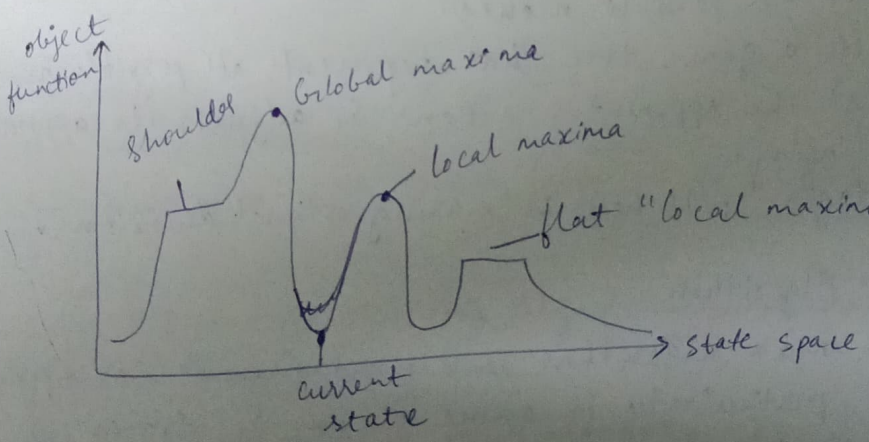
UNIT - III

① Explain about Hill Climbing search algorithm with an example.

A. Hill Climbing Search Algorithm

→ Hill Climbing is a local search algorithm.
→ It uses non-deterministic approach.
→ It continuously moves towards increasing the value to find the best solution (peak).
→ Hill Climbing follows the below steps -

- Initial State : Initial state is where the search starts.
- Neighbouring states : It identifies neighbouring states by making small adjustments.
- Move to Neighbor : If one of the neighboring states offers better solution, then it moves to that state.
- Termination : Repeat this process until no neighboring state is better than the current state. At this point you have reached a local maximum or minimum.



→ There are 3 types of hill climbing. They are -

- Simple Hill Climbing
- Steepest Ascent Hill Climbing
- Stochastic Hill Climbing

→ Advantages : low memory usage
fast at finding local solutions

→ Disadvantages : Can get stuck
No direction to move

→ Applications : used in travelling salesmen problem

→ example -

```
def hill_climbing(f, x0):
x = x0
while True:
neighbors = generate_neighbor(x)
best_neighbor = max(neighbors, key=f)
if f(best_neighbor) <= f(x):
return x
x = best_neighbor
```

---

(4) What is Adversarial search? Discuss about Min-Max algorithm.

A. Adversarial Search

→ AS is a fundamental concept in AI.

→ AS is a technique that helps the agents to make decisions in competitive environments where there are multiple agents with opposite goals.

→ ex - used in games like chess, tic tac toe, etc.

→ In AS, an agent -

- Evaluates all possible moves.
- Constructs a game tree that represents all possible moves
- Predicts the opponent's actions & adjusts its approach accordingly.

→ Advantages : it helps in making optimal decisions.

Min-Max Algorithm

→ The Min-Max algorithm is a fundamental concept in AI, particularly in game theory & strategical decision making.

→ It is designed to minimize the possible loss in worst case scenario & maximize the potential gain.

→ In a 2 player game, one player is the maximizer & another player is the minimizer.

→ The maximizer always tries to maximize their score. whereas the minimize will try to minimize the maximizer's score.

→ This algorithm operates by evaluating all possible moves for both the players.

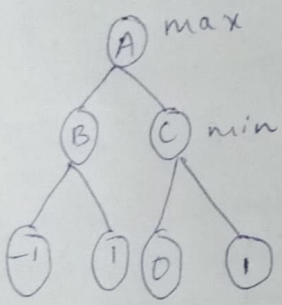
→ ex: chess, tic tac toe, etc.

→ Algorithm

- Generate Game Tree
- Evaluate Terminal Status
- Propogate utility values upwards
- Select Optimal move

→ working of Min-Max Algorithm

- Maximizer's turn: Chooses the maximum values from the child node.
- Minimizer's turn: Chooses the minimum values from the child node.



* The values are 1, -1, 0, 1
* The minimizer will choose min value from -1 & 0 values which is -1
* The maximizer will choose max value from -1 & 0 values which is 0

---

(5) Explain Alpha Beta pruning algorithm with an example.

A: Alpha Beta Pruning Algorithm

→ ABP is not actually a new algorithm. It is an optimization technique for min max algorithm.

→ This algorithm allows us to search much faster & even go into deeper levels in the game tree.

→ It cuts off the branches which need not be searched because there already exists better moves.

→ This algorithm is known as ABP because it passes 2 more extra parameter in the minmax function. They are "alpha" & "beta".

