Constraint Satisfaction Problem

1) Graph Colouring
2) Sudoku

A constraint Satisfaction Problem consists of

* a set of variables
* a domain for each variable &
* a set of constraints.

* The aim is to choose a value for each variable so that the resulting possible world satisfies the constraints, we want a model of the constraints.

* A finite CSP has a finite set of variables and a finite domain for each variable.

Given a CSP, there are a no. of tasks that can be performed.

* Determine whether (or) not there is a model
* Find a model
* Find all of the models (or) enumerate the models
* Count the no. of models
* Find the best model
* Determine whether some ~~st~~ statement holds in all models.

CSP consists of three components V, D, C

V → set of variables $\{v_1, v_2, \ldots, v_n\}$

D → set of Domains $\{D_1, D_2, \ldots, D_n\}$

C → set of constraints that specify allowable combination of values.

C_i = (scope, relationship)

where,

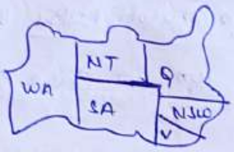
scope – It is a set of variables that participate in constraints.

Relationship – Is the relationship that defines the values that a variable can take.

* A solution to a csp is consistent & complete assignment.

* Allows useful general-purpose algorithms with more power than standard search algorithms.

Ex:- Graph Colouring :-



variables :- WA, NT, SA, Q, V, NSW

Domain :- {red, green, blue}

constraints :- adjacent regions must have different colors

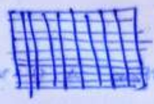
variable ______

Domain ______

Constraints ______

Solution :- { WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = red }

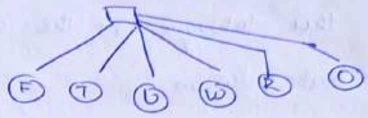
Ex'- Sodoku.



Ex-2:- CSP's

Cryptarithmetic Puzzles

$$\begin{array}{r} T\ W\ O \\ +\ T\ W\ O \\ \hline F\ O\ U\ R \end{array}$$



constraints.

$\Rightarrow$ $O + O = R + 10 \times C_{10}$

$\Rightarrow$ $C_{10} + W + W = U + 10 \times C_{100}$

$\Rightarrow$ $C_{100} + T + T = O + 10 \times C_{1000}$

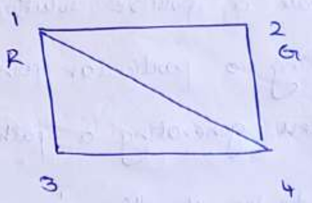
$\Rightarrow$ $C_{1000} = F$

CSP using Backtracking

$V = \{1, 2, 3, 4\}$

$D = \{$ Red, green, Blue $\}$

$C = \{1 \neq 2, 1 \neq 3, 1 \neq 4, 2 \neq 4, 3 \neq 4\}$



	1	2	3	4
Initial Dom	R,G,B	R,G,B	R,G,B	R,G,B
1=R	R	GB	GB	GB
2=G	R	G	B,G	B
3=G	R	G	G	B

Generate & Test :-

1. Generate & Test search algorithm is a very simple algorithm that guarantee to find a solution, if done systematically and there exist a solution.

~~Algorithm~~ :-

2. In this technique all the solutions are generated and tested for best solution

3. It ensures that the best solutions is checked from all possible generated solutions.

4. Also known as British Museum Search Algorithm

5. It is one of Heuristic technique which follows.
   DFS with Backtracking.

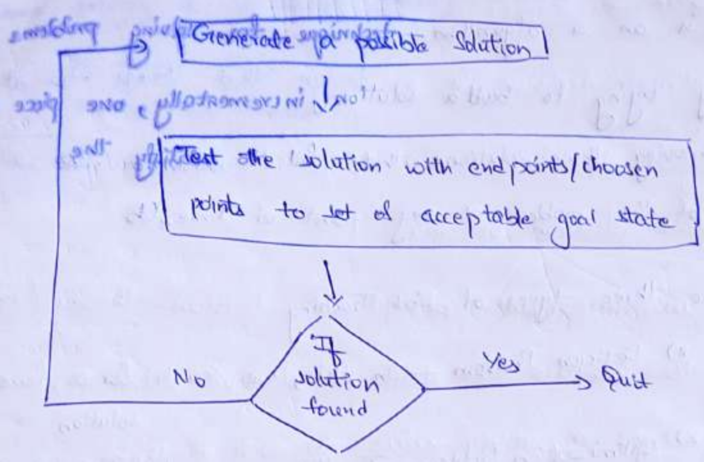
Algorithm :-

1. Generate a possible solution, For some problems, this means generating a particular point in the problem space. For others it means generating a path from a start state.

