

An agent is anything that can be viewed as preciving its environment through sensors & acting upon the ~~envi~~ environment through actuators



Percept Sequence

The complete history of everything the agent has ever perciered. An agent's choice of action @ any given instant can depend on the entire perciered sequence observed till date but not on any thing it has not perciered.

Rational Agent

- Rationality Right thing at the at right time

An agent does the right thing.

4 "Pillars" of Rationality [factors defining Rationality].

1> The performance measure that defines the criteria of success

2> The agent's prior knowledge of envt

3> The actions the agent can perform

4> Agents percept seq to the data

Scanned with CamScanner

For each possible percept sequence a rational agent should select an action that is expected to maximize the performance given the evidence provided by the percept sequence & whatever built in knowledge the agent has

Omnicient Agent

Agent that knows the actual outcome of the action & act according to it.

Types of Environment

1) Fully observable vs Partially observable.
   - → agent can percieve everything about the knowledge in envrt
   - → agent can only percieve some part of the envt

2) Single vs Multiagent
   crossword vs chess

3) deterministic vs stochastic.
   - ↳ current + action gives the prediction of next

4) Episodic vs. Sequential
   - ↳ agent acts only for a specific time i.e the episode. all the info is gained in that episode & no two episodes share the data
   - ↳ Continious absorbtion and action

5) Static vs Dynamic.

Scanned with CamScanner

6) discrete vs Continious
step/ turn based vs continiously acting

Types of Agent

1) Simple Reflex Agent :- These select actions on the basis of the current percepts ignoring the rest of the percept history.

2) Model Based Reflex Agent: The most efficient way to handle partial observability is for the agent to keep track of the part of the world it can't see now i.e. the agent should maintain some sort of internal state that depends on the percept history & therby reflects at least some of the unobserved aspects of the current state.

3) Goal based Agent: As a current state desept the agents need some sort of goal information that describes situations that are desirable.

4) Utility based Agent: Goals alone are not enough to generate high quality behavior in most envts. Goals just provide a crude binary distinction b/w happy & unhappy states. A more general performance measure allows comparison of different world states according to exactly how happy they would make the agent.

5) Learning Agent: Every agent should learn to improve its performance

Scanned with CamScanner

The 4 essentials to build a system to solve a problem.

1 → Define A problem : The problem definition should be precise.
- → The init specs
- → The final state
- → what constitutes the acceptable soln

2 → Analyze the problem : Few very imp features can have an enormous impact on appropriateness of various possible techniques

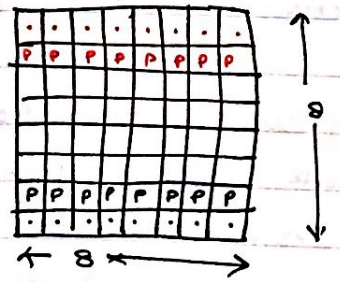
3 → Isolate & Represent the task knowledge necessary to solve the problem :

4 → Choose the best : Choose the best technique & solve it.

"To Define the problem as a statespace Search."

eg : Chess → start, final and legal moves.

start : 8×8 grid



win : king is under attack & no legal moves left to save it.
legal moves : ($\sim 10^{120}$) moves in brute force.

Scanned with CamScanner

" Soft Problems → hard computing "

" Hard Problems → soft computing "
↳ NN, FL, Evolutionary computing.

better way to write the moves.

white pawn at
square (file e, rank 2)
AND
square (file e, rank 3)
is Empty
AND
square (file e, rank 4)
is empty

→ move pawn
from square (file e, rank 2)
to
square (file e, rank 4)

This is better & more generic way to express the rules.

We also need a production system

Advantage of using State Space representation

State space representation method forms the basis of most of AI methods, It's structure correspond to the structure of Problem solving in two way

1) it allows for a formal definition of a problem as a need to convert some given situation into some desired situation using a set of permisible operation

2) it permits us to define the process of solving a particular prob as a combination of know techniques & search, the general technique of exploring the space to try to find some path from the current state to the goal state. Search is very important in the sol^n of hard probs for which no known direct techniques are available

Scanned with CamScanner

11/1/18

Water Jug Problem

Given 2 jugs 4 & 3G, neither has any measuring markers on it. There is a pump that can be used to fill the jugs. How can you get exactly 2G water into 4 G Jug.

→ (x, y) $x \rightarrow$ 4G Jug
$y \rightarrow$ 3G Jug

Start = $(0,0)$ | Goal = $(2, n)$ $0 \leq n \leq 3$

Rule 1. $(x, y) \longrightarrow (4, y)$
if $x < 4$

Rule 2. $(x, y) \longrightarrow (x, 3)$
if $y < 3$

Rule 3. $(x, y) \longrightarrow (x-d, y)$
if $x > 0$

Rule 4. $(x, y) \longrightarrow (x, y-d)$
if $y > 0$

Rule 5 $(x, y) \longrightarrow (0, y)$
if $x > 0$

Rule 6 $(x, y) \longrightarrow (x, 0)$
if $y > 0$

Scanned with CamScanner

Rule 7 $(x, y) \longrightarrow (4, y-(4-x))$

if $x+y \geq 4$ AND $y>0$

Rule 8 $(x, y) \longrightarrow (x-(3-y), 3)$

if $(x+y) \geq 3$

$x>0$

Rule 9 $(x, y) \longrightarrow (x+y, 0)$

$x+y \leq 4;\ y>0$

Rule 10 $(x, y) \longrightarrow (0, x+y)$

$x+y \leq 3;\ x>0$

Rule 11 $(0, 2) \longrightarrow (2, 0)$

Rule 12 $(2, y) \longrightarrow (0, y)$

Solution 1.

X	Y	R
0	0	2
0	3	9
3	0	2
3	3	7
4	2	5
0	2	11
2	0	X

Scanned with CamScanner

Production Systems

- Since Search forms the core of many intelligent processes it is useful to structure AI program in a way ~~that~~ facilitates describing and performing the process. Production Systems provide some structure.

1) A set of rules, each consisting of a left side that determines the applicability of the rule. & the right side that describes the operation if the rule is applied.
2) One or more knowledge / database that contains information is appropriate for particular task. Some part of the db may be permanent while other parts of it pertain only to the current problem.
3) A control Statergy that specifies the order in which the rules are compared to the db and a way of resolving the conflicts that arise when several rules match at once.
   - → The first req of a good CS is it should cause motion
   - → The second req of a good CS is it should be systematic.
4) Rule applies.

Characteristic of Problems [Problem Chars]

1) Is the problem decomposable?

   eg> Integration

   1) $f(x) = x^2 + 3x + \sin^2(x)\cos(x)$

   find $\int f(x)$

2) Can the steps be ignored or undone?

Scanned with CamScanner

eg> Theorem Problems/ Proof

→ some intermediate steps can be ignored.

eg> Chess

→ intermediate step cannot be ignored.

eg> 8 puzzle problem

2	8	3
1	6	4
7		5

→

1	2	3
8		4
7	6	5

Start END.

→ the solution steps are recoverable

3> Is the universe predictable?

eg> Chess

→ The opponent is Not predictable.

eg> 8 puzzle

→ The universe is predictable.

eg> Poker

→ Not all info is there to make a prediction.

4> Is the Good solution absolute or Relative?

- Markus was man
- Markus was Pompeinian
- Markus was born in 40 AD
- All men are mortal
- All pompeinians died in 79AD eruption
- No Mortal lives longer than 150 year
- It is NO 1991 AD.

Ques: is Markus alive?

→ Absolute solution

more than one path to get to solution

⇒ Predicate logic

Scanned with CamScanner

eg > TSP → Relative

5> Is the solution straight or a Path?

straigh → predicat

Path → TSP

6> What is the Role of Knowledge

eg> Chess game versus.

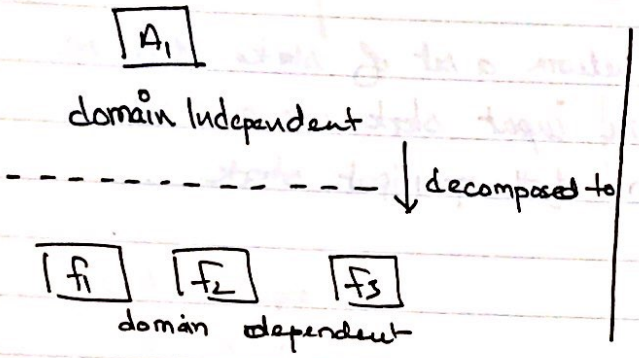
Scanning the newspaper to check ~~all~~ which party is going to win the coming polls

Chess	Polls
→ Rules	→ make new DB
→ Insight into the opponents mind	→ update it regularly
	→ many more variable

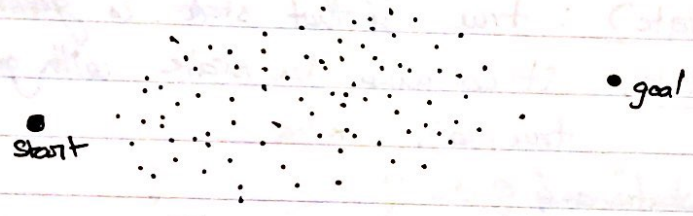
Scanned with CamScanner

AI Algorithms

15/1/18



Making AI Algo.



The state space.

Explicit: Ones that are generated as we move from start to end

Implicit: The ones that are not generated by can be.

Algorithm 1: Generate & Test

```
while [more children exist] do
        do Generate a candidate          // how to generate children
            Test whether it is a soln    // to test if it is soln
return Failure
```

Scanned with CamScanner

The subprocedures of GenerateAndTest.

1. MoveGen(State) :

- Take a state as input and return a set of states that are reachable in one step from the input state. we call set of states as successor/ children of the input state.
- Domain dependant

2. GoalTest

1> SimpleGoalTest (state) : true if that state is goal or false

2> GoalTest (Goal, state): it compares the state with goal & returns true / false

1→ assumes goal is stationary

2→ the goal can change over time

- Domain dependant.

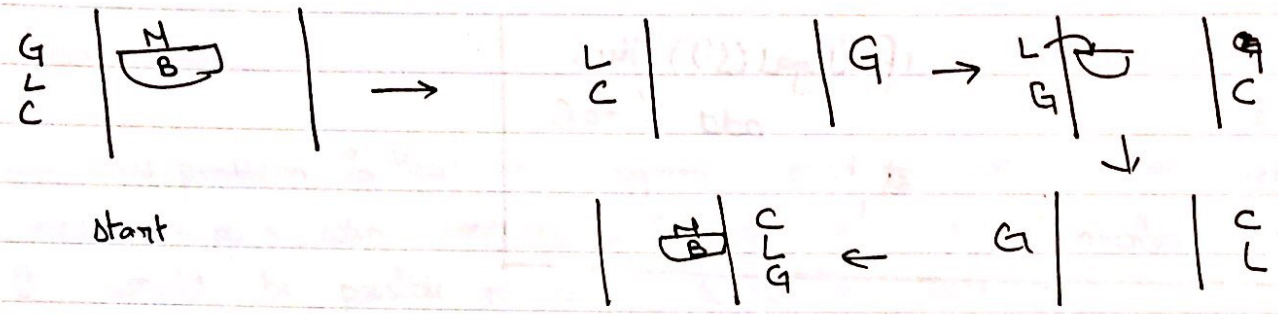
Example

The RiverCrossing Problem

→ Man, boat, goat, lion, cabbage. The 5 entities

A man needs to transport a lion L, a goat G & cabbage C accross a river. He has a boat B, in which he can take only one of them at a time it is only his presence that prevents L from eating G & G from eating the eating the cabbage. He can neither leave L with G nor C with G. How can He take them all accross the river.