- **Alpha** : It is the best value that the maximizer can guarantee.
- **Beta** : It is the best value that the minimizer guarantee.

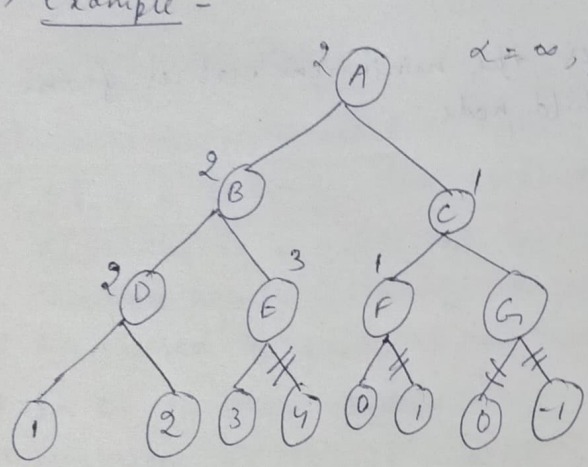
→ Condition for ABP is -

$$\alpha >= \beta$$

where, α → alpha
β → beta

- The maximize will only try to update the value of α.
- The minimizer will only try to update the value of β.

→ **example** -



(10) → Refer (4)

- **Generate the game tree** : Create a tree structure representing all possible moves.
- **Evaluate Terminal Status** : Assign values to the nodes. The values should indicate lose, win or draw.
- **Propogate utility values upwards** : If the maximizer's turn select max values. If its minimizer turn select the min value.
- **Select optimal move** : At the root of the game tree, the maximizer selects the move which leads to the highest value.

(9) The erratic vacuum world - Explain with AND-OR search tree.

A: Erractic vacuum World

→ The EVW is a variant of vacuum world problems.
→ It is used to model & understand decision-making in AI.
→ The word "erractic" means unpredictable.
→ In this world, there are 2 locations, denoted as A & B. The vacuum cleaner must clean both the locations.
→ In the EVW,
- The vacuum has 2 states, clean or dirty in each location (A & B).
- The vacuum has to clean both the locations.

→ Vacuum World Actions
- Move left: Move the vacuum to the left location.
- Move right: " " " " " right ".
- Clean: Clean the current location (if it's dirty).

→ AND-OR Search Tree
- In AI, AND-OR search tree is used to represent problem-solving processes where there are multiple possible actions & some of the actions may lead to alternative solutions or failure.
- In AND-OR tree,
  * OR nodes represent choice b/w different possible actions.
  * AND nodes represent conditions that must be satisfied for a path to be valid.

→ Erratic Vacuum World with AND-OR Search Tree

In this world, both the locations has to be cleaned by the vacuum.

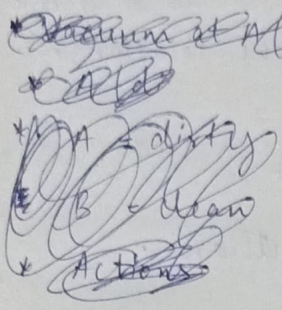
- Initial State: The vacuum starts at location A & the B. The vacuum's goal is to clean both the locations.
- Actions: The available actions are-
  * Move left
  * Move right
  * Clean

- OR Node : It represents choice b/w different possible actions
- AND Node : It represents the condition that must be satisfied for a path to be valid.

→ Example -

There are 2 locations to be cleaned A & B.

- Start State



* A : dirty
  B : clean
* Actions : Clean A, Move Right, Clean B, move left

- First OR Node (At A)
  * Option 1 : Clean A
    The vacuum cleans A & moves right
  * Option 2 : Move Right
    The vacuum moves to B & now it must be clean as it is dirty.
- Second OR Node (At B)
  * Option 1 : Clean B
    Cleans B & both the location are cleans. Hence goal is acheived.
  * Option 2 : Move left
    The vacuum moves back to A & the cycle repeats.

---

(7) Define CSP. Considering N-queen problem explain CSP.

A. CSP

→ CSP stands for Constraint Satisfaction Problems.

→ CSP is a mathematical problem defined by a set of objects whose state must be satisfied by a no. of constraints.

→ CSP plays a crucial role in AI as they help to solve various problems that requires decision-making.

→ Components of CSP -

- Variables : The elements that needs to be assigned values.
- Domains : The possible values that each variable can take.
- Constraints : The rules that define the relationships b/w variables.

→ Applications - Sudoku, Vehicle Routing, Scheduling.

N-Queens Problem

In 4-queens problem, we have 4 queens to be placed on 4*4 chessboard. It should satisfy the constraint that -

- no 2 queens should be in the same row.
- " " " " " " " column.
- " " " " " " " diagonal.

→ Define the Variables : Create 4 variables, one for each column on the chessboard. Each variable represents a queen Q_1, Q_2, Q_3 & Q_4.

