Artificial Intelligence

Introduction to Artificial Intelligence

- Artificial: “created by people” (Oxford dictionary)
- Intelligence: “ability to learn, understand and think” (Oxford dictionary)

Definition:

It is a branch of computer science by which we can create intelligence machine which can **behave like human**, **think like human** and **able to make decision**.

or

It is a branch of *Science which deals with helping machines* find solutions to complex problems in a more human-like fashion and how to make computers do things which, at the moment, people do better.

Why Artificial Intelligence?

- With the help of AI, you can create such software or devices which can solve real-world problems very easily and with accuracy such as health issues, marketing, traffic issues, etc.
- With the help of AI, you can create your personal virtual Assistant, such as Cortana, Google Assistant, Siri, etc.
- With the help of AI, you can build such Robots which can work in an environment where survival of humans can be at risk.
- AI opens a path for other new technologies, new devices, and new Opportunities.

Goals of Artificial Intelligence

- Replicate human intelligence
- Solve Knowledge-intensive tasks
- An intelligent connection of perception and action
- Building a machine which can perform tasks that requires human intelligence such as:
  - ✓ Proving a theorem
  - ✓ Playing chess
  - ✓ Plan some surgical operation
  - ✓ Driving a car in traffic
- Creating some system which can exhibit intelligent behavior, learn new things by itself, demonstrate, explain, and can advise to its user.

What Comprises to Artificial Intelligence?

- Intelligence is an abstract part of our brain which is a combination of **Reasoning, learning, problem-solving, perception, language understanding, etc**.
- To achieve the above factors for a machine or software Artificial Intelligence requires the following discipline:
  - ✓ Mathematics
  - ✓ Biology
  - ✓ Psychology
  - ✓ Sociology
  - ✓ Computer Science

- ✓ Neurons Study
- ✓ Statistics

Advantages of Artificial Intelligence

High Accuracy with less errors: AI machines or systems are prone to less errors and high accuracy as it takes decisions as per pre-experience or information.

High-Speed: AI systems can be of very high-speed and fast-decision making, because of that AI systems can beat a chess champion in the Chess game.

High reliability: AI machines are highly reliable and can perform the same action multiple times with high accuracy.

Useful for risky areas: AI machines can be helpful in situations such as defusing a bomb, exploring the ocean floor, where to employ a human can be risky.

Digital Assistant: AI can be very useful to provide digital assistant to the users such as AI technology is currently used by various E-commerce websites to show the products as per customer requirement.

Useful as a public utility: AI can be very useful for public utilities such as a self-driving car which can make our journey safer and hassle-free, facial recognition for security purpose, Natural language processing to communicate with the human in human-language, etc.

Disadvantages of Artificial Intelligence

High Cost: The hardware and software requirement of AI is very costly as it requires lots of maintenance to meet current world requirements.

Can't think out of the box: Even we are making smarter machines with AI, but still they cannot work out of the box, as the robot will only do that work for which they are trained, or programmed.

No feelings and emotions: AI machines can be an outstanding performer, but still it does not have the feeling so it cannot make any kind of emotional attachment with human, and may sometime be harmful for users if the proper care is not taken.