Scanned with CamScanner



Visual Example

Left bank

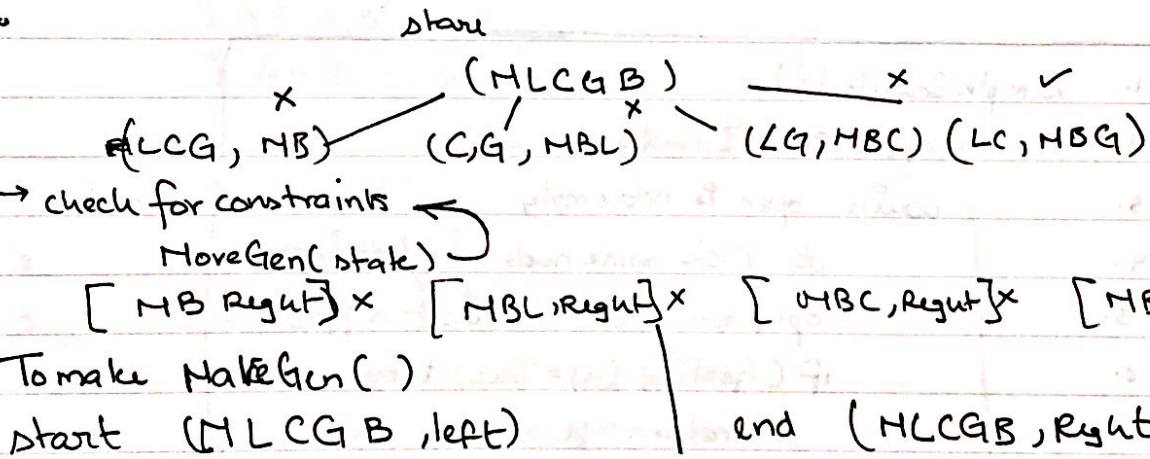
L_1 (M, L, C, G, B)

start

Right bank

L_2 (M, C, C, G, B)

end / Goal.




```
Move Gen(n)
    initialize set of successors "C" to empty set.
    Add m to the compliment of given state "n" to get new state "S"
    if ( given state has left) then
            add Right to "S"
    else add Left
    if (legal (S)) then add S to set of succors C
    for each other entity E in N:
            make "S'" of "S",
            add E to S'
```

```
            if (legal(S')) then
                add S' to C
        }
    return C
```

This moveGen is problem specific.

Simple Search1 Algorithm

- An improvement on TIG
- we now have a set datastructure, open as start

```
Simple Search 1()
    open ← {start}
    while open is not empty
        do pick some node n from open
        open ← open \ n   [set diff]
        if (GoalTest (n) = True) then
            return n
        else
            open ← open ∪ movegen(n)
return failure
```

- Adds more context to generate & test
- reduces ambiguity

Shortcoming of SS1

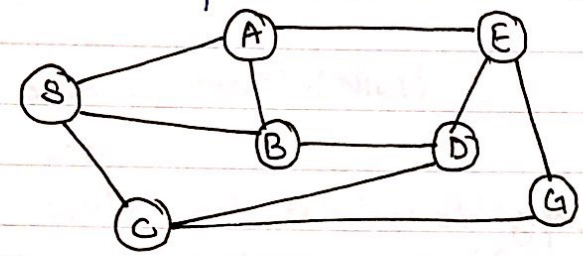
Scanned with CamScanner

(1, 2, 3) (1, 2)

16/1/18

Simple Search 2

The first problem is that the program could go into ∞ loop. This would happen when state space is graph with loops in which it would be possible to come back to the same state



open (S)

{ABC}

{ASDGB}

{AABCDGB}

⋮

```
Simple Search 2 ()
    open ← {start}
    closed ← { }
    while open is not empty do:
        pick a node from open, n
        open ← open \ (n)
        closed ← closed ∪ {n}
        if GoalTest (n) == True
            return (n)
        else
            successor ← MoveGen (n)
            open ← open ∪ successor
            open ← open \ closed
    return failure
```

The second problem is that it returns the goal state when it finds it.

Scanned with CamScanner

our simple approach is to exactly keep track of the complete path to each new state. we can do this by modifying the state representation to store a list of nodes denoting the path

→ Set is now a list of lists

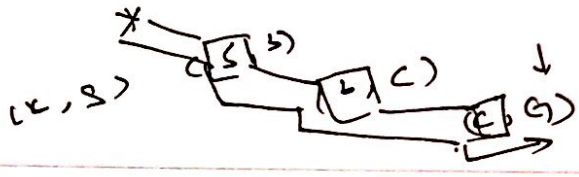
Simple Search3 ().

```
Simple Search3()
    open ← {{start}}
    close ← {}
    while open is not empty do:
        n ← pick a node n from open
        h ← Head(n)
        if GoalTest(h) = True
            return(reverse(n))
        else
            successor ← {MoveGen(h) \ closed}
            successor ← {successor \ map Char(open)}
            open ← open \ n
            for each successor s do
                add cons(s,n) to open
    return failure
```

MapChar (h) : Takes a lists of lists as an argument & returns a list containing the head of the input list.

Cons (s,n) : Appends the successors to the head of the node list n,

Scanned with CamScanner



Depth First Search

in the sublist, only parent and child is maintained
This is a blind search algorithm.

```
1. DFS()
2.     open [[start], [NIL]]
3.     close []
4.     while not null(open) do:
5.         nodepair ← Head(open)
6.         node ← Head(nodepair)
7.         If goalTest(node) = True
8.             return (reconstruct_path(nodepair, closed))
9.         else
10.            close ← Cons(nodepair, closed)
11.            children ← MoveGen(node)
12.            noloops ← RemoveSeen(children, open, closed)
13.            new ← MakePair(noloops, node)
14.            open ← Append(new, Tail(open))
15. return failure
```

To change this to BFS() in line 14. swap (new, tail(open))

Scanned with CamScanner

17/1/18

Evaluation of Search Methods

1> **Completeness** : Does the algo always find a solution when it exists.

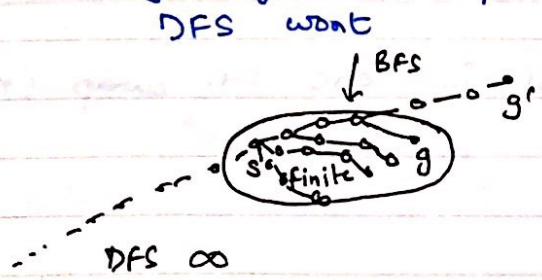
2> **Time Complexity** : In our case we will get a measure of time complexity by the no. of nodes algo picks up before it finds the solution. [No of nodes in closed]

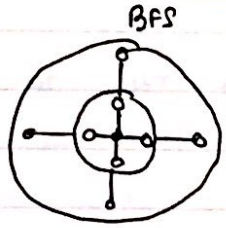
3> **Space Complexity** : How much space does the algo need measured by no. of nodes in open

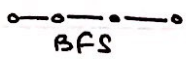
4> **Quality of Solution** : The length of path from start to goal.

BFS vs DFS

1> **Completeness** : for finite search space both BFS & DFS work be complete.
for infinite search space BFS may work but DFS wont







2> **Time Complexity** : i> Goal Close to start BFS
ii> Goal far from S and on left DFS

Scanned with CamScanner

3) Analysis Needs

DFS : $d+1$ (if we're @ d & goal @ d+1 & BFS constant)

Then we have

$$\left[(d+1) + \left(\frac{b^{d+1}-1}{b-1}\right)\right]/2$$

$$= \frac{b^{d+1} + b + db - d - 2}{2(b-1)}$$ [at high b, d $b + db - d - 2 \ll b^{d+1}$]

$$\Rightarrow \frac{b^{d+1}}{2(b)} \approx \boxed{\frac{b^d}{2}}$$

$$\Rightarrow \boxed{O\left[\frac{b^d}{2}\right]}$$

BFS : $\left[\frac{b^d-1}{b-1}\right]$ at current level

$\left[\frac{b^{d+1}-1}{b-1}\right]$ at the next level

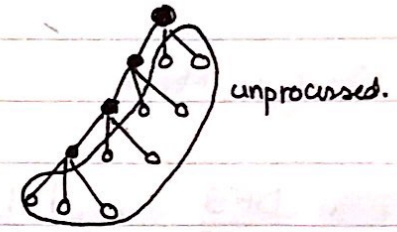
$$\left[\left(\frac{b^d-1}{b-1}\right) + \left(\frac{b^{d+1}-1}{b-1}\right)\right]/2$$

$$= \boxed{\frac{b^d(b+1)}{2(b)}}$$ $(b \gg\gg 1)$

$$\boxed{\frac{BFS}{DFS} = \frac{b+1}{b} \Rightarrow BFS \approx DFS}$$

Scanned with CamScanner

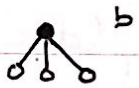
3) Space Complexity



DFS : $\boxed{(b-1)\,d+b}$

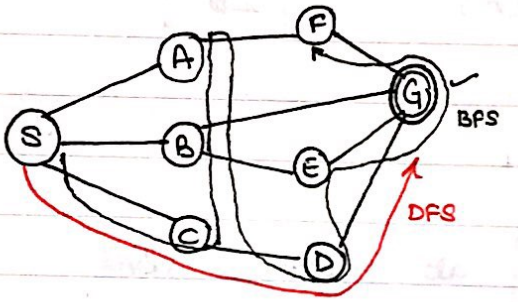
Size linearly grows with depth.

BFS : $\boxed{b^d}$



Size exponentially grows with depth

4) Quality of Solution



In some cases BFS tends to find the shortest path.

Scanned with CamScanner

18/1/18

Heuristic Search

- The idea of a heuristic function is to guide the search so that it has a tendency to explore the region leading to the goal.
- A Heuristic function is designed to help the algo to pick the candidate from the open list that is more likely to be on the path to the goal.
- A Heuristic value is computed for each of the candidate. The H.V could be an estimate of the dist to the goal from ~~that~~ current node.
- The heuristic function must not be computationally heavy, In order to ~~cor~~ cut down the computational cost of the algo.

Example: Shortest Path

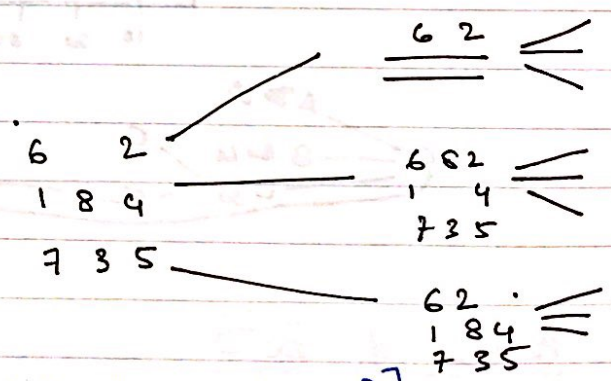
we have two heuristic functions : L_2 distance , L_1 distance.

$h(n)$ ← The heuristic function.

Example: 8 puzzle

1	2	3
8		4
7	6	5

Goal



1) distance function : $\Sigma(L_{1ij})$ [Σ (manhattan of each element)]
2) no. of tiles displaced from actual goal state.

Scanned with CamScanner

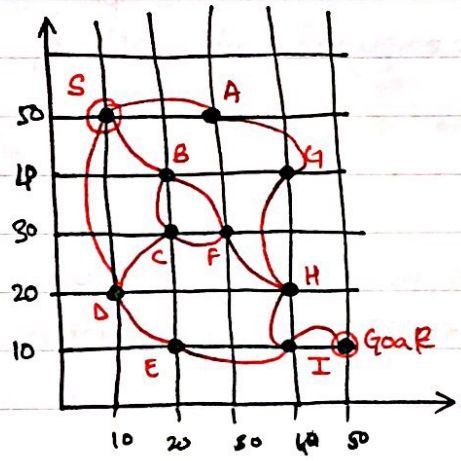
Best First Search

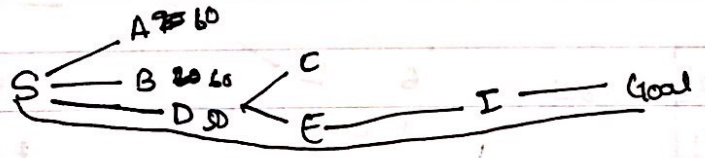
- Same as DFS but in line 14. we make a change to

14. open ← $Sort_h$ (append (New, Tail (open)))

- The data structure of sublist also is changed to:

(parent, child, h(value))





Analysis of BFS

Completeness: in a finite statespace : finds the solution
in an infinite state space : may not find sol^n

Scanned with CamScanner

Quality of Solution : depends on the heuristic function chosen.

