UNIT 2

SEARCH ALGORITHMS IN ARTIFICIAL INTELLIGENCE

PROBLEM-SOLVING AGENTS:

- In Artificial Intelligence, Search techniques are universal problem-solving methods.
- Rational agents or Problem-solving agents in AI mostly used these search strategies or algorithms to solve a specific problem and provide the best result.
- Problem-solving agents are the goal-based agents and use atomic representation.

Search Algorithm Terminologies:

1. **Search:** Searchingis a step by step procedure to solve a search-problem in a given search space. A search problem can have three main factors:
   a. **Search Space:** Search space represents a set of possible solutions, which a system may have.
   b. **Start State:** It is a state from where agent begins **the search**.
   c. **Goal test:** It is a function which observe the current state and returns whether the goal state is achieved or not.
2. **Search tree:** A tree representation of search problem is called Search tree. The root of the search tree is the root node which is corresponding to the initial state.
3. **Actions:** It gives the description of all the available actions to the agent.
4. **Transition model:** A description of what each action do, can be represented as a transition model.
5. **Path Cost:** It is a function which assigns a numeric cost to each path.
6. **Solution:** It is an action sequence which leads from the start node to the goal node.
7. **Optimal Solution:** If a solution has the lowest cost among all solutions.

PROPERTIES OF SEARCH ALGORITHMS:

Following are the four essential properties of search algorithms to compare the efficiency of these algorithms:

Completeness: A search algorithm is said to be complete if it guarantees to return a solution if at least any solution exists for any random input.

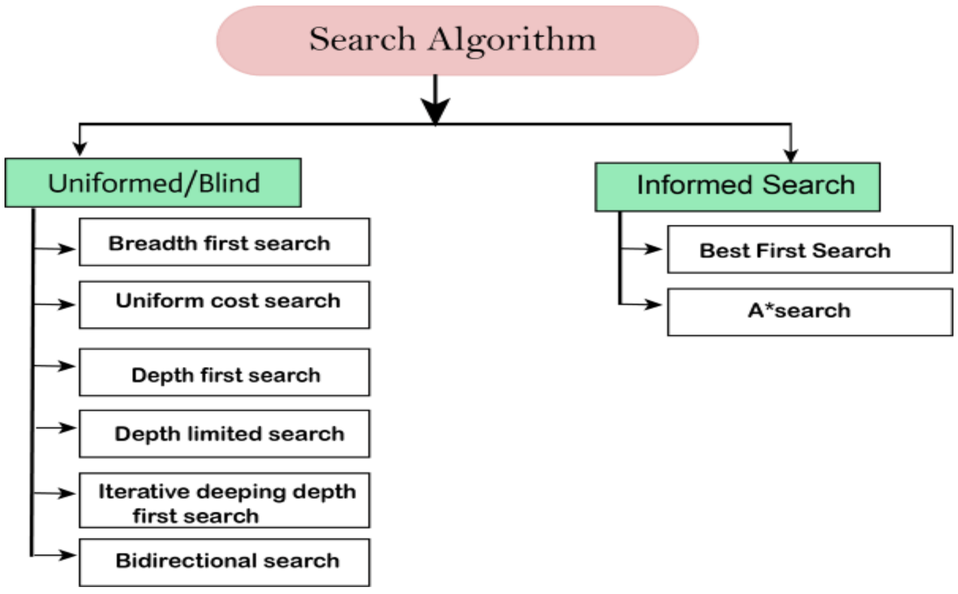
Optimality: If a solution found for an algorithm is guaranteed to be the best solution (lowest path cost) among all other solutions, then such a solution for is said to be an optimal solution.

Time Complexity: Time complexity is a measure of time for an algorithm to complete its task.

Space Complexity: It is the maximum storage space required at any point during the search, as the complexity of the problem.

TYPES OF SEARCH ALGORITHMS

Based on the search problems we can classify the search algorithms into uninformed (Blind search) search and informed search (Heuristic search) algorithms.



