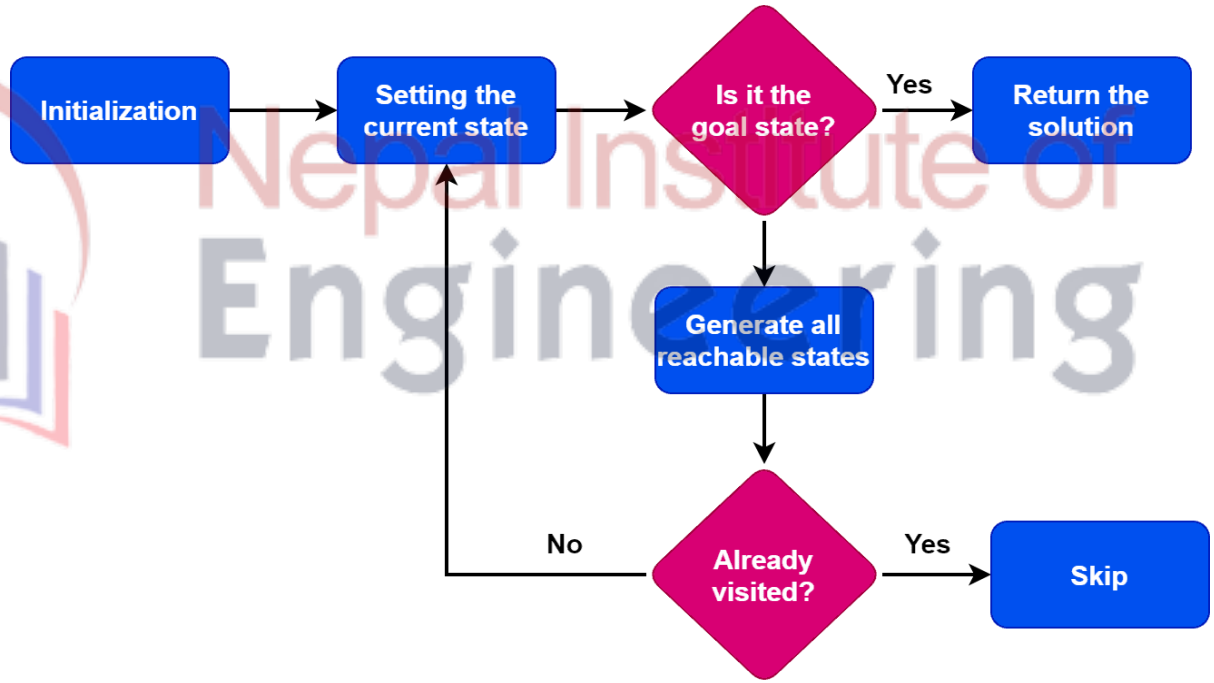
Problem solving and searching techniques

Problem as a state space search

- State space search is a method used widely in artificial intelligence and computer science to find a solution to a problem by searching through the set of possible states of the problem.
- Algorithm is



Problem Formulation

- **There are basically three types of problem in artificial intelligence:**
  - **Ignorable:** In which solution steps can be ignored.
  - **Recoverable:** In which solution steps can be undone.
  - **Irrecoverable:** Solution steps cannot be undo.
- **Steps problem-solving in AI:** The problem of AI is directly associated with the nature of humans and their activities. So we need a number of finite steps to solve a problem which makes human easy works.
- These are the following steps which require to solve a problem :
- **Problem definition:** Detailed specification of inputs and acceptable system solutions.
- **Problem analysis:** Analyse the problem thoroughly.

- **Knowledge Representation:** collect detailed information about the problem and define all possible techniques.
- **Problem-solving:** Selection of best techniques.

Components to formulate the associated problem:

- **Initial State:** This state requires an initial state for the problem which starts the AI agent towards a specified goal. In this state new methods also initialize problem domain solving by a specific class.
- **Action:** This stage of problem formulation works with function with a specific class taken from the initial state and all possible actions done in this stage.
- **Transition:** This stage of problem formulation integrates the actual action done by the previous action stage and collects the final stage to forward it to their next stage.