Test to see if this is actually a solution by comparing the chosen point (or) the endpoint of the chosen path to the set of acceptable goal states.

If a solution has been found, quit, otherwise retur to step 1.

Flowchart :-



* It is a depth first search procedure since complete solutions must be generated before they can be tested.
* It is most systematic form, it is simply an exhaustive search of the problem space.
* It operates by generating solution randomly.

Properties of Generate & Test

1) Comple
2) Non - Redundant
3) Informed

Back Tracking:-

Back Tracking is an a algorithm technique for solving problems recursively by trying to build a solution incrementally, one piece at a time, removing those solutions that fail to satisfy the constraints of the problem at any point of time.

There are three types of Back Tracking

1) Decision Problem — In this, we search for a feasible solution

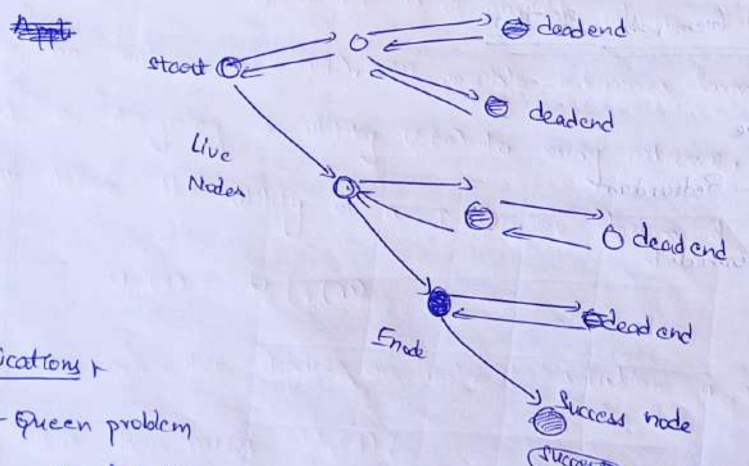
2) Optimization Problem — In this, we search for the best solution

3) Enumeration Problem — In this, we find all feasible solutions.

2 Types of constraints

1) Implicit :- How each element in a tuple should be related.

2) Explicit :- Rules that restrict each element in a set



Applications :-

N-Queen problem

Sum of Subsets

Graph Colouring

Hamilton

Game Playing

Game playing is a search problem defined by.

* Initial state
* Successor function
* Goal test
* path cost / utility / payoff functions.

Naturality of game

* obey's laws of the game
* Characters aware of the environment
* Path finding (A* algorithm)
* Decision making
* Planning

The game AI is the illusion of human Behaviour

* Smart to a certain extent
* Non - repeating behaviour.
* Emotion influences (irrationality, personality).
* Being integrated in the environment

Game AI needs various computer science Desciplines

* knowledge Based Systems
* Machine learning
* Multi - agent Systems
* Computer Graphics & Animation
* Data Structures.

Game Types:-

1) Strategy games
2) Role playing games
3) Action Games
4) Sport Games
5) Adventure games
6) Puzzle games
7) Emulations.

~~Optimal trees~~ in

Optimal Desision in Games

An optimal decision is a design that leads to at least as good as known (or) expected outcome as all others available decision options.

It is an important concept in decision-theory / ai, in order to compare the different decision outcomes, one commonly assigns a utility value to each of them.

Utility — is only an arbitrary term for quantifying the desirability of a particular decision outcome and not neccessarily related to "usefulness".

Mathematically :-

Each decision d is a set 'D' of available options will lead to an outcome $o = f(d)$. All possible outcomes form the set O. Assigning a utility $U_O(o)$ to every outcome, we can define the utility of a particular design d as

$$U_D(d) = U_O(f(d)).$$

We can then define an optimal solution d_{opt} as one that maximizes $U_D(d)$:

$$d_{opt} = \arg\max_{d \in D} U_D(d).$$

where, predicting the outcomes 'o' for every decision d;

assigning a utility $U_D(o)$ to every outcome 'o'.

finding the decision d that max $U_D(d)$.