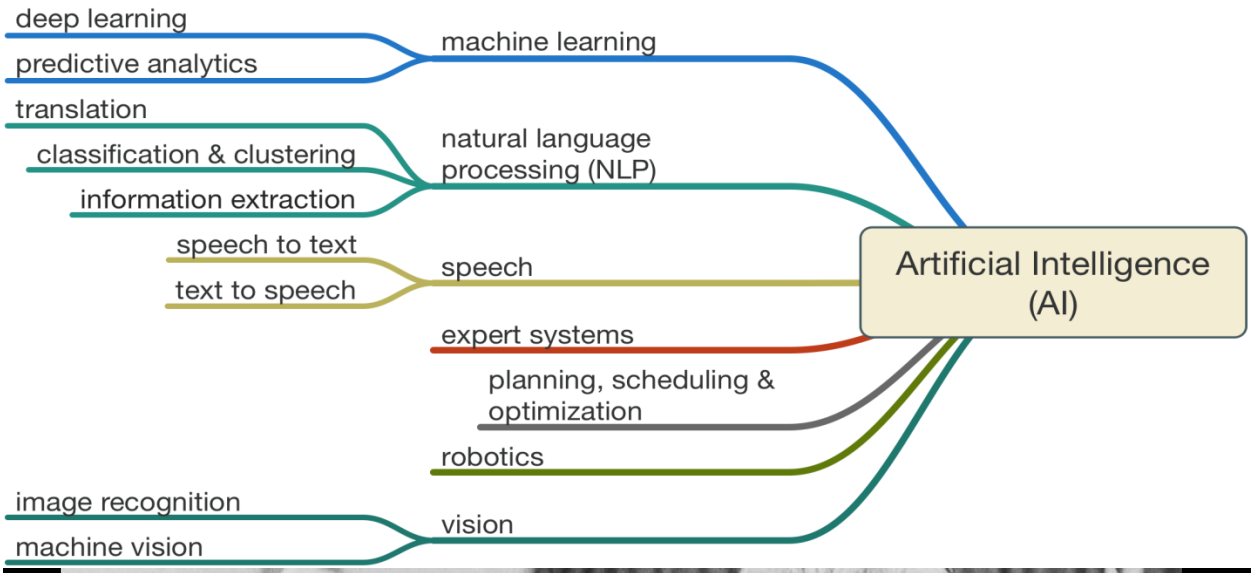
Increase dependency on machines: With the increment of technology, people are getting more dependent on devices and hence they are losing their mental capabilities.

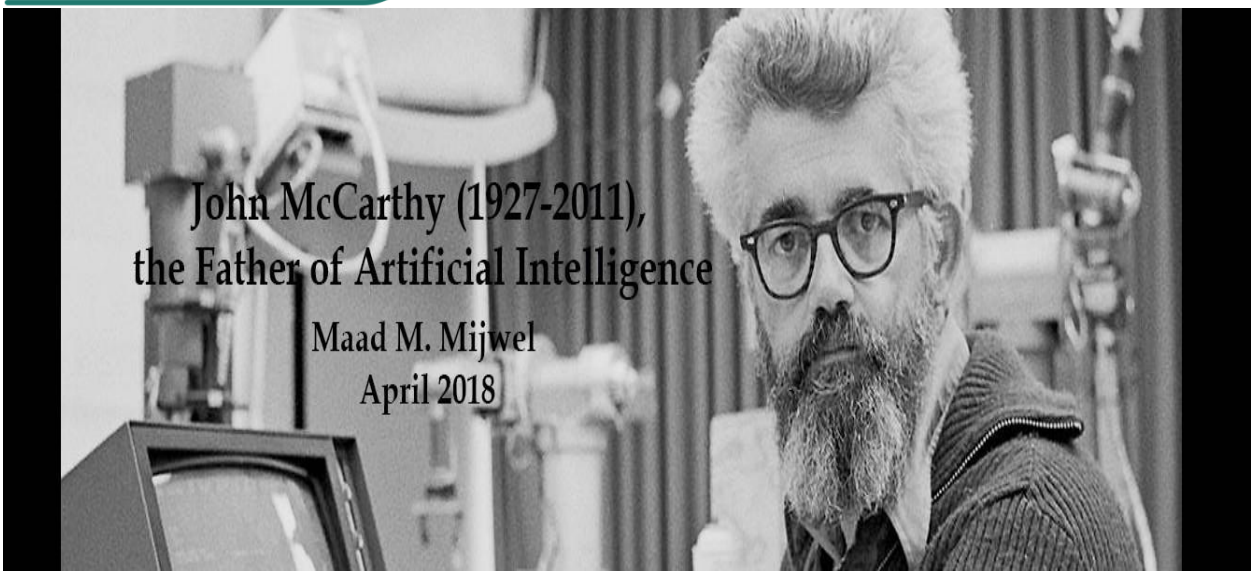
No Original Creativity: As humans are so creative and can imagine some new ideas but still AI machines cannot beat this power of human intelligence and cannot be creative and imaginative.