Space Complexity : depends on the heuristic function

Time Complexity : depends on the heuristic function

1) $$\boxed{\text{Effective Branching factor} = \frac{\text{total No of Nodes Expanded}}{\text{No. of Nodes in Solution}}}$$

- if EBF ≈ 1 [EBF $\rightarrow 1$] then it is a good $h(x)$

2) $$\boxed{\text{Penetrance} = \frac{\text{no. of nodes in Sol}^n}{\text{total no. of nodes expanded}}}$$

if $P \rightarrow 0$ then this is a bad $h(x)$ P should tend to 1

Scanned with CamScanner

23/1/18

Hill Climbing (steepest gradient ascent)

→ There is no need for goal test, this automatically takes the shortest path as it'll avoid all paths where value of h increases

→ This is repeated till the value of h becomes zero

open ← $Sort_h$ (New)

↓

FirstNode

↳ Non path nodes are not stored.

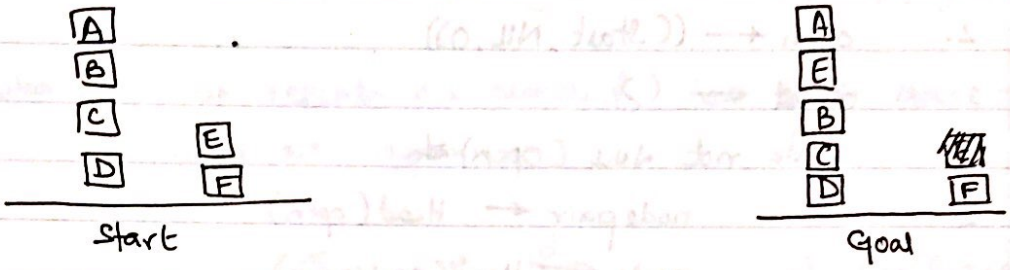
```
Hill Climbing (start):
node ← start
newnode ← Head (Sort_h (MoveGen(node)))
while h(newnode) < h(node) do:
        node ← newnode
        newnode ← Head(Sort_h (MoveGen (node)))
return newnode.
```

- This uses steepest gradient ascent.
- This is prone to getting stuck in local maxima.

Scanned with CamScanner

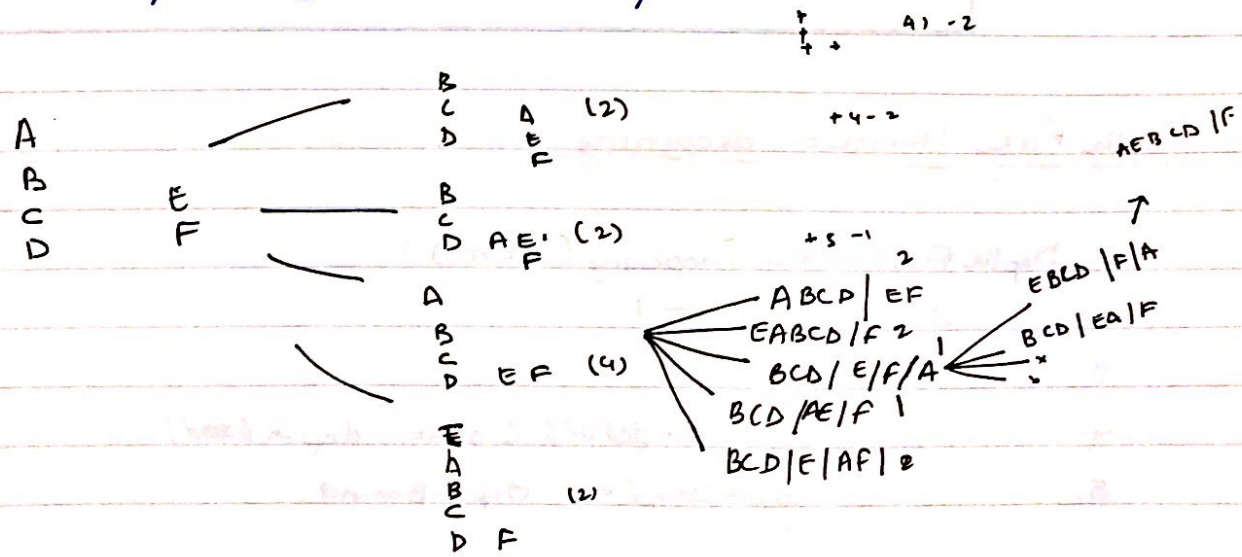
Blocks Word Problem

Given some blocks, rearrange to goal state but only one at a time from the table from the top of a pile.



$h_1(n)$: Checks whether each block is on the correct block wrt final configuration, we add 1 for every block that is on a block it is suppose to be on 4 − 1 for each in the wrong place.

$h_2(n)$: First it looks at the entire pile that the block is on, if the config of the pile is correct wrt goal it adds 1 for every block in the pile or else −1 for every block in that pile.



Scanned with CamScanner

24/1/18

Algorithm : Depth Bound DFS.

Adds a max depth level of search space.

```
DepthBoundDFS (start, depthBound):
    open ← ((Start, NIL, 0))
    closed ← ()
    while not Null (open) do:
        nodepair ← Head (open)
        node ← Head(nodepair)
        if goalTest(node) = TREE
            return reconstructPath( nodepair, closed)
        else
            closed ← Cons ( nodepair, closed)
            if rest (Head (nodepair))) < depthbound
                children ← moregen(node)
                noloops ← Remove seen (children, open, closed)
                new ← MakePair (noloops , node, Rest (CH(np)))
                open ← Append (new, Tail(open))
    return "No Path"
```

Depth First Iterative deepening.

```
DepthFirstIterativeDeepening (start):
    depthBound ← 1
    while True :
        DepthBoundedDFS ( start , depthBound)
        depthBound ← depthBound.
```

Scanned with CamScanner

- This iteratively applies DFS
- This is simulating BFS and in some cases the quality of solution of ~~the~~ it is better. [BFS]
- This also has linear space complexity like BFS ~~than~~. [DFS]

Total improvement : space complexity is reduced & quality of solution improved.

Drawback : This repeats the search of low depth state space over and over again.

$$\frac{L+I}{L} = 1 + \frac{b^d - 1}{b-1} \Big/ b^d \quad \Big| \text{ at very large values of } d.$$

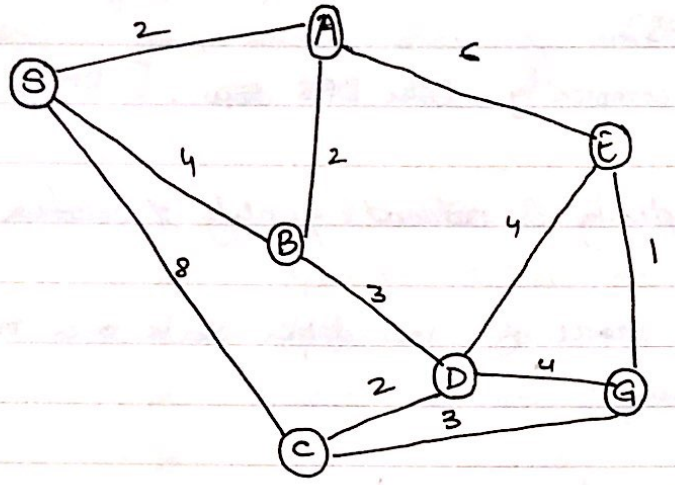
$$= 1 + \frac{1}{b-1} = \boxed{\frac{b}{b-1}}$$

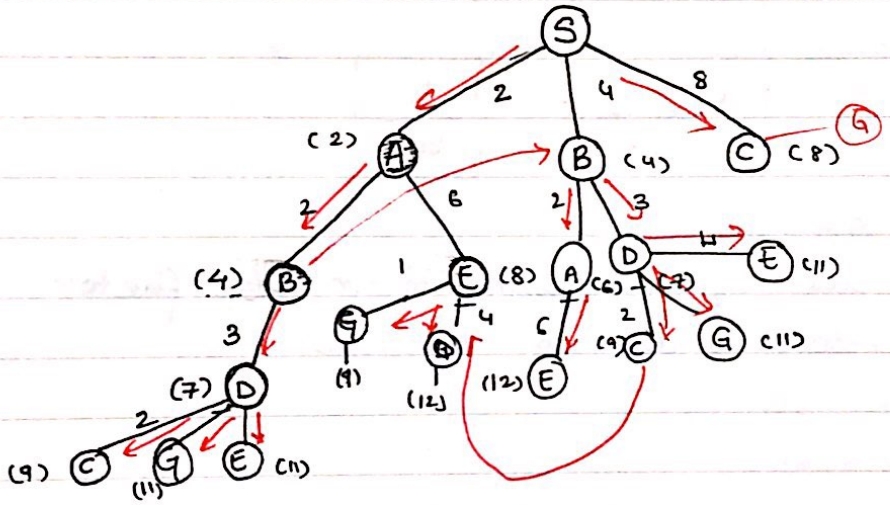
∴ @ high depth and high Branching factor the factor by which additional work is done is negligible

Scanned with CamScanner

25/1/18

Branch & Bound





Example: The TSP.

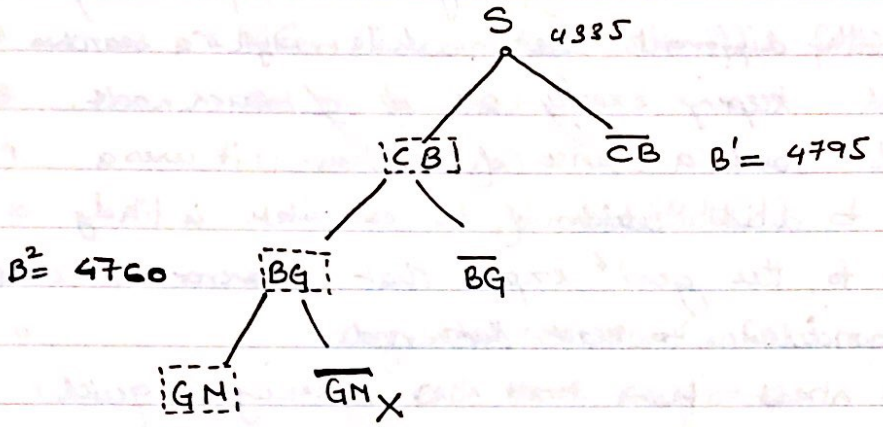
	Chennai	Goa	Mumbai	Delhi	Bangalore
Chennai	0	800	1280	2190	360
Goa	800	0	590	2080	570
Mumbai	1280	590	0	1540	120
Delhi	2190	2080	1540	0	2434
Bangalore	360	570	1210	2434	0

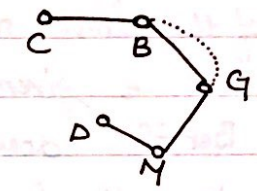
Scanned with CamScanner

$$\frac{(360+800)+(570+590)+(1590+1240)+(1540+2080)+(360+570)}{(2)}$$

$= 4335$

i $\quad S = NiL$ (No edge) $\quad$ Bound $= 4335.$





1) The path refines on each step
2) The function is refined fwd.

Scanned with CamScanner

30/1/18

A* Algorithm

Both B&B and Dijkstra's algo extends the least cost partial solution. while the latter is designed to solve a general problem. the former uses a similar blind approach even though it has a specific goal to achieve.