Min Max Algorithm

* It is a specialized search algorithm that returns optimal sequence of moves for a player in zero-sum game.
* MMA is a recursive (or) backtracking algorithm which is used in decision-making and game theory. It provides an optimal move for the player assuming that opponent is also playing optimally.
* MMA is mostly used in game playing. Ex:- Chess, Checkers Tic-tac-toe, etc.
* In this algorithm two players play the game, one is called MAX & other is called MIN.
* Both players are opponent of each other, where MAX will select the maximized value & min will select the minimum value.
* MMA follows depth-first search.

$$\begin{bmatrix} Max = -\infty \\ Min = \infty \end{bmatrix} \begin{matrix} [\text{worst values}] \\ [\text{initial values}] \end{matrix}$$

Properties of Min-Max Algorithm

* Complete :- It will definitely find a solution (if exist).
* Optimal :- MMA is optimal if both players play optimally.
* Time Complexity :- As it is a DFS

The time complexity = $O(b^m)$

where b - branching factor

m - max depth.

REDMI NOTE 9
CLICK BY AKHIL

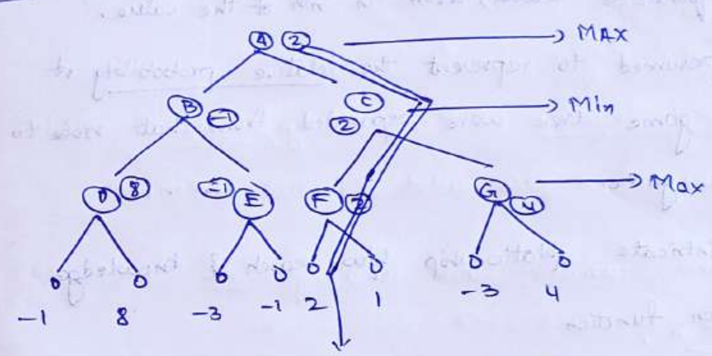
Space Complexity — Space Complexity of Min-Max Algo $\Rightarrow O(bm)$

Limitations:

1) Slow for complex games (Chess)
2) Huge branching factor

Limitations can be improved in "Alpha-beta Pruning"

Ex:-



Path :- A → C → F

Evaluation Function:-

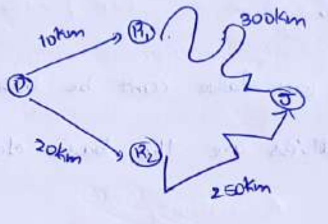
* An evaluation Function is also known as heuristic evaluation function (or) static evaluation function, is a function used by game-playing computer programs to estimate the value (or) goodness of a position in a game tree.
* A tree of search evaluation is usually part of a minimax (or) related search paradigm..
* The value is quantized scalar, often in nth of the value.
* The value is presumed to represent the relative probability of winning if the game tree were expanded from that node to the end of the game.
* There is an intricate relationship b/w search & knowledge in the evaluation function.
* Deeper search favors less near-term tactical factors & more subtle long-horizon positional motifs in the evaluation.

There is also a trade-off b/w efficacy of encoded knowledge in the ~~evaluation function~~ & computational knowledge.

Because the evaluation function depends on the nominal depth of search as well as the extensions & reductions employed in the search, there is no generic or stand-alone formulation for an evaluation function.

Heuristic search refers to a search strategy that attempts to optimize a problem by iteratively improving the solution based on a given heuristic function.

Heuristic function, is a function that ranks alternative in search algorithms at each branching step based on available information to decide which branch to follow.



$R_1 \Rightarrow 10+300 = 310km$

$R_2 \Rightarrow 20+250 = 270km$

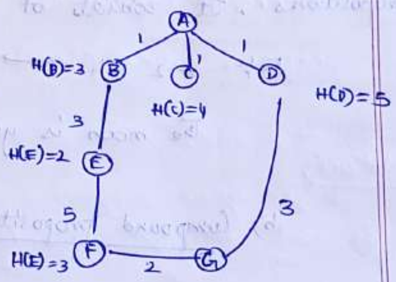
Types of Heuristic

1) Admissible :- In this heuristic function, it underestimates the cost of reaching the goal $\Rightarrow H(n) \le H'(n)$

2) Non - Admissible :- Overestimate the cost of reaching the goal.

$\Rightarrow H(n) > H'(n)$

h(n) is always less than (or) equal to actual cost of lowest cost path from node 'n' to goal.