UNINFORMED/BLIND SEARCH ALGORITHMS

It can be divided into five main types:

- Breadth-first search
- Uniform cost search
- Depth-first search
- Iterative deepening depth-first search
- Bidirectional Search

1. Breadth-first Search:

- BFS algorithm starts searching from the root node of the tree and expands all successor node at the current level before moving to nodes of next level.
- The breadth-first search algorithm is an example of a general-graph search algorithm.
- Breadth-first search implemented using FIFO queue data structure.
- **Time Complexity:** Time Complexity of BFS algorithm can be obtained by the number of nodes traversed in BFS until the shallowest Node. Where the d= depth of shallowest solution and b is a node at every state.
- **$T(b) = 1+b^2+b^3+\dots\dots+b^d = O(b^d)$**
- **Space Complexity:** Space complexity of BFS algorithm is given by the Memory size of frontier which is $O(b^d)$.
- **Completeness:** BFS is complete, which means if the shallowest goal node is at some finite depth, then BFS will find a solution.
- **Optimality:** BFS is optimal if path cost is a non-decreasing function of the depth of the node.

2. Depth-first Search

- Depth-first search isa recursive algorithm for traversing a tree or graph data structure.
- It is called the depth-first search because it starts from the root node and follows each path to its greatest depth node before moving to the next path.
- DFS uses a stack data structure for its implementation.

Completeness: DFS search algorithm is complete within finite state space as it will expand every node within a limited search tree.

Time Complexity: Time complexity of DFS will be equivalent to the node traversed by the algorithm. It is given by:

$$T(n)= 1+ n^2+ n^3 +\dots\dots\dots+ n^m=O(n^m)$$

Where, **m**= maximum depth of any node and this can be much larger than d (Shallowest solution depth)

Space Complexity: DFS algorithm needs to store only single path from the root node, hence space complexity of DFS is equivalent to the size of the fringe set, which is **O(bm)**.

Optimal: DFS search algorithm is non-optimal, as it may generate a large number of steps or high cost to reach to the goal node.

3. Depth-Limited Search Algorithm:

Depth-limited search can solve the drawback of the infinite path in the Depth-first search. In this algorithm, the node at the depth limit will treat as it has no successor nodes further.

Completeness: DLS search algorithm is complete if the solution is above the depth-limit.

Time Complexity: Time complexity of DLS algorithm is $O(b^{\ell})$.

Space Complexity: Space complexity of DLS algorithm is $O(b\times\ell)$.

Optimal: Depth-limited search can be viewed as a special case of DFS, and it is also not optimal even if $\ell>d$.

4. Uniform-cost Search Algorithm:

- Uniform-cost search is a searching algorithm used for traversing a weighted tree or graph.
- This algorithm comes into play when a different cost is available for each edge.
- The primary goal of the uniform-cost search is to find a path to the goal node which has the lowest cumulative cost.

Completeness: Uniform-cost search is complete, such as if there is a solution, UCS will find it.

Time Complexity: Let C* **is Cost of the optimal solution**, and **ε** is each step to get closer to the goal node. Then the number of steps is = $C^*/\varepsilon+1$. Here we have taken +1, as we start from state 0 and end to C^*/ε.
Hence, the worst-case time complexity of Uniform-cost search is$O(b^{1 + [C^*/\varepsilon]})/$.

Space Complexity: The same logic is for space complexity so, the worst-case space complexity of Uniform-cost search is $O(b^{1 + [C^*/\varepsilon]})$.

Optimal: Uniform-cost search is always optimal as it only selects a path with the lowest path cost.

5. Iterative deepeningdepth-first Search:

- The iterative deepening algorithm is a combination of DFS and BFS algorithms.

- This search algorithm finds out the best depth limit and does it by gradually increasing the limit until a goal is found.
- The iterative search algorithm is useful uninformed search when search space is large, and depth of goal node is unknown.