- B&B generates a search tree that may have duplicate copies of the same node with different cost. while Dijkstra searches over a given graph keeping exactly one copy of each node.
- BestFS does not have a sense of direction. it uses a heuristic function to decide which of the candidate is likely to be the closest to the goal & expand that. However it doesn't keep track of cost incurred to reach that node
- BestFS only looks ahead from node n, seeking a quick path to the goal. while B&B only looks behind. keeping track of the best path found so far. A* does both

$$f^*(x) = g^*(x) + h^*(x)$$

→ The actual cost

↳ The correct prediction

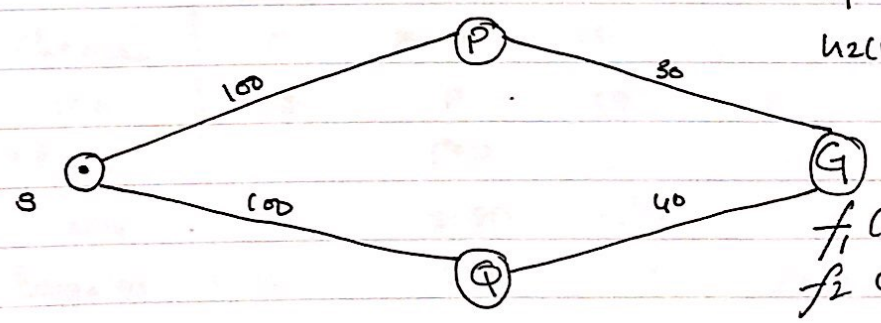
$h(x)$ → just a prediction

$g(x)$ → Best till Now | $g^*(x)$ the Best | From B&B.

- $g^*(x) \leq g(x)$ and $h(x) \leq h^*(x)$
  if this doesn't hold then the solution is incorrect.

$h_1(P) = 50$ $h_1(Q) = 45$

$h_2(P) = 15$ $h_2(Q) = 10$



$f_1(P) = 100+50$

$f_2(P) = 100+45$

Scanned with CamScanner

```
Procedure A*
    open ← List (Start)
    f(start) ← h(start)
    parent(start) ← NIL
    closed ← {}
    while open is not empty do:
        Remove n from open with least f(n)
        Add n to closed
        if goalTest(n) = True
            return [Reconstruct-Path()]
        else
            neighbours ← moveGen(n) ; open ← RemoveSeen(op, cl, nes)
            for each m ∈ neighbours
                case m ∉ open & m ∉ closed
                    g(m) = g(n) + k(n, m)
                    f(m) = g(m) + h(m)
                    parent[m] ← n
                case m ∉ closed [ m ∈ open]
                    if(g(m) > g(n) + k(m,n)) :
                        g(m) = g(n) + k(m,n)
                        f(m) = g(m) + h(m)
                        parent[m] ← n.
                case m ∉ open [ m ∈ closed]
                    if (g(m) > g(n) + k(m,n)]
                        g(m) = g(n) + k(m,n)
                        f(m) + g(m) + h(m)
                        parent[m] ← n
                        propagate(m)
```

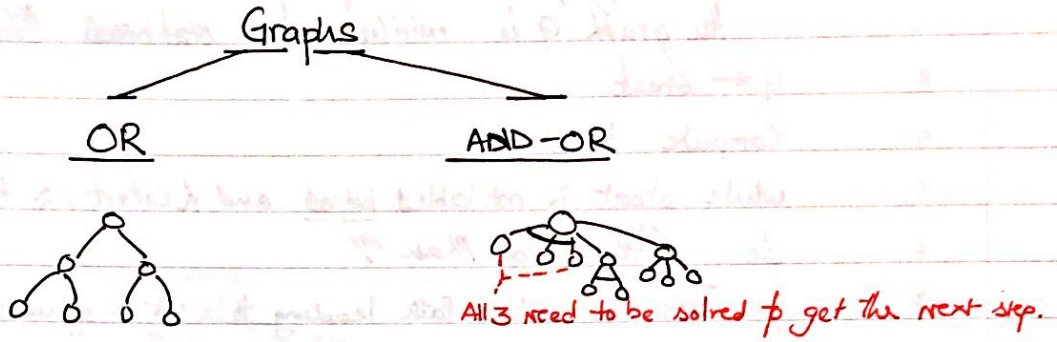
Scanned with CamScanner

```
Propagate(m)
    for each s ∈ moregen(m)
        if (g(s) > g(m) + K(m,s))
            g(s) ← g(m) + K(m,s)
            f(m) ← g(s) + h(s)
            parent(s) ← n
            if s ∈ closed
                propagate(s)
```

Scanned with CamScanner

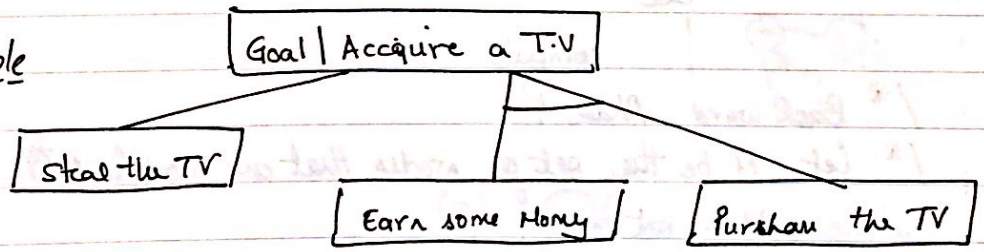
31/1/18

Graphs



AND-OR Graph are useful for representing the sol^n of a problem that can be solved by decomposing them into smaller problems All of Which must be solved, This decomposition/reduction generates ~~and~~ arcs, that we call AND arcs.
One AND arc may point to any no of successor nodes. All of which must be solved in order for the arc to point to a solution.

Example



To solve i.e work on AND-OR graphs we have the AO* algorithm
- This is based on best first search.
- This has two phases, 1) the forward pass 2) the backward pass.

Futility: The. highest possible value.

Scanned with CamScanner

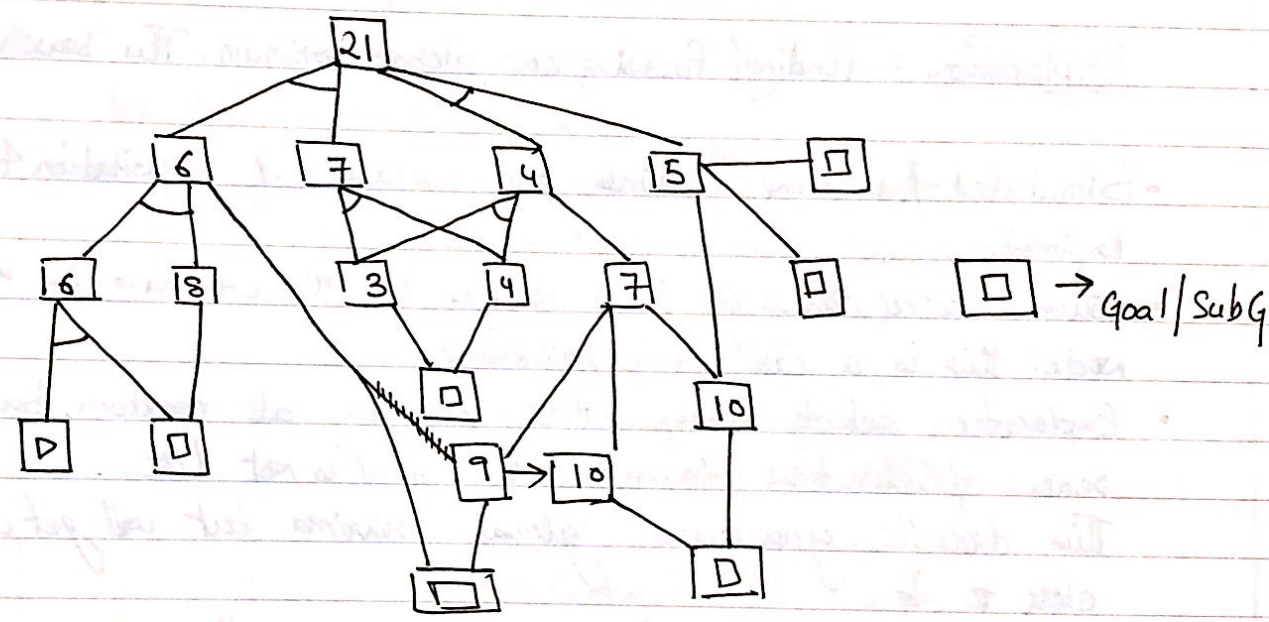
```
AO*( )
    /* uses a graph G instead of open & closed
       the graph G is initilized to the stat node */
    G ← start
    Compute h(start)
    while start is not labled solved and h(start) ≤ futility. d
    do  /*forward Phase */
        Trace the marked Path leading to a set U of unexpanded nodes
        Select n from U
        Children ← successor (n)
        if children is empty
            h(n) ← futility
        else
            for each S ∈ children
                Add S to G
                if S is Primitive
                    h(S) ← 0
                else
                    compute h(S)
        /* Back ward Phase */
        /* Let M be the set of nodes that are modified */
        while M is not empty
            select deepest node d from M & remove from M.
            Compute best cost of d from its children
            Mark best option at as Marked
            if all Nodes connected through Marked are labelled solved
                label d as solved
            if d has changed
                Add all parents of d to M
    if start is marked solved
```

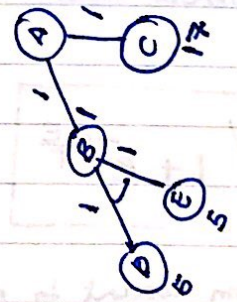
Scanned with CamScanner

30: return marked subgraph starting at start

31. else

32. return failure.





Scanned with CamScanner

1/2/18

Simulated Annealing

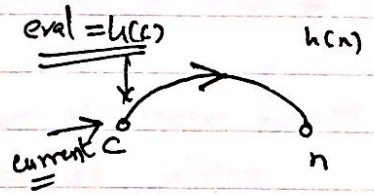
- An extension of hill Climbing, but HC was prone to local maxima. Simulated Annealing tries to overcome local maxima

Exploitation : Refining / using / extracting work.

Exploration : finding / focusing on global optimum. The search is global.

- Simulated Annealing combines the concept of Exploitation + Exploration.
- When we're stuck @ local maxima in HC we have no "Best" node. this is a case of exploitation.
- Exploration selects some other location at random, but the sense of direction towards the goal is not lost.
  This doesn't guarantee global maxima but will get us close to it.
- Annealing is derived from metallurgy, this behave like a minimizational problem

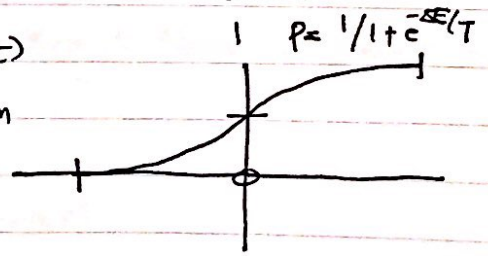
SA = HC + Random Walk



$$\boxed{p = 1 / 1 + e^{-\frac{\Delta E}{T}}}$$

Eval(c) − Eval(n) = ΔE

The P(c from current to next)
T = constant based on Problem
$\Delta E = h(c) - h(n)$



Scanned with CamScanner

T controls ΔE | as $\Delta E \uparrow$ $P \to 1$ | $\Delta E \downarrow$ $P \to 0$

```
Simulate Annealing( .. )
    node ← start
    best_node ← node
    T ← Some large value.
    for time = 1 to No. of Epochs :
        do while (termination criteria) /*[ / M cycles ]*/
            neighbours ← RandomNeigbour( node)
            ΔE ← Eval(neighbour) - Eval (node)
            if Random(0,1) < 1/(1+e^(-ΔE/T)) :
                New_Node ← neighbour
                if Eval ( neighbour) > Eval (best_node)
                    best_node ← neighbour
                T ← Cooling Function (T, time)
    return best_node
```

Scanned with CamScanner

20/2/18

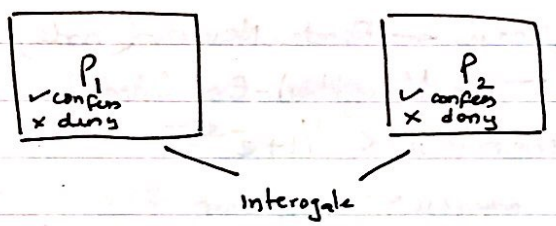
Game Playing

1. Every game has well defined rules ∴ they are easy to evaluate
2. They can be evaluated easily.

Game Theory

Rational choice in a multiagent scenario

Prissoner's Delima



		P_1 C	P_1 D
P_2	C	E, E	B, F
	D	F, B	D, D

~~B = E~~ $B < D < E < F$

Punishment is dependent on choice

CC → E punish

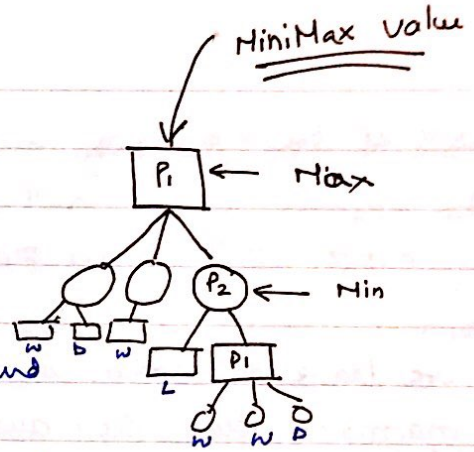
P

P_1 and P_2 can not ~~chao~~ communicate.