→ Define the Domain : $Q_1, Q_2, Q_3 Q_4 = \{1, 2, 3, 4\}$
This represents the 4 rows on the chessboard.

→ Define the constraints : No 2 queens should share the same row.

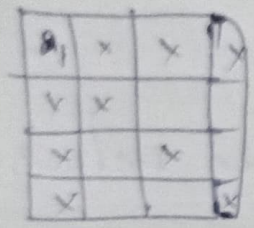
$Q_1 \neq Q_2$
$Q_1 \neq Q_3$
$Q_1 \neq Q_4$
$Q_2 \neq Q_3$
$Q_2 \neq Q_4$
$Q_3 \neq Q_4$

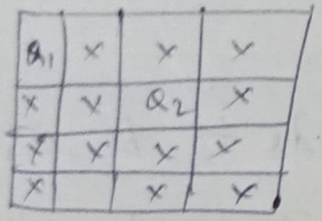
No 2 queens should share the same column.

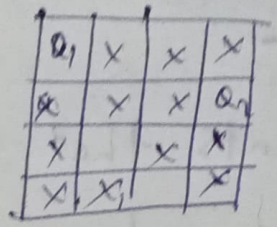
No 2 queens should share the same diagonal.

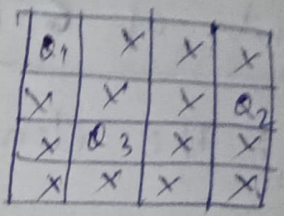
$|Q_1 - Q_2| \neq 1$ $\quad$ $|Q_2 - Q_3| \neq 1$
$|Q_1 - Q_3| \neq 2$ $\quad$ $|Q_2 - Q_4| \neq 2$
$|Q_1 - Q_4| \neq 3$ $\quad$ $|Q_3 - Q_4| \neq 1$

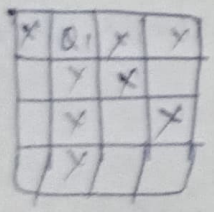
→ Solution:

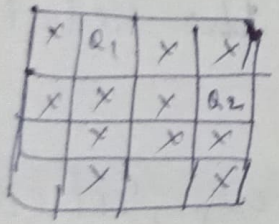


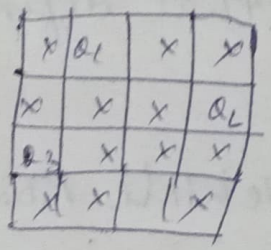


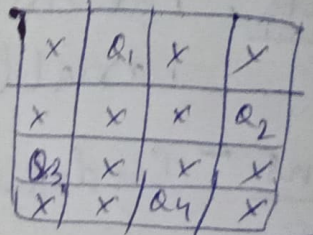












② Elaborate simulated annealing algorithm in detail with an example.

A.

Unit - 3

2. Elaborate simulated annealing algorithm in detail with example.

(A) Simulated annealing is one of the most proffered heuristic methods for solving the optimization problems.

* Kirkpatrick et al introduced SA.
* SA is inspired by the annealing process in metalworking.
* Annealing involves heating a metal to a high temperature and then cooling it slowly to arrange its molecules optimally, minimizing energy.
* A basic optimization algorithm compares the current solution with nearby solutions.
* If a better solution is found, it moves to that solution; otherwise, it stops.
* This can trap the algorithm in local minima (or maxima), missing the best possible solution.
* SA uses two loops : an inner loop & an outer loop.
* It mimics the annealing process, starting with a high "temp." that allows broad exploration & gradually cooling to refine the search.
* If the new solution is better SA accepts it as the new current solution.
* If the new solution is worse, it may still accept it based on a probability, which depends on:

→ The difference b/w the solutions.
→ The current "temperature"

* This randomness helps avoid getting stuck in local minima.

Temp. parameters:-

* High temperature = wider search area, more exploration, higher chance of finding the global minimum.

* Low temp. = Smaller search area, more precision but higher risk of being trapped in local minima.

* Start the iterative process with a high temperature.

* Run the inner loop to explore solutions.

* Gradually lower the temperature.

* Continue until the temp. is very low or a set number of iterations is completed.

* The goal of SA is to balance the exploration and exploitation to find the global minimum efficiently.

function SIMULATED_ANNEALING (problem, schedule) returns a solution state
  inputs problem, a problem
    schedule, a mapping from time to "temperature"
  local variables : current, a node
    next, a node
    T, a "temperature", controlling prob. of downward steps

current ← MAKE-NODE [INITIAL_STATE [problem]]
for t ← 1 to ∞ do
  T ← schedule [t]

if T=0 then return current
next
ΔE ← VALUE [next] ← next
if ΔE > 0 then current ← next
else current ← next only with probability $e^{\Delta E/T}$