Completeness: This algorithm is complete is ifthe branching factor is finite.

Time Complexity: Let's suppose b is the branching factor and depth is d then the worst-case time complexity is **$O(b^d)$**.

Space Complexity: The space complexity of IDDFS will be **O(bd)**.

Optimal: IDDFS algorithm is optimal if path cost is a non- decreasing function of the depth of the node.

6. Bidirectional Search Algorithm:

- Bidirectional search algorithm runs two simultaneous searches, one form initial state called as forward-search and other from goal node called as backward-search, to find the goal node.
- Bidirectional search replaces one single search graph with two small subgraphs in which one starts the search from an initial vertex and other starts from goal vertex.
- The search stops when these two graphs intersect each other.
- Bidirectional search can use search techniques such as BFS, DFS, DLS, etc.

Completeness: Bidirectional Search is complete if we use BFS in both searches.

Time Complexity: Time complexity of bidirectional search using BFS is **$O(b^d)$**.

Space Complexity: Space complexity of bidirectional search is **$O(b^d)$**.

Optimal: Bidirectional search is Optimal.

INFORMED SEARCH

- Informed search algorithms use domain knowledge. In an informed search, problem information is available which can guide the search.
- Informed search strategies can find a solution more efficiently than an uninformed search strategy.
- Informed search is also called a Heuristic search.
- A heuristic is a way which might not always be guaranteed for best solutions but guaranteed to find a good solution in reasonable time.
- Informed search can solve much complex problem which could not be solved in another way.
- The informed search algorithm is more useful for large search space.
- Informed search algorithm uses the idea of heuristic, so it is also called Heuristic search.

Heuristics function

- Heuristic is a function which is used in Informed Search, and it finds the most promising path.
- It takes the current state of the agent as its input and produces the estimation of how close agent is from the goal.
- The heuristic method, however, might not always give the best solution, but it guaranteed to find a good solution in reasonable time.
- Heuristic function estimates how close a state is to the goal.
- It is represented by h(n), and it calculates the cost of an optimal path between the pair of states.
- The value of the heuristic function is always positive.
- **h(n) <= h*(n)** Here h(n) is heuristic cost, and h*(n) is the estimated cost. Hence heuristic cost should be less than or equal to the estimated cost.

An example of informed search algorithms is a traveling salesman problem.

PURE HEURISTIC SEARCH:

- Pure heuristic search is the simplest form of heuristic search algorithms.
- It expands nodes based on their heuristic value h(n).
- It maintains two lists, OPEN and CLOSED list. In the CLOSED list, it places those nodes which have already expanded and in the OPEN list, it places nodes which have yet not been expanded.
- On each iteration, each node n with the lowest heuristic value is expanded and generates all its successors and n is placed to the closed list.
- The algorithm continues unit a goal state is found.
- In the informed search we will discuss two main algorithms which are given below:
  - **Best First Search Algorithm(Greedy search)**
  - **A* Search Algorithm**

1.) Best-first Search Algorithm (Greedy Search):

- Greedy best-first search algorithm always selects the path which appears best at that moment.
- It is the combination of depth-first search and breadth-first search algorithms.
- It uses the heuristic function and search.
- Best-first search allows us to take the advantages of both algorithms.
- With the help of best-first search, at each step, we can choose the most promising node.
- In the best first search algorithm, we expand the node which is closest to the goal node and the closest cost is estimated by heuristic function, i.e.
- **f(n)= g(n).** Were, h(n)= estimated cost from node n to the goal.
- The greedy best first algorithm is implemented by the priority queue.

Advantages:

- Best first search can switch between BFS and DFS by gaining the advantages of both the algorithms.
- This algorithm is more efficient than BFS and DFS algorithms.

Disadvantages:

- It can behave as an unguided depth-first search in the worst case scenario.

- It can get stuck in a loop as DFS.
- This algorithm is not optimal.