We limit ourselves to 2 player board games, with 0 sum ie 1 coins -1 lose (0-0 draw) and the moves are alternating and complete info is visible and the game is deterministic.

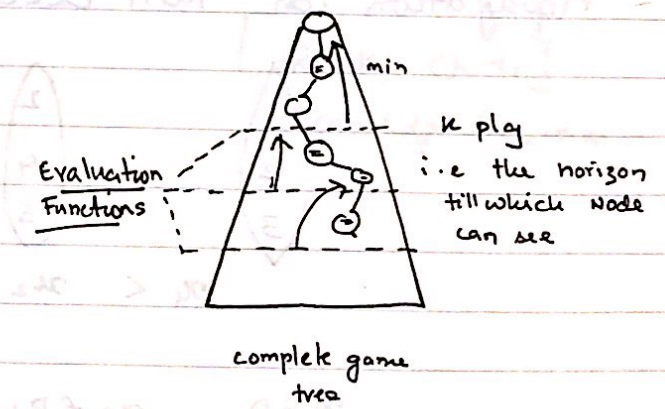
Scanned with CamScanner

Game Tree



- → layered tree
- → Each level has 2 types of nodes
- → The two types are Max and Min and we always start with max.
- → Initially the leaves of the tree are labled, win, lose, draw. WRT to max
- → This is done by propagating value from leaf to root via BackProb
- → The value @ root is the MiniMax value : Most probable outcome when the game is played rationally.
- → min will always try to minimize the value i.e make max lose and max will alway do the opposite.
- → Stratergy is a subtree in which there is one choice for max and all for min.

Do Mini-Max Pseudo Code from the book # 21/2/18



Scanned with CamScanner

22/2/18

Constraint Satisfaction

crypto arithmatic puzzle.

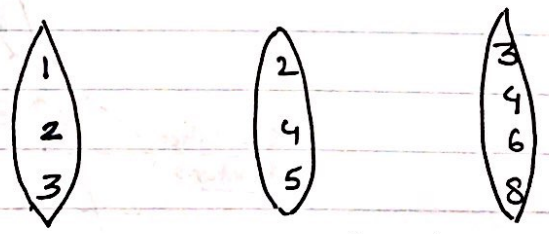
we have to assign a decimal value to each letter in such a manner that the answer to the problem is correct. If some letters occurs more than once, it must be assigned the same number each time. No two letters (different) can have the same digit assigned.

Constraint Propagation Algo:

First the constraints are discovered and propagated as far as possible throughout the system. If there is still no solution then search begins.

A guess about something is made and added as a possible ~~solution~~ new constraint.

- Propagation can then occur with this new constraint & so forth



$x_1 < x_2 < x_3$

$x_1 \in D_1 \quad x_2 \in D_2 \quad x_3 \in D_3$

This is a 4 step algorithm

Scanned with CamScanner

① propagate available constraints. To do this, first set open to the set of objects that must have values assigned to them in a complete sol^n. Then do until an inconsistency is detected or open isn't empty.

a) select object 'ob' from open. strengthen as much as possible the set of constraint that apply to ob.

b) If the set is diff from the set that was previously assigned. the last time ob was examined or if this is the first time ob has been examined then add to open all object sharing a constraint with ob.

c) remove ob from open.

② if union of constraint discovered above defines a sol^n. Then quit and report solution.

③ if union of constraints discovered above defines a contradiction return failure.

④ if ~~neg~~ neither of ②/③ occurs then it is necessary to make a guess at something in order to proceed and then call C.Prop again

$$\begin{array}{r} C_4C_3C_2\,4 \\ \text{SEND} \\ +\ \text{MORE} \\ \hline \text{M O NEY} \end{array}$$

[R&K Ch-4]

Knowledge Rep, issues.

C_4 atmost 1 $\therefore$ $m=1$

C_3 atmost 1 $\therefore$ $S = 8, 9$

since M already $=1$ $O=0$

$S + m + 8_8 > 9$

$N = E + C_2$

$N + R + C_1 = E$

Scanned with CamScanner

3/14/18

Knowledge Representation

- **Predicate Logic**

one particular way of representing facts is the language of logic. logical formalism is appealing because it immediatly suggests a powerful way of deriving new knowledge from old i.e mathematical deduction.

$\rightarrow$	(implication)	
$\neg$	(Not)	
$\vee$	(OR)	connectors.
$\wedge$	(and)	
$\exists$	(there Exists)	quantifiers
$\forall$	(for all)	

This originates from prepositional logic.

eg if it is raining then it is not sunny RAIN $\rightarrow \neg$ SUNNY.

Short comings

Eg 1: Socrates is a man : SOCRATES MAN
Plato is a man : PLATO MAN
→ No common identifier.

Predicate ↘ ↙ Argument.

MAN(SOCRATES)

MAN(PLATO)

Predicate is a part of sentence/clause containing a verb and stating a something about the subject.

Scanned with CamScanner

$\forall x$: man$(x) \longrightarrow$ mortal(x) (all men are mortal)

eg 2

1. Marcus was a man

   man(marcus)

2. Marcus was a Pompeian

   Pompeian(marcus)

3. Ceasor was a ruler

   ruler(ceasor)

4. All Romans were either loyal to Ceaser or hated

   $\forall x$: Roman$(x) \longrightarrow$ loyal(Ceasar)$x \vee$ hated$(x$, ceasar)

5. Everyone is loyal to someone.

   ~~$\forall x) \exists y$~~ $\exists y \; \forall x : \rightarrow$ loyal(x, y)

   $\exists y \; \forall x : \rightarrow$ loyal(x, y)

6. People only tried assassinate rulers they are not loyal to.

   $\forall x : \forall y$: person$(x) \wedge$ ruler$(y) \wedge$ trytoassassinate$(x, y) \rightarrow$
   $\neg$loyal(x, y)

7. Marcus tried to assassinate Ceasar:

   trytoassasinate (markus, Ceasar).

8. All men are people

   $\forall x$: men$(x) \longrightarrow$ person(x).

was marcus loyal to Ceasar?

The logic uses and-OR graphs.

Compatable function & Predicate.

gt(1,0)
gt(2,0)
⋮

tells if 1 are is > than other

Whatever proof procedure we use, when it comes upon one of these predicates, instead of searching for it, explicitly in the DB. or attempting to deduce by further reasoning we can simply invoke a procedure which will specify in addition to our regular rules that will evaluate it and return true/false. This is ~~ten~~ known as computable predicate

1. Marcus was a man
2. Marcus was Pompeian
3. Marcus was born in 40 A.D
4. All men are mortal
5. All Pompeians died when volcano errupted in 79 A.D
6. No Mortal Lives longer than 150 Years.
7. It is Now 1991.
8. Alive means Not dead.
9. if some one dies then he is dead at all Later times.

man (Marcus)

Pompeian (Marcus)

BornON (Marcus, 40AD)

$\forall x$: man(x) $\rightarrow$ mortal (x)

$\forall x$: Pompeian (x) $\rightarrow$ DIED(x, 79) // errupted(volcano, 79) $\wedge \forall x$

$\forall (x)$: ~~AGE~~ AGEGT(150, x) $\rightarrow$ DEAD(x)

NOW() $\Rightarrow$ 1991

alive (x) $\rightarrow$ $\neg$ DEAD(x) $\quad \forall x : \forall t$ alive(x,t) $\rightleftarrows \neg$ Dead (x,t)

$\forall x : \forall t_1 : \forall t_2$: mortal(x) $\wedge$ born $(x, t_1) \wedge gt(t_2 - t_1, 150)$
$\rightarrow$ died (x, t_2)

$\forall x : \forall t_1 : \forall t_2$: died $(x, t_1) \wedge gt(t_2, t_1) \rightarrow$ dead (x, t_2)

Scanned with CamScanner

15/3/17

Predicate logic (Contd...)

Resolution produces proofs by refutation. In other words, to prove a statement, resolution attempts to show that the negation of the statement produces a contradiction with the known statement

Clause form (9 step process to get a clause form / canonical form)

"All Romans who know Marcus either hate Ceasor or think that anyone who hates anyone is crazy."

$\forall x [Roman(x) \wedge knows(Marcus)]$
$\rightarrow [hate(x, Ceaser) \vee \forall y : \exists z [hate(y,z) \rightarrow thinkCrazy(x,y)]]$

1) Eliminate Implications ($\rightarrow$)

1. Using the fact $A \rightarrow B$ means $\neg(A \vee B)$

$\forall x \neg [Roman(x) \wedge Know(Markus)]$
$\vee [hate(x, Ceaser) \vee y : \neg \exists z \; hate(y,z) \vee thinkCray(x,y)]$

2) Reduce scope of $\neg$ to single term ($\neg\exists y \rightarrow \forall y$)

eg $\neg \exists y : p(x) \rightarrow \forall y \neg p(x)$

$\forall x [\neg Roman(x) \vee \neg know(Markus)] \vee [hate(x, Ceaser) \; \forall y : \forall z \neg(hate(y,z)$
$\vee thinkCrazy(x,y)]$

3) Standardize vars so that each quantifier a unique variable

Eg $\forall x P(x) \vee \forall(x) : Q(x)$
becomes $\forall x P(x) \vee \forall(y) Q(y)$

Scanned with CamScanner

4) Move all quantifier to left without changing relative order

$\forall x \forall y \forall z : [\neg Roman(x) \vee \neg Know(Markus)] \vee [hate(x, Ceasar) \vee \neg hate(y, z) \vee think(crazy(x, y))]$

5) Eliminate Existential Quantifier: ($\exists$)

eg $\exists y : P(y)$ is now $P(S_1)$ [S_1 is a function that gives Y]

6) Drop the prefix : i.e all the quantifiers in Left.

$[\neg Roman(x) \vee \neg Knows(Markus)] \vee [hate(x, Ceasar) \vee \neg hate(y, z) \vee think(crazy(x, y))]$

7) Convert the sentence into conjunction (AND) of disjuncts (OR).
apply distributive law

$(a \wedge b) \vee c = (a \vee c) \wedge (b \wedge c)$

8) Create a seperate clause corresponding to each conjunct.

9) Standardize apart the vars in set of clauses generated in (8)

Scanned with CamScanner

20/3/18

Resolution

The resolution procedure is a simple iterative process at each step two clauses called the parent clauses are compared and then resolved yielding a new clause that the has been inferred from them.

cancelled

winter V Summer ¬ Winter V cold

⇒ Summer V cold [this is ~~ex~~ XOR]

winter , ¬ winter → empty clause AKA <u>contradiction</u>

1) Given set of clauses F in clause form.

1 <u>Convert all Prepositions of F to clause form</u>

2 <u>Negate "P" and convert the ¬P to clause form & add to F.</u>

3 <u>Repeate till contradicte or no progress</u>

1. Select two clauses & call them the parent clause
2. Resolve them together : if there are ANY pairs of literals 'l' and '¬l' s.t one of the parent clause has 'l' and other has '¬l' then select one such pair and eliminate 'l' and '¬l' from resolvant
3. if the resolvant is empty then a contradiction is found if NOT, add it to set of clauses avail to the procedure.