- **Goal test:** This stage determines that the specified goal achieved by the integrated transition model or not, whenever the goal achieves stop the action and forward into the next stage to determines the cost to achieve the goal.
- **Path costing:** This component of problem-solving numerical assigned what will be the cost to achieve the goal. It requires all hardware software and human working cost.

Constraint Satisfaction Problems (CSP)

- Finding a solution that meets a set of constraints is the goal of constraint satisfaction problems (CSPs), a type of AI issue.
- **There are mainly three basic components in the constraint satisfaction problem:**
- **Variables**
  - The things that need to be determined are variables.

- Variables in a CSP are the objects that must have values assigned to them in order to satisfy a particular set of constraints.
- Boolean, integer, and categorical variables are just a few examples of the various types of variables.

- **Domains**
  - The range of potential values that a variable can have is represented by domains.
  - Depending on the issue, a domain may be finite or limitless.
  - For instance, in Sudoku, the set of numbers from 1 to 9 can serve as the domain of a variable representing a problem cell.
- **Constraints**
  - The guidelines that control how variables relate to one another are known as constraints.

- Constraints in a CSP define the ranges of possible values for variables.
- Unary constraints, binary constraints, and higher-order constraints are only a few examples of the various sorts of constraints.
- For instance, in a sudoku problem, the restrictions might be that each row, column, and 3×3 box can only have one instance of each number from 1 to 9.

Constraint Satisfaction Problems (CSP) algorithms:

- The **backtracking algorithm**
  - is a depth-first search algorithm that methodically investigates the search space of potential solutions up until a solution is discovered that satisfies all the restrictions.
  - The method begins by choosing a variable and giving it a value before repeatedly attempting to give values to the other variables.
  - The method returns to the prior variable and tries a different value if at any time a variable cannot be given a value that fulfills the requirements.
  - Once all assignments have been tried or a solution that satisfies all constraints has been discovered, the algorithm ends.

- The **forward-checking algorithm**
  - is a variation of the backtracking algorithm that condenses the search space using a type of local consistency.
  - For each unassigned variable, the method keeps a list of remaining values and applies local constraints to eliminate inconsistent values from these sets.
  - The algorithm examines a variable's neighbors after it is given a value to see whether any of its remaining values become inconsistent and removes them from the sets if they do.
  - The algorithm goes backward if, after forward checking, a variable has no more values.

Constraint Satisfaction Problems (CSP) representation

- The finite set of variables V_1, V_2, V_3 ……..V_n.
- Non-empty domain for every single variable D_1, D_2, D_3 ……..D_n.
- The finite set of constraints C_1, C_2 ……, Cm.
  - where each constraint C_i restricts the possible values for variables,
    - e.g., $V_1 \neq V_2$
  - Each constraint C_i is a pair <scope, relation>
    - Example: <(V_1, V_2), V_1 not equal to V_2>
  - Scope = set of variables that participate in constraint.
  - Relation = list of valid variable value combinations.
    - There might be a clear list of permitted combinations. Perhaps a relation that is abstract and that allows for membership testing and listing.

Uninformed Search Algorithms

- Uninformed search is a class of general-purpose search algorithms which operates in brute force-way.
- Uninformed search algorithms do not have additional information about state or search space other than how to traverse the tree, so it is also called blind search.
- Following are the various types of uninformed search algorithms:
  - Breadth-first Search
  - Depth-first Search
  - Depth-limited Search
  - Iterative deepening depth-first search
  - Bidirectional Search

Breadth-first Search

- Breadth-first search is the most common search strategy for traversing a tree or graph. This algorithm searches breadthwise in a tree or graph, so it is called 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.
- **$T(b) = 1+b^2+b^3+.....+ b^d= O(b^d)$**

- **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.