Sol :- $H(B)=3$; $H(E)=2$, $H(C)=4$, $H(D)=5$

$F(n) = H(n) + G(n)$

$F(B) \Rightarrow 3+1 \Rightarrow 4$

$F(C) = 1+4 = 5$

$F(D) = 5+1 = 6$

$F(E) = 2+4 = 6$

$F(F) = 3+9 \Rightarrow 12$

$F(G) = 2+11 \Rightarrow 13$

$\therefore$ B) $\Rightarrow 3 < 11$

$\rightarrow$ Admissible

$\therefore$ D) $\Rightarrow 5 > 4$

Non Admissible

Propositional Logic

A proposition is a collection of declarative statements that has either a truth value "true" (or) truth value "false". A propositional ~~variables~~ consists of propositional variables and connectives.

* Denoted by capital letters.
* Propositions can be true (or) false can't be both.
* The propositions & connectives are the basic elements of the propositional logic.
* A propositional formula which is always true is called tautology, and it is also called as valid sentence.
* A propositional formula which is always false is called as Contradiction.

Syntax of proposition logic

a) Atomic proposition :- AP are the simple propositions. It consist of a single proposition symbol.

Ex:- 2+2 =4 ⟶ True

The moon is yellow ⟶ False

b) Compound propostion :- CP are constructed by combining simpler (or) atomic propositions, using parenthesis () & logical connectives.

Ex:- It is hot today, lets stay indoor.

Logical connectives :-

LC are used ~~to~~ connect ~~two~~ simpler propositions (or) representing a sentence logically.

1) Negative :- A sentence such as $\sim P$ is called negation of P.

A literal can be either (+)ve (or) (-)ve

2) Conjunction :- A sentence which has '$\wedge$' connective such as

$P \wedge Q$ is called conjunction.

Ex:- He is smart and intelectual

3) Disjunction :- A sentence which has '$\vee$' connective, such as

$P \vee Q$ is called disjunction.

Ex:- He is a student (or) faculty.

4) Implication :- A sentence such as $P \rightarrow Q$ is called an implication.

It is known as if-then rules.

Ex:- If it rains, the surrounding will be wet.

5) Biconditional :- A sentence such as $P \Leftrightarrow Q$ is a Biconditional statement.

Ex:- If I am eating, then I am on dining table.

connective symbols	Word	Technical terms	Example
$\wedge$	AND	Conjunction	$A \wedge B$
$\vee$	OR	Disjunction	$A \vee B$
$\rightarrow$	Implies	Implication	$A \rightarrow B$
$\Leftrightarrow$	If & only if	Biconditional	$A \Leftrightarrow B$
$\sim / \neg$	Not	Negation	$\sim A$ / $\sim B$

Properties:

Commutativity:

$$P \wedge Q = Q \wedge P$$

$$P \vee Q = Q \vee P$$

Associativity

$$(P \wedge Q) \wedge R = P \wedge (Q \wedge R)$$

$$(P \vee Q) \vee R = P \vee (Q \vee R)$$

Identity

$$P \wedge True = P$$

$$P \vee True = True$$

Distributive

$$P \wedge (Q \vee R) = (P \wedge Q) \vee (P \wedge R)$$

$$P \vee (Q \wedge R) = (P \vee Q) \wedge (P \vee R)$$

De-Morgan's law:

$$\sim (P \wedge Q) = (\sim P) \vee (\sim Q)$$

$$\sim (P \vee Q) = (\sim P) \wedge (\sim Q).$$

Limitations:

* We cant represent like ALL, some (or) none with propositional logic.
* Propositional logic has limited expressive power.
* In propositional logic, we cant describe statements in terms of their properties (or) logical relationships.

REDMI NOTE 9
CLICK BY AKHIL

First Order logics :-

* FOL is another way of knowledge representation in A.I
* FOL is sufficiently expressive to represent the natural language statements in a concise way.
* FOL is also known as Predicate Logic (or) First Order - Predicate logic.
* FOL can develop information about the objects & relation b/w objects ~~in an~~ using a power-full language.
*
  Objects :- A, B, people, number, colors etc.,

  Relations :- It can be unary ~~& binary numbers~~, n-ary numbers

  Functions :- Father of, best friend.
* As a natural language, FOL has two parts
  * Syntax
  * Semantics.

Syntax :-

FOL determines which collection of symbols is a logical expressions in first-order logic.

Basic Elements :

Constant :- 1, 2, A, Akhil, . . .

variable :- x, y, z, a, b . . .

Predicates :- Brother, Father, >, . . .