Applications of AI

- ➢ AI Application in E-Commerce
  - ✓ Personalized Shopping
  - ✓ AI-powered Assistants
  - ✓ Fraud Prevention
- ➢ Applications Of Artificial Intelligence in Education
  - ✓ Administrative Tasks Automated to Aid Educators
  - ✓ Creating Smart Content
  - ✓ Voice Assistants

- ✓ Personalized Learning
- Applications of Artificial Intelligence in Lifestyle
  - ✓ Autonomous Vehicles
  - ✓ Spam Filters
  - ✓ Facial Recognition
  - ✓ Recommendation System
- Applications of Artificial intelligence in Navigation
- Applications of Artificial Intelligence in Robotics
- Applications of Artificial Intelligence in Human Resource
- Applications of Artificial Intelligence in Healthcare
- Applications of Artificial Intelligence in Agriculture
- Applications of Artificial Intelligence in Gaming
- Applications of Artificial Intelligence in Automobiles
- Applications of Artificial Intelligence in Social Media
  - ✓ Instagram
  - ✓ Facebook
  - ✓ Twitter
- Applications of Artificial Intelligence in Marketing
- Applications of Artificial Intelligence in Chatbots
- Applications of Artificial Intelligence in Finance





History of Artificial Intelligence

Maturation of Artificial Intelligence (1943-1952)

Year 1943: The first work which is now recognized as AI was done by Warren McCulloch and Walter pits in 1943. They proposed a model of **artificial neurons**.

Year 1949: Donald Hebb demonstrated an updating rule for modifying the connection strength between neurons. His rule is now called **Hebbian learning**.

Year 1950: The Alan Turing who was an English mathematician and pioneered Machine learning in 1950. Alan Turing publishes **"Computing Machinery and Intelligence"** in which he proposed a test. The test can check the machine's ability to exhibit intelligent behavior equivalent to human intelligence, called a **Turing test**.

The birth of Artificial Intelligence (1952-1956):

Year 1955: An Allen Newell and Herbert A. Simon created the "first artificial intelligence program"Which was named as **"Logic Theorist"**. This program had proved 38 of 52 Mathematics theorems, and find new and more elegant proofs for some theorems.

Year 1956: The word "Artificial Intelligence" first adopted by American Computer scientist John McCarthy at the Dartmouth Conference. For the first time, AI coined as an academic field.

At that time high-level computer languages such as FORTRAN, LISP, or COBOL were invented. And the enthusiasm for AI was very high at that time.

The golden years-Early enthusiasm (1956-1974)

Year 1966: The researchers emphasized developing algorithms which can solve mathematical problems. Joseph Weizenbaum created the first chatbot in 1966, which was named as ELIZA.

Year 1972: The first intelligent humanoid robot was built in Japan which was named as WABOT-1.

The first AI winter (1974-1980)

The duration between years 1974 to 1980 was the first AI winter duration. AI winter refers to the time period where computer scientist dealt with a severe shortage of funding from government for AI researches.

During AI winters, an interest of publicity on artificial intelligence was decreased.

A boom of AI (1980-1987)

Year 1980: After AI winter duration, AI came back with "Expert System". Expert systems were programmed that emulate the decision-making ability of a human expert.

In the Year 1980, the first national conference of the American Association of Artificial Intelligence was held at Stanford University.

The second AI winter (1987-1993)

The duration between the years 1987 to 1993 was the second AI Winter duration.

Again Investors and government stopped in funding for AI research as due to high cost but not efficient result. The expert system such as XCON was very cost effective.

The emergence of intelligent agents (1993-2011)

Year 1997: In the year 1997, IBM Deep Blue beats world chess champion, Gary Kasparov, and became the first computer to beat a world chess champion.

Year 2002: for the first time, AI entered the home in the form of Roomba, a vacuum cleaner.

Year 2006: AI came in the Business world till the year 2006. Companies like Facebook, Twitter, and Netflix also started using AI.

Deep learning, big data and artificial general intelligence (2011-present)

Year 2011: In the year 2011, IBM's Watson won jeopardy, a quiz show, where it had to solve the complex questions as well as riddles. Watson had proved that it could understand natural language and can solve tricky questions quickly.

Year 2012: Google has launched an Android app feature "Google now", which was able to provide information to the user as a prediction.

Year 2014: In the year 2014, Chatbot "Eugene Goostman" won a competition in the infamous "Turing test."

Year 2018: The "Project Debater" from IBM debated on complex topics with two master debaters and also performed extremely well.