Time Complexity: The worst case time complexity of Greedy best first search is O(b^m).

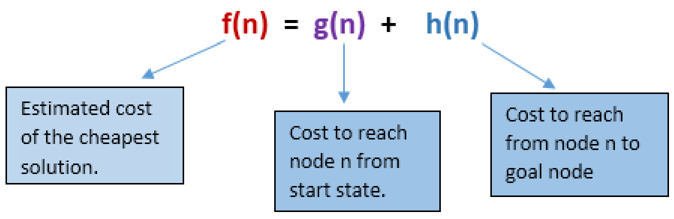
Space Complexity: The worst case space complexity of Greedy best first search is O(b^m). Where, m is the maximum depth of the search space.

Complete: Greedy best-first search is also incomplete, even if the given state space is finite.

Optimal: Greedy best first search algorithm is not optimal.

2.) A* Search Algorithm:

- A* search is the most commonly known form of best-first search.
- It uses heuristic function h(n), and cost to reach the node n from the start state g(n).
- It has combined features of UCS and greedy best-first search, by which it solve the problem efficiently.
- A* search algorithm finds the shortest path through the search space using the heuristic function.
- This search algorithm expands less search tree and provides optimal result faster.
- A* algorithm is similar to UCS except that it uses g(n)+h(n) instead of g(n).
- In A* search algorithm, we use search heuristic as well as the cost to reach the node. Hence we can combine both costs as following, and this sum is called as a **fitness number**.


f(n) = g(n) + h(n)
Estimated cost of the cheapest solution.
Cost to reach node n from start state.
Cost to reach from node n to goal node

Advantages:

- A* search algorithm is the best algorithm than other search algorithms.
- A* search algorithm is optimal and complete.
- This algorithm can solve very complex problems.

Disadvantages:

- It does not always produce the shortest path as it mostly based on heuristics and approximation.
- A* search algorithm has some complexity issues.
- The main drawback of A* is memory requirement as it keeps all generated nodes in the memory, so it is not practical for various large-scale problems.

Complete: A* algorithm is complete as long as:

  - Branching factor is finite.
  - Cost at every action is fixed.

Optimal: A* search algorithm is optimal if it follows below two conditions:

- **Admissible:** the first condition requires for optimality is that h(n) should be an admissible heuristic for A* tree search. An admissible heuristic is optimistic in nature.
- **Consistency:** Second required condition is consistency for only A* graph-search.

  If the heuristic function is admissible, then A* tree search will always find the least cost path.

Time Complexity: The time complexity of A* search algorithm depends on heuristic function, and the number of nodes expanded is exponential to the depth of solution d. So the time complexity is O(b^d), where b is the branching factor.

Space Complexity: The space complexity of A* search algorithm is **O(b^d)**

HILL CLIMBING ALGORITHM IN ARTIFICIAL INTELLIGENCE

- Hill climbing algorithm is a local search algorithm which continuously moves in the direction of increasing elevation/value to find the peak of the mountain or best solution to the problem. It terminates when it reaches a peak value where no neighbor has a higher value.
- Hill climbing algorithm is a technique which is used for optimizing the mathematical problems. One of the widely discussed examples of Hill climbing algorithm is Traveling-salesman Problem in which we need to minimize the distance traveled by the salesman.
- It is also called greedy local search as it only looks to its good immediate neighbor state and not beyond that.
- A node of hill climbing algorithm has two components which are state and value.
- Hill Climbing is mostly used when a good heuristic is available.
- In this algorithm, we don't need to maintain and handle the search tree or graph as it only keeps a single current state.