Functions :- sqrt, leftlegOf, . . .

connectives :- $\wedge, \vee, \neg, \Rightarrow, \Leftrightarrow$

Equality :- ==

Quantifier :- $\forall, \exists$

Atomic Sentences :-

This sentences are formed from a predicate symbol followed by a parenthesis.

Ex:- A & B are sisters

$\Rightarrow$ sisters(A, B)

Complex Sentences :-

Are made by combining atomic sentences using connections.

Quantifiers in First-order logic

A quantifier is a language element which generates quantification & quantification specifies the quantity of specimen in the universe of discourse.

a) Universal Quantifier :- (for all, everyone, everything)

b) Existential quantifier :- (for some, at least one).

Points to remember :-

* The main connective for universal quantifier $\forall$ is implication $\rightarrow$.

* The main connective for existential quantifier $\exists$ is and $\wedge$.

Properties of Quantifiers

* In universal quantifier, $\forall x \forall y$ is similar to $\forall y \forall x$.
* In Existential quantifier, $\exists x \exists y$ is similar to $\exists y \exists x$.
* $\exists x \forall y$ is not similar to $\forall y \exists x$.

Substitution :-

* Substitution is an operation allowing to replace some variables occurring in a formula with terms.

* The goal of applying a substitution is to make a certain formula more specific so that it matches another formula.

* Substitution allows unification of formulae.

* A substitution 'σ' is any finite mapping of variables into terms of the form.

$$\sigma : V \longrightarrow TER$$

* Any finite substitution can be represented as

$$\sigma :- \{x_1/t_1, x_2/t_2, \ldots, x_n/t_n\};$$

where t_i is a term to be substituted for variable x_i, $i = 1, 2, \ldots, n$.

* ϕ_σ is the formula (term) resulting from simultaneous replacement of the variable of ϕ with the appropriate terms of σ.

* Any substitution σ ($\sigma : V \rightarrow TER$) is extended to operate on terms and formulae so that a finite mapping of the form.

$$\sigma : TER \cup FOR \longrightarrow TER \cup FOR$$

Properties of Substitution

1. Let

$E \rightarrow$ denote an expression (formula (or) term)

$\epsilon \rightarrow$ denote an empty substitution.

$\lambda \rightarrow$ be an one-to-one renaming substitution

σ & $\theta \rightarrow$ denote any substitution.

* $E(\sigma\theta) = E(\sigma)\theta$
* $\sigma(\theta\gamma) = (\sigma\theta)\gamma$ associativity
* $E\epsilon = E$
* $\epsilon\sigma = \sigma\epsilon = \sigma$

Unification :-

Unification is a process of determining and applying a certain substitution to a set of expressions (terms (or) formulae) in order to make them identical.

Let $E_1, E_2 \cdots, E_n \in TER \cup FOR$ are certain expressions. We shall say that expressions $E, E_2, \cdots, E_n$ are unifiable if and only if there exist a substitution σ, such that

$$\{E, E_2, \cdots, E_n\}\sigma = \{E_1\sigma, E_2\sigma, \cdots, E_n\sigma\}$$

is a single-element set.

Substitution σ satisfying the above condition is called a unifier (or a unifying substitution) for expressions $E, E_2, \cdots, E_n$.

Ex:- $P(x, F(y))$ —① $P(a, F(g(z))$ —②

① & ② are identical if 'x' is replace with a and 'y' is replaced with $g(z)$

$p(a, F(g(z))$

$[a/x, g(z)/y]$

Conditions for unifications

1) Predicate symbols must be same

2) No. of arguments in both the expressions must be identical.

3). If two similar variables present in same expression, then unification fails.

Algorithm:-

unify (A1, A2)

⇒ ① if A1 (or) A2 is variable/ constant

↳ if A1 & A2 are identical

→ return NIL

↳ Else if A1 occurs in A2 return fail

↳ Else return (A2/A1)}

↳ Check for A2 in A1

↳ fail if A2 occurs in A1

↳ else return (A1/A2)

② if predicate && different arguments → fail

③ Else subst to NIL

④ Loop subst

REDMI NOTE 9
CLICK BY AKHIL

$Q(a, g(x, a), f(y)), \quad Q(a, g(f(b), a), x\}$

Substitute x with $f(b)$ $[f(b)/x]$

$Q(a, g(f(b), a), f(y)), \quad Q(a, g(f(b)), a f(b)\}$

Substitute (b/y) [y is substituted with b]

$[Q(a, g(f(b), a), f(b)), Q(a, g(f(b), a), f(b)\}$