Scanned with CamScanner

given

1) P

2) $(P \wedge Q) \rightarrow R$

3) $(S \vee T) \rightarrow Q$

T

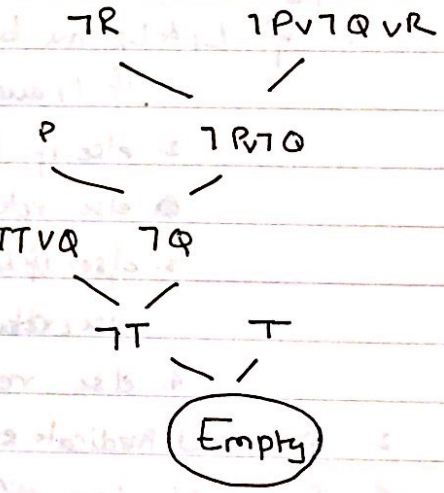
1)

→ in clause

$$\begin{bmatrix} P \\ \neg P \vee \neg Q \vee R \\ \neg S \vee Q \\ \neg T \vee Q \\ T \\ \neg R \end{bmatrix}$$

2) R is now $\neg R$. Add to F.

1. $\neg R$ and $\neg P \vee \neg Q \vee R$
   → $\neg P \vee \neg Q$ ①
2. P and $\neg P \vee \neg Q$
   → $\neg Q$
3. $\neg T \vee Q$ and $\neg Q$
4. → $\neg T$
5. T and $\neg T$
   → Empty contradiction found.



A contradiction occurs when a clause becomes so restricted that there is no way it can be true, this is indicated by generation of empty

Basically proof by contradiction.

Scanned with CamScanner

Unification Algorithm

The matching rules are simple, diff const or predicates cannot match, identical ones can. A var can match another var, const or predicate expression with the restriction that the expression must not contain any instance of the var being matched.

Eg $P(x, x)$ and $P(y, z)$

y/x means

$P(y, y)$, z/y

$P(y, z)$

→ matched.

Unify (L_1, L_2)

1. if L_1 & L_2 are both variables or const :
   1. if L_1 and L_2 are identical : return NIL
   2. else if L_1 is variable and L_1 occurs in L_2 : return fail
   3. else return $L_2 \setminus L_1$
   4. else if L_2 is variable and L_2 occurs in L_1 : return fail
      else return $L_1 \setminus L_2$
   4. else return fail.
2. if initial Predicate symbol in L_1 & L_2 are Not identical: return fail.
3. if L_1 and L_2 have diff No of args : return fail.
4. Set "SUBST" (substitution list) = NIL
5. for i = 1 to len(arg(L_1)) :
   1. call unify (arg(L_1)[i] , arg(L_2)[i]) putting result in S
   2. if S is fail : return fail.
   3. if S is Not NIL : i) Apply S to remainder of L_1 & L_2
      ii) SUBST. APPEND (S)
6. Return SUBST.

Scanned with CamScanner

21/3/18

Resolution for Predicate logic.

1. man (Markus)
2. Pompeen (Markus)
3. born (Markus, 40)
4. $\neg$ man (X_1) $\vee$ mortal (X_1)
5. $\neg$ Pompein (X_2) $\vee$ died (X_2, 79)
6. erupted (Volcano, 79)
7. $\neg$ mortal (X_3) $\vee$ $\neg$ born (x_3, t_1) $\vee$ $\neg$ gt ($t_2 - t_1$, 150) $\vee$ died (x_3, t_2)
8. now = 2018
9. ~~not~~ $\neg$ alive (x_4, t_3) $\vee$ $\neg$ died (x_4, t_3)
10. dead (x_5, t_4) $\vee$ alive (x_5, t_4)
11. $\neg$ dead (x_6, t_5) $\vee$ $\neg$ gt (t_6, t_5) $\vee$ dead (x_6, t_6)

- Prove $\neg$ alive (Narcus, now)

alive (Narcus, now) — 9

markus \ x_4
now \ t_3

11. — $\neg$ dead (Narcus, Now)

~~x_6~~ markus \ x_6
now \ t_6

$\neg$ dead (Narkus, t_5) $\vee$ $\neg$ gt (now, t_5) — 5.

79 \ t_5

Narkus \ x_2
~~now~~ \ t_5

$\neg$ Pompein (markus) $\vee$ gt (~~now~~ 2018, 79)
5 (true)

$\neg$ pompein (markus) — 2.

Empty

RIP Mark boi

Scanned with CamScanner

Prolog

Natural languag of AI PROgramming LOGic

- A prolog program, unlike other languges compreses a discription of the problem using facts and rules. Execution is more of defining a goal that forces the prolog inference engine to search and move in a direction that yeilds one or more solution that ~~too~~ help to achive the goal.

Eg delicious (cakes)
" (pickles)
" (briyani)
Relish (Priya, coffee)
likes (Priya, Food) if delicious (Food)
spicy (pickles)
likes (Prakash, food) if dicious (food) and spicy (Food)

There in prolog will be

```
1> delicious(Cakes).
   "  (  ).
   likes(priya, Food) :- delicious (Food).
   likes (prakash, Food) :- delicious (Food), spicy (Food).
```

comment % /* */

On executing we get a prompt for a query.

Scanned with CamScanner

?- delicious(Food).
← Food = Cakes (;) → this'll give next ans while there are more.

?- likes(Priya, Food).

Priya Relishes Coffee & also likes pickles

?- Relishes(Who, coffee), likes(Who, pickels)
← Priya

Variable Creation Rule

Must begin with upper case and then any letter, digit & _

- Anonymous Variables : '_' returns the other arg, not the answer.

Eg car(maruti, white).
car(feat, black).
car(_, Color)
← white, black.

Control Structure

p :- a, b, c p is true if a, b, c are all true.
p :- a, b, c ; d, e, f p is true if a, b, c is true or d, e, f is true

a :- b → c ; d (like ?: in C++)

< /2	(<=)	rand(x) (0-x)
> /2	(>=)	a(x, y) => x & y
=/2	(==)	o(x, y) = x | y
=\=/2	(/=)	x(x, y) => x xor y.
is/2	(assignment)	

Scanned with CamScanner

Prolog is basically DFS and Backtracking.

3/4/18

Matching (in Prolog)

Predicate : likes (priya, coffee)

Query.

?- likes (who, coffee)

who → variable

output who = priya.

found (priya, driving)
found (prakash, cartoon)
?

?- likes (who, coffee) , found (who, driving)

—out who = priya.

?- likes (who, coffee) , found (who, cartoon)

after who is substituted with priya, found is matched, who is ~~to~~ matched but cartoons is not so it backtracks and finds another value of who.

The scanning is done top down.

Scanned with CamScanner

Functionality

```
likes(prakash, x) :- edible(x), tastes(x, sweet)
tastes(choco, sweet)
tastes(gourd, bitter)
tastes(toffee, sweet)
edible(chocolate)
"      (gourd)
"      (toffee)
```

```
?- likes(prakash, x)
      x = chocolate, toffee.
```

Cuts in Prolog

```
do :- p(N), q(N), nl, write(N), !, r(N).
                 ↑ new line
```

p(1) p(2) p(3) p(4) q(2) q(3) q(4) r(3)	?- do N=1 → p(1) → q(2) × N=2 → p(2) → q(2) → nl → print(2) → r(2) × × ← ! ← backtrack

```
car(palio, black, 450,000)
car(Hyundai, silver, 300,000)
car(fiat, silver, 400,000)
car(Maruti, black, 100,000)
get_car(Col, cst_lets_than) :- car(name, color, cost), !, cost < cst_l_t, nl,
                                write(Name).
```

Scanned with CamScanner

?- get_car(black, 350,000).
output fail.

Loop in Prolog.

```
getinput :-
    redo,
        grab(ch), string_chars(c, [ch]),
        write (c) , ch= /213,
    redo                    (=/2 is ==)
```

Recursion

```
count (0).        [base value]
count (N) :-
    write ('old N= '), write(N) , nl,
    N1 is N-1
    write ('New N=') , write(N1) nl,
    count (N1).
```

? count (5)
5 \n 4 \n 4 \n 3 \n 3 \n 2 \n 2 \n 1 \n 1 \n 0 .

Eg recursive program : factorial.

```
fact ( 1,1).
fact ( N, f_of_N) :-
        Φ is N-1,
        factorial ( Φ, f_of_Φ)
        write ( f_of_Φ),
        f_of_N is N * f_of_Φ .
```

?- fact (4,X) .

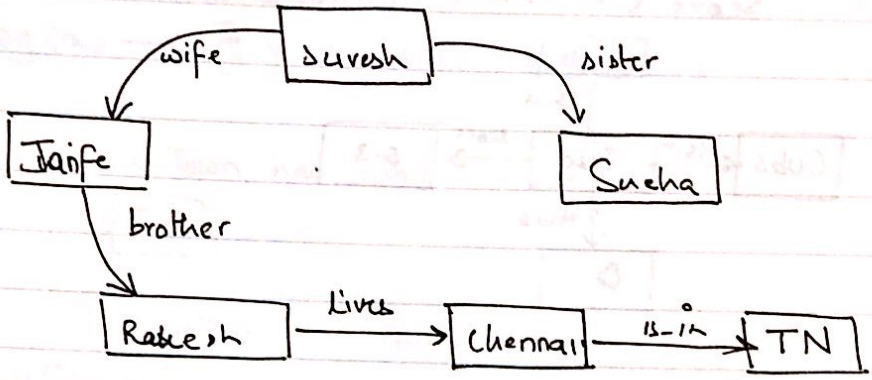
Scanned with CamScanner

4/4/18

Knowledge Structures

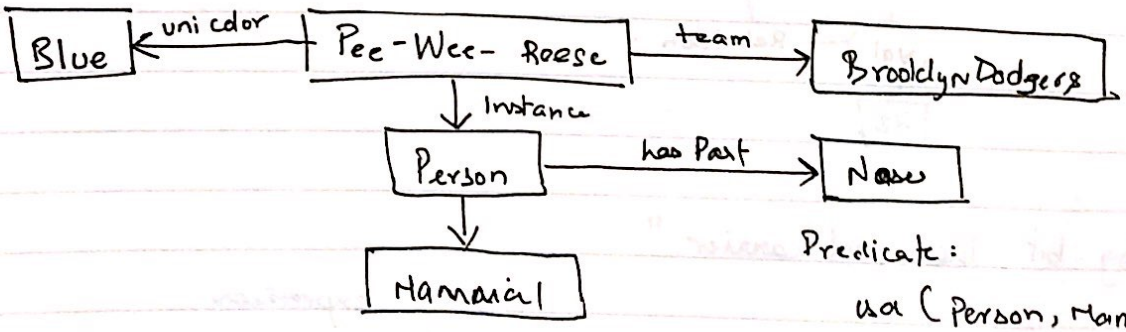
Semantic Nets

Graphical form of representation



- Marker Passing
- Activation Propagation
- Spreading Activation
- Intersectional Search

} Kaam ek Naam anek (same algo)



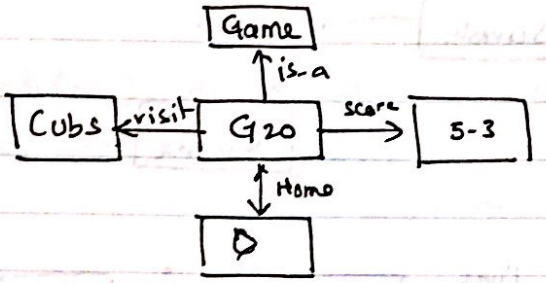
Predicate:

usa (Person, Mammal)
instance (P-W-Reese, Person)
has part (Person, Nose)
Team (B-D, P-W-R)
UC (P-W-C, Blue)

Scanned with CamScanner

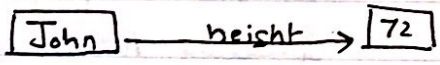
3 or more place predicates can be converted to binary by creating an new object by represent the P.S and then introducing bin-Pred to describe the relationship to this new object of each of the original objects.