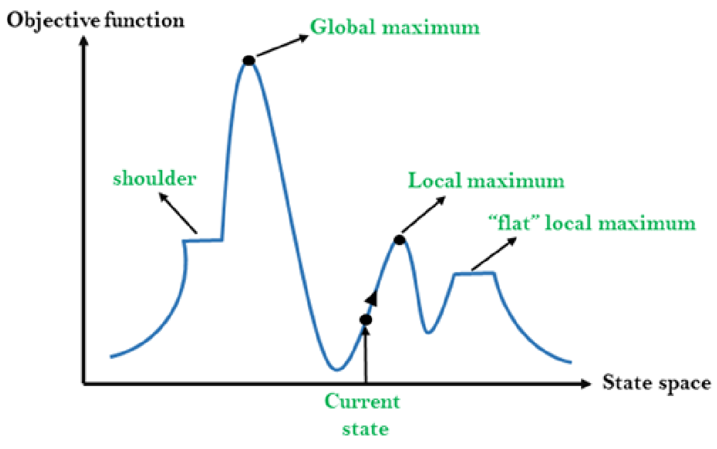
Features of Hill Climbing:

Following are some main features of Hill Climbing Algorithm:

1. **Generate and Test variant:** Hill Climbing is the variant of Generate and Test method. The Generate and Test method produce feedback which helps to decide which direction to move in the search space.
2. **Greedy approach:** Hill-climbing algorithm search moves in the direction which optimizes the cost.
3. **No backtracking:** It does not backtrack the search space, as it does not remember the previous states.

State-space Diagram for Hill Climbing:

- The state-space landscape is a graphical representation of the hill-climbing algorithm which is showing a graph between various states of algorithm and Objective function/Cost.
- On Y-axis we have taken the function which can be an objective function or cost function, and state-space on the x-axis.
- If the function on Y-axis is cost then, the goal of search is to find the global minimum and local minimum.
- If the function of Y-axis is Objective function, then the goal of the search is to find the global maximum and local maximum.



<u>Different regions in the state space landscape:</u>

Local Maximum: Local maximum is a state which is better than its neighbor states, but there is also another state which is higher than it.

Global Maximum: Global maximum is the best possible state of state space landscape. It has the highest value of objective function.

Current state: It is a state in a landscape diagram where an agent is currently present.

Flat local maximum: It is a flat space in the landscape where all the neighbor states of current states have the same value.

Shoulder: It is a plateau region which has an uphill edge.

<u>TYPES OF HILL CLIMBING ALGORITHM:</u>

1. **Simple hill Climbing:**
2. **Steepest-Ascent hill-climbing:**
3. **Stochastic hill Climbing:**

1. Simple Hill Climbing:

- Simple hill climbing is the simplest way to implement a hill climbing algorithm.
- It only evaluates the neighbor node state at a time and selects the first one which optimizes current cost and set it as a current state**.**
- It only checks it's one successor state, and if it finds better than the current state, then move else be in the same state.

This algorithm has the following features:

- Less time consuming
- Less optimal solution and the solution is not guaranteed

Algorithm for Simple Hill Climbing:

- **Step 1:** Evaluate the initial state, if it is goal state then return success and Stop.
- **Step 2:** Loop Until a solution is found or there is no new operator left to apply.
- **Step 3:** Select and apply an operator to the current state.

- **Step 4:** Check new state:
    a. If it is goal state, then return success and quit.
    b. Else if it is better than the current state then assign new state as a current state.
    c. Else if not better than the current state, then return to step2.
- **Step 5:** Exit.

2. Steepest-Ascent hill climbing:

- The steepest-Ascent algorithm is a variation of simple hill climbing algorithm.
- This algorithm examines all the neighboring nodes of the current state and selects one neighbor node which is closest to the goal state.
- This algorithm consumes more time as it searches for multiple neighbors

Algorithm for Steepest-Ascent hill climbing:

- **Step 1:** Evaluate the initial state, if it is goal state then return success and stop, else make current state as initial state.
- **Step 2:** Loop until a solution is found or the current state does not change.
    a. Let SUCC be a state such that any successor of the current state will be better than it.
    b. For each operator that applies to the current state:
        a. Apply the new operator and generate a new state.
        b. Evaluate the new state.
        c. If it is goal state, then return it and quit, else compare it to the SUCC.
        d. If it is better than SUCC, then set new state as SUCC.
        e. If the SUCC is better than the current state, then set current state to SUCC.
- **Step 5:** Exit.