Google has demonstrated an AI program "Duplex" which was a virtual assistant and which had taken hairdresser appointment on call, and lady on other side didn't notice that she was talking with the machine.

Artificial Intelligence Technique

It is a method that exploits the knowledge that should be represented in such a way that:

- Knowledge captures generalization. Situations that share common properties are grouped together. Without the property, excessive amount of memory and modifications will be required.
- It can be easily modified to correct errors and to reflect changes in the world.
- It can be used in many situations even though it may not be totally accurate or complete.
- It can be used to reduce its own volume by narrowing range of possibilities.

There are three important AI techniques:

- **Search:** Provides a way of solving problems for which no direct approach is available. It also provides a framework into which any direct techniques that are available can be embedded.
- **Use of knowledge:** This technique provides a way of solving complex problem by exploiting the structures of the object that are involved.
- **Abstraction:** This technique provides the way of separating important features and variations from the many unimportant one i.e., it hides the detail of something.

Example: if we want to find the square root of a number then we simply call the function sqrt() in C. we don’t want to know the implementation details of the function.

General problem solving

- Problem solving in games such as “sudoku”. It can be done by building an AI system to solve a particular problem.
- To do this one needs to design the problem statement first and then generating the solution by keeping the condition in mind.
- Some of the most popularly used problem solving with the help of AI are:
  - ✓ Chess
  - ✓ Travelling salesman problem
  - ✓ Tower of Hanoi problem
  - ✓ Water jug problem
  - ✓ N-queen problem
  - ✓

Agent in AI

An intelligent agent (IA) is **an entity that makes a decision, that enables artificial intelligence to be put into action**. It can also be described as a software entity that conducts operations in the place of users or programs after sensing the environment. It uses actuators to initiate action in that environment.

Types of AI Agents

- Simple Reflex Agent
- Model-based reflex agent
- Goal-based agents
- Utility-based agent
- Learning agent

Simple Reflex agent:

The Simple reflex agents are the simplest agents. These agents take decisions on the basis of the current percepts and ignore the rest of the percept history.

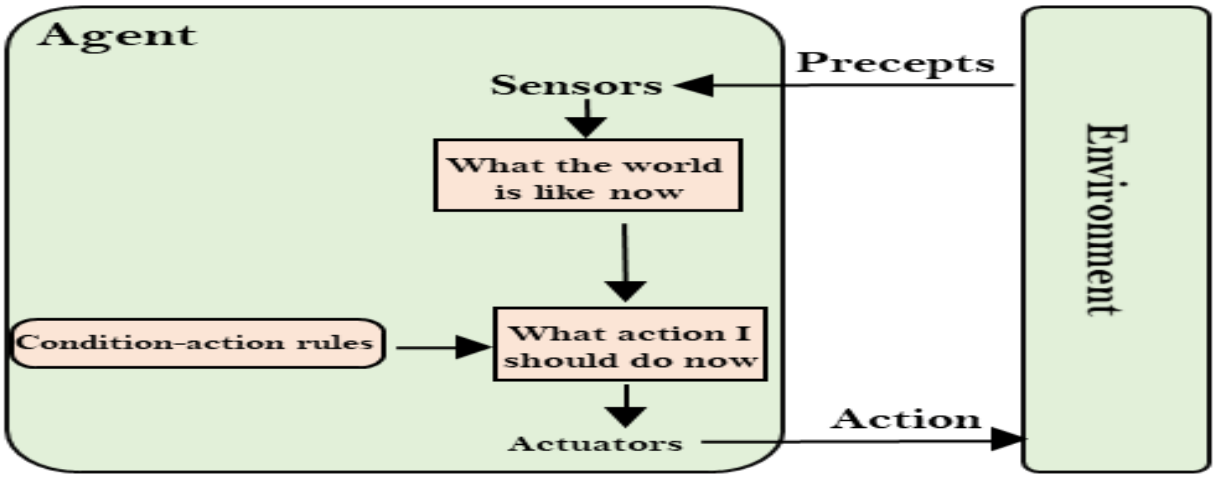
These agents only succeed in the fully observable environment.

The Simple reflex agent does not consider any part of percepts history during their decision and action process.

The Simple reflex agent works on Condition-action rule, which means it maps the current state to action. Such as a Room Cleaner agent, it works only if there is dirt in the room.