Depth-first Search

- Depth-first search is a 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.
- The process of the DFS algorithm is similar to the BFS algorithm.

- **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:
- **$T(n)= 1+ n^2+ n^3 +.....+ n^m=O(n^m)$**
- **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.

Depth-Limited Search Algorithm

- A depth-limited search algorithm is similar to depth-first search with a predetermined limit.
- 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.
- Depth-limited search can be terminated with two Conditions of failure:
- Standard failure value: It indicates that problem does not have any solution.
- Cutoff failure value: It defines no solution for the problem within a given depth limit.

- **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 $\ell > d$.

Iterative deepening depth-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.
- This algorithm performs depth-first search up to a certain "depth limit", and it keeps increasing the depth limit after each iteration until the goal node is found.
- This Search algorithm combines the benefits of Breadth-first search's fast search and depth-first search's memory efficiency.

- 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.

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 sub graphs 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 Algorithm

- 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.

- Admissibility of the heuristic function is given as:
  - 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.

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.
- Best first search algorithm:
  - **Step 1:** Place the starting node into the OPEN list.
  - **Step 2:** If the OPEN list is empty, Stop and return failure.
  - **Step 3:** Remove the node n, from the OPEN list which has the lowest value of h(n), and places it in the CLOSED list.
  - **Step 4:** Expand the node n, and generate the successors of node n.

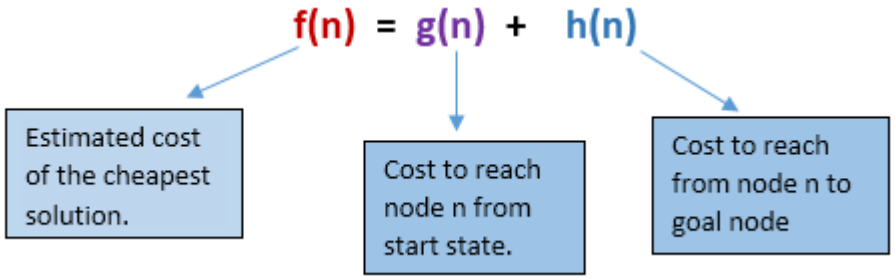
- **Step 5:** Check each successor of node n, and find whether any node is a goal node or not. If any successor node is goal node, then return success and terminate the search, else proceed to Step 6.
- **Step 6:** For each successor node, algorithm checks for evaluation function f(n), and then check if the node has been in either OPEN or CLOSED list. If the node has not been in both list, then add it to the OPEN list.
- **Step 7:** Return to Step 2.

- **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.

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**.

- Hence we can combine both costs as following, and this sum is called as a **fitness number**.



Algorithm of A* search

- **Step1:** Place the starting node in the OPEN list.
- **Step 2:** Check if the OPEN list is empty or not, if the list is empty then return failure and stops.
- **Step 3:** Select the node from the OPEN list which has the smallest value of evaluation function (g+h), if node n is goal node then return success and stop, otherwise
- **Step 4:** Expand node n and generate all of its successors, and put n into the closed list. For each successor n', check whether n' is already in the OPEN or CLOSED list, if not then compute evaluation function for n' and place into Open list.
- **Step 5:** Else if node n' is already in OPEN and CLOSED, then it should be attached to the back pointer which reflects the lowest g(n') value.
- **Step 6:** Return to **Step 2**.

- **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)**

Game Playing

- Game playing is a popular application of artificial intelligence that involves the development of computer programs to play games, such as chess, checkers, or Go.
- The goal of game playing in artificial intelligence is to develop algorithms that can learn how to play games and make decisions that will lead to winning outcomes.
- One of the earliest examples of successful game playing AI is the chess program Deep Blue, developed by IBM, which defeated the world champion Garry Kasparov in 1997.
- Since then, AI has been applied to a wide range of games, including two-player games, multiplayer games, and video games.
- There are two main approaches to game playing in AI, rule-based systems and machine learning-based systems.
  - **Rule-based systems** use a set of fixed rules to play the game.
  - **Machine learning-based systems** use algorithms to learn from experience and make decisions based on that experience.