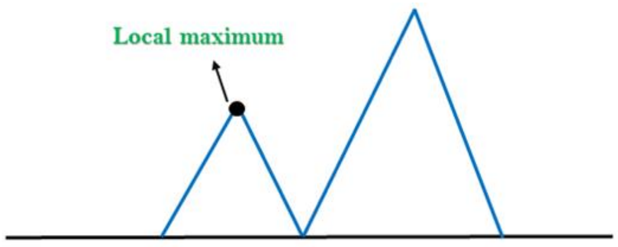
3. Stochastic hill climbing:

- Stochastic hill climbing does not examine for all its neighbor before moving.
- Rather, this search algorithm selects one neighbor node at random and decides whether to choose it as a current state or examine another state.

PROBLEMS IN HILL CLIMBING ALGORITHM:

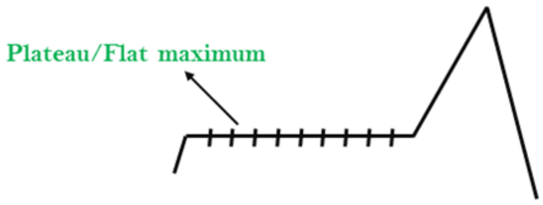
1. Local Maximum: A local maximum is a peak state in the landscape which is better than each of its neighboring states, but there is another state also present which is higher than the local maximum.

Solution: Backtracking technique can be a solution of the local maximum in state space landscape. Create a list of the promising path so that the algorithm can backtrack the search space and explore other paths as well.



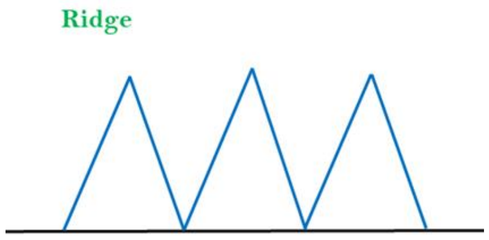
2. Plateau: A plateau is the flat area of the search space in which all the neighbor states of the current state contains the same value, because of this algorithm does not find any best direction to move. A hill-climbing search might be lost in the plateau area.

Solution: The solution for the plateau is to take big steps or very little steps while searching, to solve the problem. Randomly select a state which is far away from the current state so it is possible that the algorithm could find non-plateau region.



3. Ridges: A ridge is a special form of the local maximum. It has an area which is higher than its surrounding areas, but itself has a slope, and cannot be reached in a single move.

Solution: With the use of bidirectional search, or by moving in different directions, we can improve this problem.



Simulated Annealing:

- A hill-climbing algorithm which never makes a move towards a lower value guaranteed to be incomplete because it can get stuck on a local maximum. And if algorithm applies a random walk, by moving a successor, then it may complete but not efficient.
- **Simulated Annealing** is an algorithm which yields both efficiency and completeness.
- In mechanical term **Annealing** is a process of hardening a metal or glass to a high temperature then cooling gradually, so this allows the metal to reach a low-energy crystalline state.
- The same process is used in simulated annealing in which the algorithm picks a random move, instead of picking the best move.
- If the random move improves the state, then it follows the same path. Otherwise, the algorithm follows the path which has a probability of less than 1 or it moves downhill and chooses another path.

PROBLEM REDUCTION IN AI

- We already know about the divide and conquer strategy, a solution to a problem can be obtained by decomposing it into smaller sub-problems.
- Each of this sub-problem can then be solved to get its sub solution.
- These sub solutions can then be recombined to get a solution as a whole. That is called is **Problem Reduction**.
- This method generates arc which is called as **AND** arcs. One AND arc may point to any number of successor nodes, all of which must be solved for an arc to point to a solution.