Problems for the simple reflex agent design approach:

- They have very limited intelligence
- They do not have knowledge of non-perceptual parts of the current state
- Mostly too big to generate and to store.
- Not adaptive to changes in the environment.



Model-based reflex agent

The Model-based agent can work in a partially observable environment, and track the situation.

A model-based agent has two important factors:

Model: It is knowledge about "how things happen in the world," so it is called a Model-based agent.

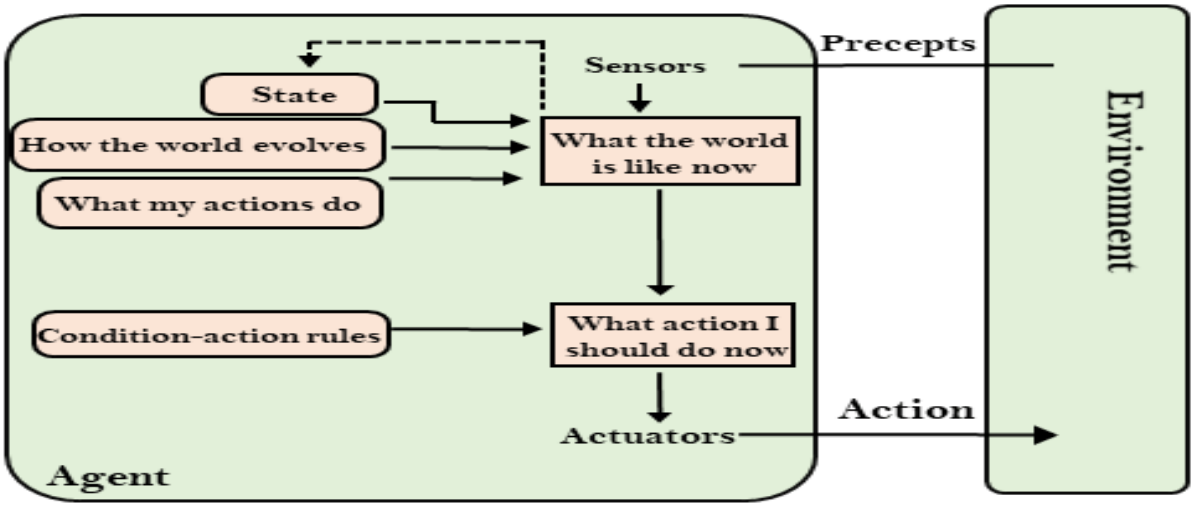
Internal State: It is a representation of the current state based on percept history.

These agents have the model, "which is knowledge of the world" and based on the model they perform actions.

Updating the agent state requires information about:

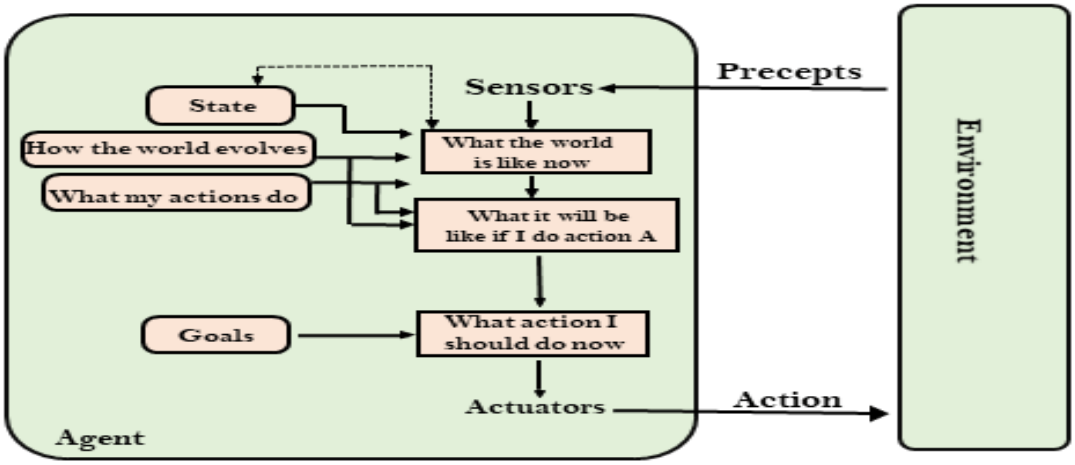
How the world evolves

How the agent's action affects the world.