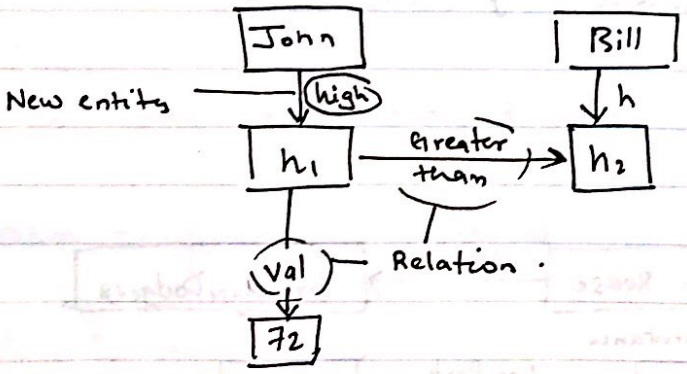
Eg: Score (D, C, 5:3)



There are two types of links.

- Link that defines an entity
- Link that links existing ones

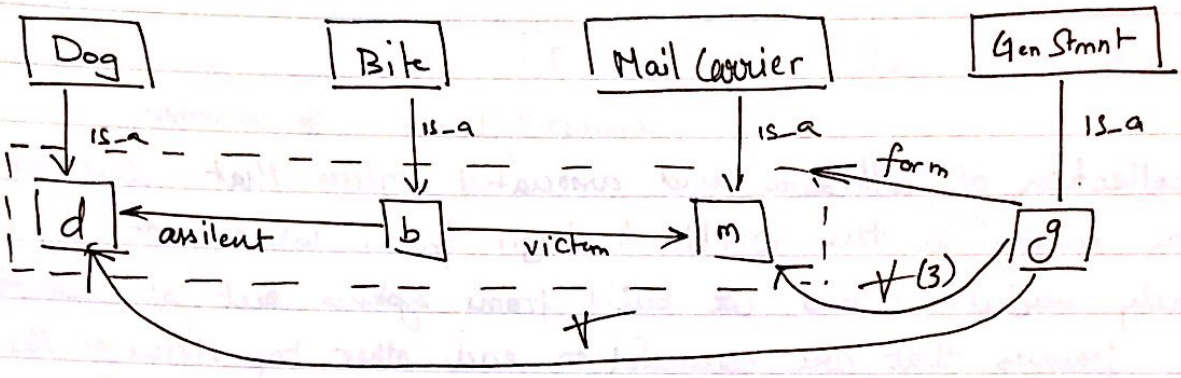




"The dog bit the mail carrier"

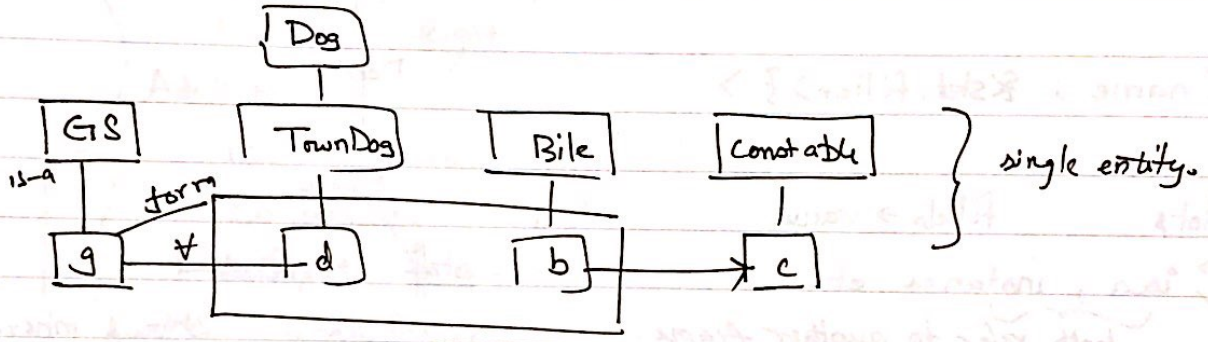
Suppose we want to represent simple quantified expressions rep in semantic nets. One way to do this is to partition the S.N into a heirarchical set of spaces each of which corresponds to the scope of one or more variables.

Scanned with CamScanner



1) $\forall x: \; dog(x) \rightarrow \exists y: \; m_c(y) \wedge bite(x, y)$

2) Every dog in the Town has bitten the constable.

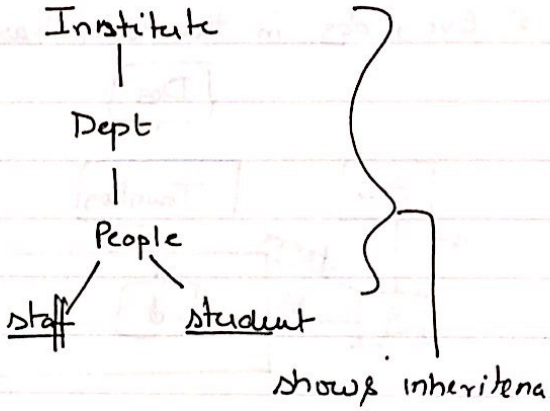


3) Every dog has bitten Every M_C

10/4/18

Frames [Marvin Minsky - MIT]

A collection of attributes and associated values that describe some entity in the world. A single frame taken alone is rarely useful instead we build frame systems out of collections of frames that are connected to each other by virtue of the fact that the values of an attribute of one frame may be another frame.



< name : {<slot, filler>} >

slots → fillers → values

↳ is_a , instance etc

both refer to another frame.

Can have attached procedure

1. Fwd chaining [if_added]
2. Backward chaining [if_needed]

if_added : a filled is added, now do something.

if_needed : a filler is only called when needed.

Elephant S
: is_A mammal
: Color : Grey

Mammal
⋮
eyes := 2

Royal Elephant
: is_A Elephant : is_A AA
: color : white

African Animal
|
T

dyde
: Instance_of : Royal Elephant.

Example

Person
is_a : Mammal
cardinality : 6,000,000,000
handed : Right

Adult Male
is_a : Person
cardinality : 2×10^9
height : 5-10

ML_baseball_Player
is_a : Adult_Male
Cardinate : 624
height : 6-1
bats : Equal-handed (inherit)
batting avg : 0.252
team : -
Uniform : .

Fielder
is_a : ML_baseball_Player
cardinality : 876
batting avg : 0.262

Pee-wee-Rule
: instance : Fielder
height : 5-10
bat : Right
avg : 0.309
team : Dodgers

ML-B-Team
is_a : Team
Cardinality : 26
team size : 24

Dodgers
instance : ML-B-Team
team_size : 24
Manager : some_guy
Players : {_ _ _}

Scanned with CamScanner

generic frame contain default Knowledge & are used to create individual frames.

Eg : Trip Planner

Trip
- TravelPart
- Lodging

Trip27
INSTANCE OF Trip
First_leg. → [calls the entry producudure] [≡ NEXT→]

Scanned with CamScanner

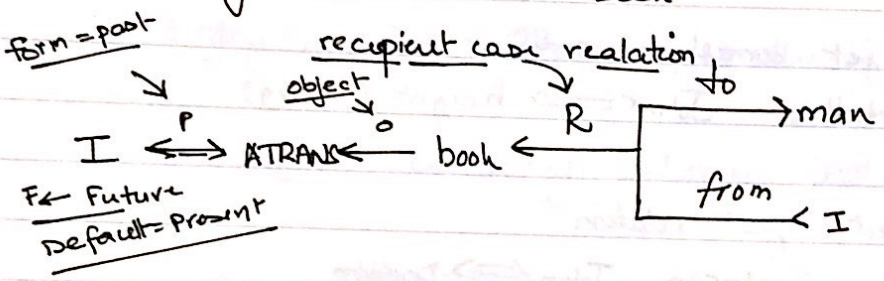
11/4/18

CDC

Conceptual Dependency

Theory of how to represent the kind of knowledge about events that is usually contained in Natural language sentence. The goal is to represent the knowledge in a way that facilitates drawing inferences from the sentences and is independent of the language in which the sentences were originally stated.

" I gave the man a Book "



— Primitive Actions

1. ATRANS : Transfer of abstract Relationship.
2. PTRANS : Transfer of physical Location.
3. PROPEL : Application of force.
4. MOVE : Movement of body part of owner.
5. GRASP : Grasping of an object by an actor
6. INGEST : Taking something in.
7. EXPEL : Expulsion of something from the body.
8. MTRANS : Tranfer of mental info
9. MBUILD : Build new from old.
10. ATTEND : Attention.

Scanned with CamScanner

Entities / Entries

ACT – Actions

PPs – object & Picture Producers

PA – Modifier of PP . Picture aiders.

Rules

1. PP $\Longleftrightarrow$ ACT : object is doing some action.

   Jean Ran = Jean $\overset{P}{\Longleftrightarrow}$ Ran

2. PP $\Longleftrightarrow$ PA : objet connect to modifier

   John is tall = John $\Longleftrightarrow$ height (> avg)

3. PP $\Longleftrightarrow$ PP : object - object relation

   John is a Doctor = John $\Longleftrightarrow$ Doctor

4. PA $\longrightarrow$ PP : descriptions

   A nice boy = nice $\longrightarrow$ boy

5. PP $\Rightarrow$ PP : dependancy.

   John's dog : dog $\Longrightarrow$ John

6. ACT $\overset{o}{\longleftarrow}$ PP : Person - action cause - effect

   John pushed the car John $\overset{P}{\Longleftrightarrow}$ propel $\overset{o}{\longleftarrow}$ car

7. ACT $\overset{o}{\longleftarrow}$ PP, PP object → from PP to PP.

⋮

From Book

Scanned with CamScanner

Scripts

- Extension of frames.
- Sequence of Events.

- A script is a structure that describes a stereotype seq of events in a perticular context.
- A script consists of set of slots. associated with each slot may be some info about what kinds of values it may contain as well as default values to be used if no other info is available.

<u>Entry Condition</u> : They trigger the event.

<u>Results</u> : The set of outcomes after running the script that will be in front of us

<u>Props</u> : slots represent objects involved in the script.

<u>Roles</u> : People involved in the script.

<u>Track</u> : Specific version of script that contains some common slots from the other version

<u>Scenes</u> : The actual seq of events that occur.

Scanned with CamScanner

Script RESTAURANT

Track : coffee shop

Prop : Tables
Menu
F = Food
check

Roles : Money
S = customer
W = Waiter
C = Cook
M = cashier
O = owner

Scene - 1

S PTRANS S in to restaurant

S ATTEND eyes to table

S MBUILD where to sit

S MOVE S to sitting pos

Scene - 2

S PTRANS menu to S.

Entry : S is hungry
S has money

Result : S has less money
O has more money
S is not hungry
S is pleased (optional)

Scanned with CamScanner

12/4/18

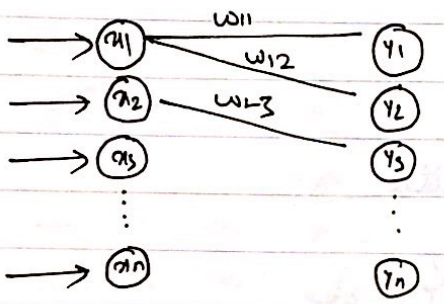
Neural Networks

- MP_ neuron
- Linear Seprability
- Hebb network.