- The most common search technique in game playing is **Minimax search procedure**. It is depth-first depth-limited search procedure. It is used for games like chess and tic-tac-toe.

Minimax algorithm uses two functions

- **MOVEGEN :** It generates all the possible moves that can be generated from the current position.
- **STATICEVALUATION :** It returns a value depending upon the goodness from the viewpoint of two-player
- This algorithm is a two player game, so we call the first player as PLAYER1 and second player as PLAYER2. The value of each node is backed-up from its children. For PLAYER1 the backed-up value is the maximum value of its children and for PLAYER2 the backed-up value is the minimum value of its children. It provides most promising move to PLAYER1, assuming that the PLAYER2 has make the best move.
- It is a recursive algorithm, as same procedure occurs at each level.

Adversarial Search

- Adversarial search is a search, where we examine the problem which arises when we try to plan ahead of the world and other agents are planning against us.
- **Searches in which two or more players with conflicting goals are trying to explore the same search space for the solution, are called adversarial searches, often known as Games**.
- Games are modeled as a Search problem and heuristic evaluation function, and these are the two main factors which help to model and solve games in AI.

Types of Games in AI

- **Imperfect information:** If in a game agents do not have all information about the game and not aware with what's going on, such type of games are called the game with imperfect information, such as tic-tac-toe, Battleship, blind, Bridge, etc.
- **Deterministic games:** Deterministic games are those games which follow a strict pattern and set of rules for the games, and there is no randomness associated with them. Examples are chess, Checkers, Go, tic-tac-toe, etc.
- **Non-deterministic games:** Non-deterministic are those games which have various unpredictable events and has a factor of chance or luck. This factor of chance or luck is introduced by either dice or cards. These are random, and each action response is not fixed. Such games are also called as stochastic games.
  Example: Backgammon, Monopoly, Poker, etc.

Minimax Search

- Mini-max algorithm 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.
- Mini-Max algorithm uses recursion to search through the game-tree.
- Min-Max algorithm is mostly used for game playing in AI. Such as Chess, Checkers, tic-tac-toe, go, and various two-players game. This Algorithm computes the minimax decision for the current state.
- In this algorithm two players play the game, one is called MAX and other is called MIN.
- Both the players fight it as the opponent player gets the minimum benefit while they get the maximum benefit.
- Both Players of the game are opponent of each other, where MAX will select the maximized value and MIN will select the minimized value.
- **The minimax algorithm performs a depth-first search algorithm for the exploration of the complete game tree.**
- The minimax algorithm proceeds all the way down to the terminal node of the tree, then backtrack the tree as the recursion.

Alpha Beta Pruning

- Alpha-beta pruning is a modified version of the minimax algorithm. It is an optimization technique for the minimax algorithm.
- As we have seen in the minimax search algorithm that the number of game states it has to examine are exponential in depth of the tree. Since we cannot eliminate the exponent, but we can cut it to half. Hence there is a technique by which without checking each node of the game tree we can compute the correct minimax decision, and this technique is called **pruning**. This involves two threshold parameter Alpha and beta for future expansion, so it is called **alpha-beta pruning**. It is also called as **Alpha-Beta Algorithm**.
- Alpha-beta pruning can be applied at any depth of a tree, and sometimes it not only prune the tree leaves but also entire sub-tree.

- The two-parameter can be defined as:
  - **Alpha:** The best (highest-value) choice we have found so far at any point along the path of Maximizer. The initial value of alpha is **-∞**.
  - **Beta:** The best (lowest-value) choice we have found so far at any point along the path of Minimizer. The initial value of beta is +∞.
- The Alpha-beta pruning to a standard minimax algorithm returns the same move as the standard algorithm does, but it removes all the nodes which are not really affecting the final decision but making algorithm slow. Hence by pruning these nodes, it makes the algorithm fast.