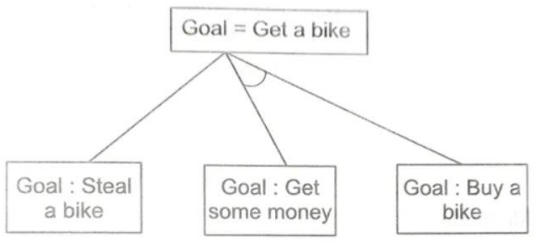


Fig: AND / OR Graph

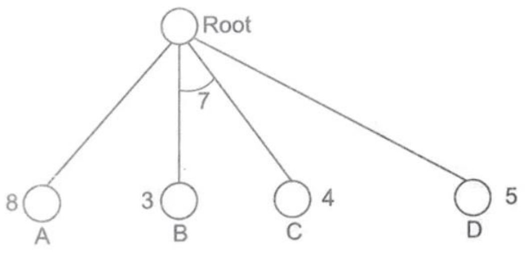


Fig: AND / OR Tree

CONSTRAINT SATISFACTION PROBLEMS (CSP) IN AI

- Constraint satisfaction means solving a problem under certain constraints or rules.
- Constraint satisfaction is a technique where a problem is solved when its values satisfy certain constraints or rules of the problem. Such type of technique leads to a deeper understanding of the problem structure as well as its complexity.
- Constraint satisfaction depends on three components, namely:
  - **X:** It is a set of variables.
  - **D:** It is a set of domains where the variables reside. There is a specific domain for each variable.
  - **C:** It is a set of constraints which are followed by the set of variables.

Solving Constraint Satisfaction Problems

The requirements to solve a constraint satisfaction problem (CSP) is :

- A state-space
- The notion of the solution.

A state in state-space is defined by assigning values to some or all variables such as **$\{X_1=v_1, X_2=v_2,$ and so on…$\}$.**

Types of Domains in CSP

- **Discrete Domain:** It is an infinite domain which can have one state for multiple variables.
  For example, a start state can be allocated infinite times for each variable.
- **Finite Domain:** It is a finite domain which can have continuous states describing one domain for one specific variable. It is also called a continuous domain.

Constraint Types in CSP

- **Unary Constraints:** It is the simplest type of constraints that restricts the value of a single variable.
- **Binary Constraints:** It is the constraint type which relates two variables. A value **x_2** will contain a value which lies between **x1** and **x3**.

- **Global Constraints:** It is the constraint type which involves an arbitrary number of variables.

Some special types of solution algorithms are used to solve the following types of constraint

- **Linear Constraints:** These type of constraints are commonly used in linear programming where each variable containing an integer value exists in linear form only.
- **Non-linear Constraints:** These type of constraints are used in non-linear programming where each variable (an integer value) exists in a non-linear form.

Note: *A special constraint which works in real-world is known as* ***Preference constraint.***

CSP Problems

- **Graph Coloring:** The problem where the constraint is that no adjacent sides can have the same color.
- **Sudoku Playing:** The game play where the constraint is that no number from 0-9 can be repeated in the same row or column.
- **Latin square Problem:** In this game, the task is to search the pattern which is occurring several times in the game. They may be shuffled but will contain the same digits.

BEYOND CLASSICAL SEARCH

LOCAL SEARCH ALGORITHMS AND OPTIMIZATION PROBLEMS

- The search algorithms that we have seen so far are designed to explore search spaces systematically.
- This systematicity is achieved by keeping one or more paths in memory and by recording which alternatives have been explored at each point along the path.
- When a goal is found, the path to that goal also constitutes a solution to the problem. In many problems, how ever, the path to the goal is irrelevant.
- If the path to the goal does not matter, we might consider a different class of algorithms, ones that do not worry about paths at all.
- Local search algorithms operate using a single current node (rather than multiple paths) and generally move only to neighbours of that node.
- Typically, the paths followed by the search are not retained.
- Although local search algorithms are not systematic, they have two key advantages:
  - they use very little memory—usually a constant amount
  - they can often find reasonable solutions in large or infinite (continuous) state spaces for which systematic algorithms are unsuitable