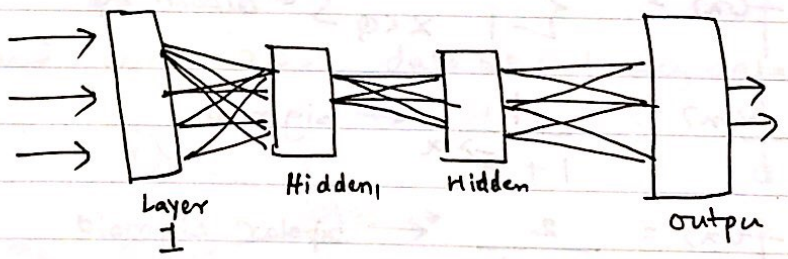
$Net = \sum x_i w_i$

$y_{out} = f(Net)$

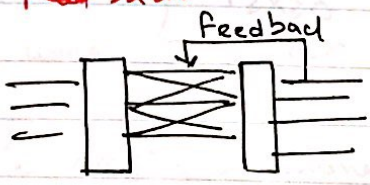
1. Single layer FF Network



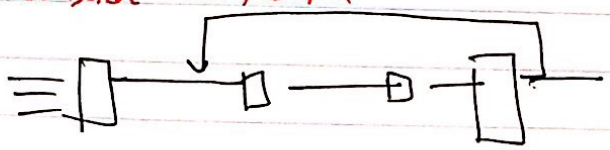
2. Multi Layer FF Network



3 - Single layer Feedback



4. Multi Layer feedback Network



Scanned with CamScanner

Learning

Broad

1. Parameter : update weights
2. Structure : change the structure of Net

finer

1. Supervised : Classification is using known desired output
2. Unsupervised : Not output defined learning is ~~an~~ self supervised
3. Reinforcement : — copy from PA —

Activation

1. $f(x) = \{x\}$ ← identity

2. $f(x) = \begin{cases} 1 & x \geq \theta \\ 0 & x < \theta \end{cases}$ ← step

3. $f(x) = \begin{cases} +1 & x \geq \theta \\ -1 & x < \theta \end{cases}$ ← bipolar step

4. $f(x) = \dfrac{1}{1+e^{-\lambda x}}$ ← sigmoid

5. $f(x) = \dfrac{2}{1+e^{-\lambda x}}$ ← bipolar sigmoid

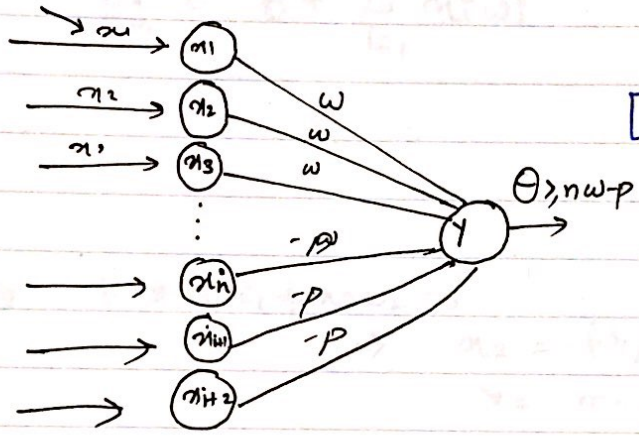
6. $f(x) = \begin{cases} 1 & x > 1 \\ x & 0 \leq x \leq 1 \\ 0 & x < 0 \end{cases}$ ← ramp

1. Bias wts : To prevent overfitting.

Scanned with CamScanner

MP Neuron

- McCulloch Pits Neuron
- first NN
- No training, only analysis and based on this updating is done
- This uses binary step function
- The wts can be +ve / -ve [existential] due to the activation.

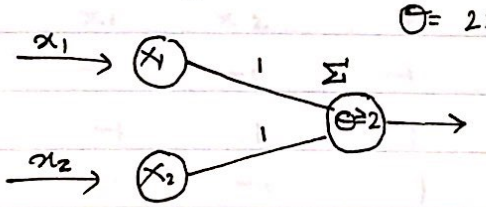


$\theta \geq nw - p$

[all wts of one kind will have the same magnitude]

Q1 Implement and using MPN. data set = binary data.

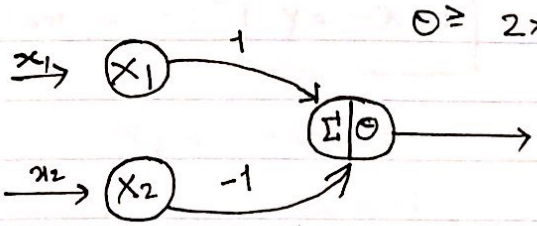
$\theta = 2 \times 1 - 0 = 2$



x_1	x_2	Y	Σ
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	2

Q2 implement ~~NAND~~ AND NOT using

$\theta \geq 2 \times 1 - 1 = 1$



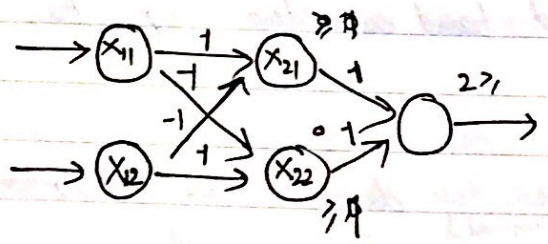
x_1	x_2	Y	Σ
0	0	0	0
0	1	0	-1
1	0	1	1
1	1	0	0

Scanned with CamScanner

$2 \times 0 - 2 = \quad x \geq 2$

Q3 Implement XOR using NP

→ GG WP ←



X_1	X_2	Y
0	0	0
0	1	1
1	0	1
1	1	0

Scanned with CamScanner

17/4/18

Linear Seperability -

The seperation of i/p space in 2 regions is based on whether network responce is +ve or -ve.
A decision line is drawn to seperate +ve & -ve responces
The decision line may be called as the decision making line / decision support line.

$$y_{in} = b + \sum_{i=1}^{n} x_i w_i$$

$$y_{in} \geq 0$$

$y_{in} = 0$

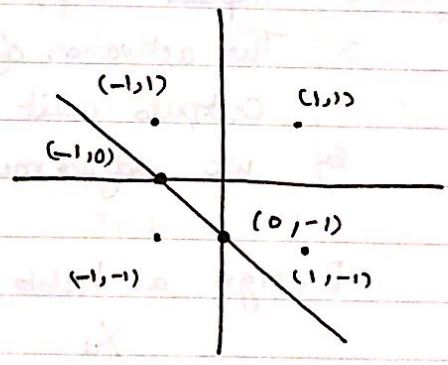
$\Rightarrow \quad b + x_1 w_1 + x_2 w_2 = 0$

$\Rightarrow \quad x_2 = -(w_1/w_2)x_1 - b/w_2$

$y = mx + c.$

Using lin / Sep, obtain response for Or gate for bipolar i/o.

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



$m = \Delta y / \Delta x = -1 \quad \boxed{y = -x - 1}$

$x_2 = -1 x_1 - 1$

$\Rightarrow \quad \frac{-w_1}{w_2} = -1 \quad ; \quad \frac{-b}{w_2} = 1$

$w_1 = w_2 = b = 1$

Scanned with CamScanner

Hebb Network

Change in synaptic gap b/w neuron is when an axon of cell A is near enough to excite B and repeatedly/permanently takes plce in firing it. Some growth/metabolic change occurs in one or both such that A's efficiency as one of the cells firing B increases.

$$\boxed{w_i' = w_i + xy} \; ; \; \boxed{b = b_0 + y}$$

binary data cannot be used, bipolar data is used.

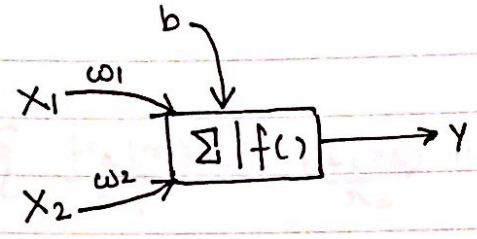
Algorithm

1. Initialize the wts as 0.
2. Repeat steps 3-5 for each input vector & target.
3. The activation of input layer neuron is identity.
4. Outputs unit are set to targeted value
5. wt adjustment & bias adjustment is done.

Q Design a hebb Net for AND function

X_1	X_2	Y
-1	-1	-1
-1	1	-1
1	-1	-1
1	1	1

Scanned with CamScanner



$\omega_1 = \omega_2 = b = 0$

1) $X_1 = 1 \quad X_2 = 1 \quad Y = 1$

$\omega_1 = 0 + 1*1 = 1$ $\qquad$ $Y_{out} = 3$

$\omega_2 = 0 + 1*1 = 1$

$b = 0 + 1 = 1$

— $X_1 = 1 \quad X_2 = 1 \quad b = 1$

2) $\omega_1 = 1 + -1*1 = 0$ $\qquad$ $Y_{out} = -2$

$\omega_2 = 1 + -1*-1 = 2$

$b = 1 - 1 = 0$

1.

3) $\omega_1 = 0 + (-1)(-1) = 1$ $\qquad$ $Y_{out} = -1$

$\omega_2 = 2 + (1)(-1) = 1$

$b = 0 - 1 = -1$

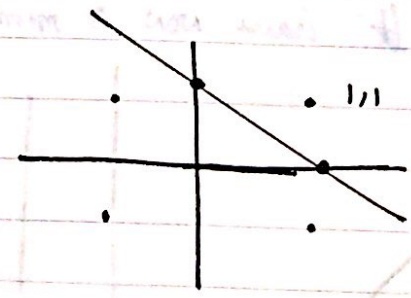
4) $\omega_1 = 1 + (-1)(-1) = 2$ $\qquad$ $Y_{out} = -2$

$\omega_2 = 1 + (-1)(-1) = 2$

$b = 0 - 1 + -1 = -2$

7, 0

makes = seperable



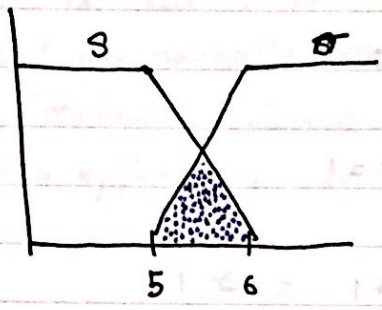
Scanned with CamScanner

19/4/18

Fuzzy Logic

Crisp vs Fuzzy

A $\underset{\sim}{A}$



A fuzzy set A in the universe of discourse. can be defined as a set of ordered Pairs given by

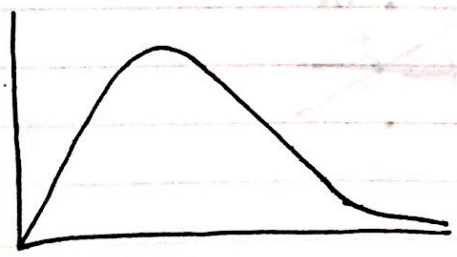
$$\underset{\sim}{A} \{ (x, \mu_A(x)) \mid x \in U \}$$

Universe of discourse : Range of all possible values that comprise input.

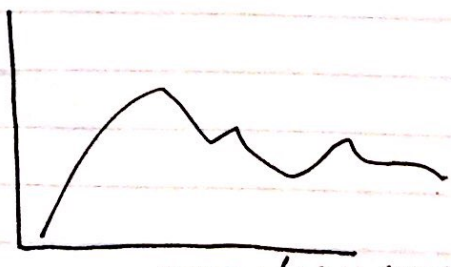
Number function μ_A : forms the basis of fuzzy set

$$\mu_A \rightarrow [0,1]$$

Support : Support S of set F in universal crisp set U is that set which contains all elements of U s.t the ~~ff~~ have non 0 membership value in F.



convex



concave/non convex

Scanned with CamScanner

CH-7

$$\underset{\sim}{A} = \left\{ \frac{\mu_A(x_1)}{x_1} + \frac{\mu_A(x_2)}{x_2} + \dots + \frac{\mu_A(x_i)}{x_i} \right\}$$

1. Unions : $AB = \{ \mu_{A+B} = \max(\mu_A(x) \cdot \mu_B(x)) \}$
2. intersect $A \cap B = \{ \mu_{AB} = \min(\mu_1 x_1, \mu_3 x_3) \}$

3. $\mu_A^{+}(x) = \{ 1 - \mu(A) \}$

4 Sum : $\mu_{A+B}(x) = \{ \mu_A(x) + \mu_B(x) - \mu_A(x)\mu_B(x) \}$

5 Bounded Sum: $\mu_{A \oplus B}(x) = \{ \min(1, \mu_A(x) + \mu_B(x) \}$
6 Bounded diff: $\mu_{A \ominus B}(x) = \{ \max(0, \mu_A(x) - \mu_B(x) \}$

Associativity, Distributity, commutativity ... hold.

- Genetic Algorithms
- Non Monotonic & Probalistic Reasoning
- NLP (6 phases) → Page 285 R&K. 291
- Expert Systems Ch - 20.

Scanned with CamScanner