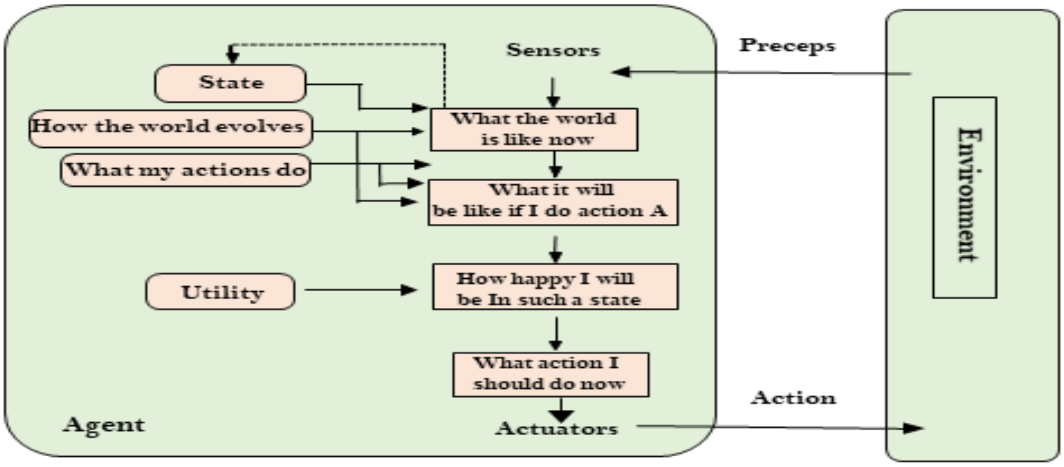
Goal-based agents

- The knowledge of the current state environment is not always sufficient to decide for an agent to what to do.
- The agent needs to know its goal which describes desirable situations.
- Goal-based agents expand the capabilities of the model-based agent by having the "goal" information.
- They choose an action, so that they can achieve the goal.
- These agents may have to consider a long sequence of possible actions before deciding whether the goal is achieved or not. Such considerations of different scenario are called searching and planning, which makes an agent proactive.



Utility-based agents

- These agents are similar to the goal-based agent but provide an extra component of utility measurement which makes them different by providing a measure of success at a given state.
- Utility-based agent act based not only goals but also the best way to achieve the goal.
- The Utility-based agent is useful when there are multiple possible alternatives, and an agent has to choose in order to perform the best action.
- The utility function maps each state to a real number to check how efficiently each action achieves the goals.