- Hill climbing is sometimes called greedy local search because it grabs a good neighbour state without thinking ahead about where to go next.
- Although greed is considered one of the seven deadly sins, it turns out that greedy algorithms often perform quite well.
- Hill climbing often makes rapid progress toward a solution because it is usually quite easy to improve a bad state

LOCAL SEARCH IN CONTINUOUS SPACES

SEARCHING WITH NONDETERMINISTIC ACTIONS

- When the environment is either partially observable or nondeterministic (or both), percepts become useful.
- In a partially observable environment, every percept helps narrow down the set of possible states the agent might be in, thus making it easier for the agent to achieve its goals.
- When the environment is nondeterministic, percepts tell the agent which of the possible outcomes of its actions has actually occurred.
- In both cases, the future percepts cannot be determined in advance and the agent's future actions will depend on those future percepts.
- So the solution to a problem is not a sequence but a contingency plan (also known as a strategy) that specifies what to do depending on what percepts are received.
    - The erratic vacuum world
    - AND–OR search trees
    - Try, try again

SEARCHING WITH PARTIAL OBSERVATIONS

- The agent's percepts do not suffice to pin down the exact state.
- As noted previous if the agent is in one of several possible states, then an action may lead to one of several possible outcomes—even if the environment is deterministic.
- The key concept required for solving partially observable problems is the belief state, representing the agent's current belief about the possible physical states it might be in, given the sequence of actions and percepts up to that point.
- When the agent has no sensors at all then we add in partial sensing as well as nondeterministic actions.

- **Searching with no observation**

  We can define the corresponding sensorless problem as follows:
    - Belief states:
    - Initial state
    - Actions
    - Transition model
    - Goal test
    - Path cost
    -

- **Searching with observations**

  We can think of transitions from one belief state to the next for a particular action as occurring in three stages
    - The prediction stage
    - The observation prediction stage
    - The update stage

- **Solving partially observable problems**

- **An agent for partially observable environments**

  In partially observable environments—which include the vast majority of real-world environments—maintaining one's belief state is a core function of any intelligent system. This function goes under various names, including
    - monitoring,
    - Filtering

- State estimation
- Recursive state estimator

ONLINE SEARCH AGENTS AND UNKNOWN ENVIRONMENTS

- In contrast, an online search agent interleaves computation and action: first it takes an action, then it observes the environment and computes the next action.
- Online search is a good idea in dynamic or semidynamic domains—domains where there is a penalty for sitting around and computing too long.
- Online search is also helpful in nondeterministic domains because it allows the agent to focus its computational efforts on the contingencies that actually arise rather than those that might happen but probably won't.
- Of course, there is a tradeoff: the more an agent plans ahead, the less often it will find itself up the creek without a paddle.
- Online search is a necessary idea for unknown environments, where the agent does not know what states exist or what its actions do. In this state of ignorance, the agent faces an exploration problem and must use its actions as experiments in order to learn enough to make deliberation worthwhile.
- The canonical example of online search is a robot that is placed in a new building and must explore it to build a map that it can use for getting from A to B.
- Methods for escaping from labyrinths—required knowledge for aspiring heroes of antiquity—are also examples of online search algorithms.
- Spatial exploration is not the only form of exploration, however. Consider a newborn baby: it has many possible actions but knows the outcomes of none of them, and it has experienced only a few of the possible states that it can reach.
- The baby's gradual discovery of how the world works is, in part, an online search process.

Online search problems

- ACTIONS(s), which returns a list of actions allowed in state s;
- The step-cost function c(s, a, s')—note that this cannot be used until the agent knows that s' is the outcome.
- GOAL-TEST(s).