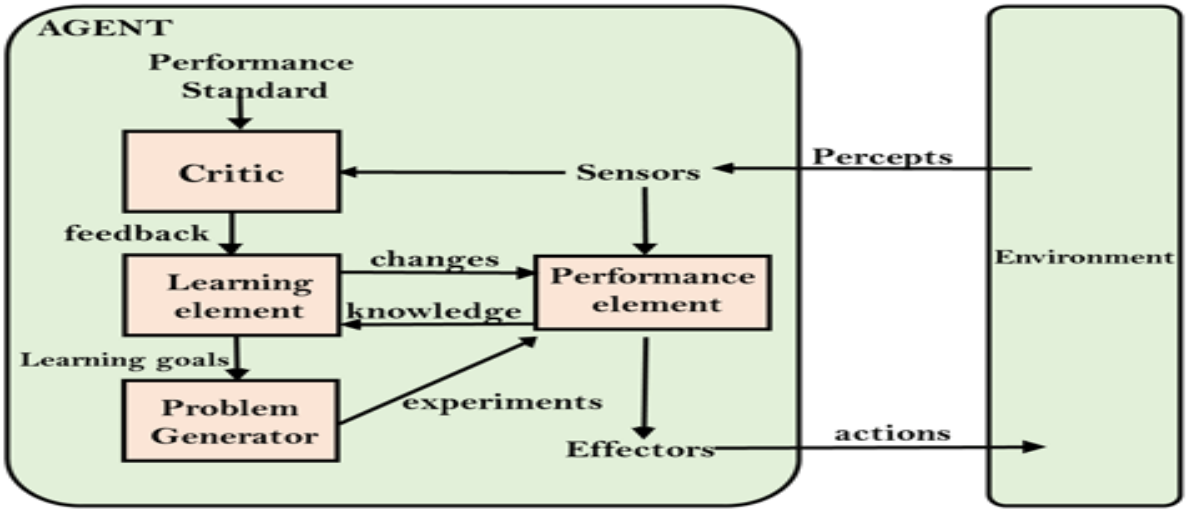


Learning Agents

- A learning agent in AI is the type of agent which can learn from its past experiences, or it has learning capabilities.
- It starts to act with basic knowledge and then able to act and adapt automatically through learning.
- A learning agent has mainly four conceptual components, which are:
    - **Learning element:** It is responsible for making improvements by learning from environment
    - **Critic:** Learning element takes feedback from critic which describes that how well the agent is doing with respect to a fixed performance standard.
    - **Performance element:** It is responsible for selecting external action
    - **Problem generator:** This component is responsible for suggesting actions that will lead to new and informative experiences.

Hence, learning agents are able to learn, analyze performance, and look for new ways to improve the performance.

- Search Algorithms in Artificial Intelligence
- Search algorithms are one of the most important areas of Artificial Intelligence. This topic will explain all about the search algorithms in AI.



Agent Environment in AI

An environment is everything in the world which surrounds the agent, but it is not a part of an agent itself. An environment can be described as a situation in which an agent is present.

The environment is where agent lives, operate and provide the agent with something to sense and act upon it. An environment is mostly said to be non-feministic.

Features of Environment

As per Russell and Norvig, an environment can have various features from the point of view of an agent:

1. Fully observable vs Partially Observable
2. Static vs Dynamic

3. Discrete vs Continuous
4. Deterministic vs Stochastic
5. Single-agent vs Multi-agent
6. Episodic vs sequential
7. Known vs Unknown
8. Accessible vs Inaccessible

1. Fully observable vs Partially Observable:

- If an agent sensor can sense or access the complete state of an environment at each point of time then it is **a fully observable** environment, else it is **partially observable**.
- A fully observable environment is easy as there is no need to maintain the internal state to keep track history of the world.
- An agent with no sensors in all environments then such an environment is called as **unobservable**.

2. Deterministic vs Stochastic:

- If an agent's current state and selected action can completely determine the next state of the environment, then such environment is called a deterministic environment.
- A stochastic environment is random in nature and cannot be determined completely by an agent.
- In a deterministic, fully observable environment, agent does not need to worry about uncertainty.

3. Episodic vs Sequential:

- In an episodic environment, there is a series of one-shot actions, and only the current percept is required for the action.
- However, in Sequential environment, an agent requires memory of past actions to determine the next best actions.

4. Single-agent vs Multi-agent

- If only one agent is involved in an environment, and operating by itself then such an environment is called single agent environment.
- However, if multiple agents are operating in an environment, then such an environment is called a multi-agent environment.
- The agent design problems in the multi-agent environment are different from single agent environment.

5. Static vs Dynamic:

- If the environment can change itself while an agent is deliberating then such environment is called a dynamic environment else it is called a static environment.

- Static environments are easy to deal because an agent does not need to continue looking at the world while deciding for an action.
- However for dynamic environment, agents need to keep looking at the world at each action.
- Taxi driving is an example of a dynamic environment whereas Crossword puzzles are an example of a static environment.

6. Discrete vs Continuous:

- If in an environment there are a finite number of percepts and actions that can be performed within it, then such an environment is called a discrete environment else it is called continuous environment.
- A chess gamecomes under discrete environment as there is a finite number of moves that can be performed.
- A self-driving car is an example of a continuous environment.

7. Known vs Unknown

- Known and unknown are not actually a feature of an environment, but it is an agent's state of knowledge to perform an action.
- In a known environment, the results for all actions are known to the agent. While in unknown environment, agent needs to learn how it works in order to perform an action.
- It is quite possible that a known environment to be partially observable and an Unknown environment to be fully observable.

8. Accessible vs Inaccessible

- If an agent can obtain complete and accurate information about the state's environment, then such an environment is called an Accessible environment else it is called inaccessible.
- An empty room whose state can be defined by its temperature is an example of an accessible environment.
- Information about an event on earth is an example of Inaccessible environment.

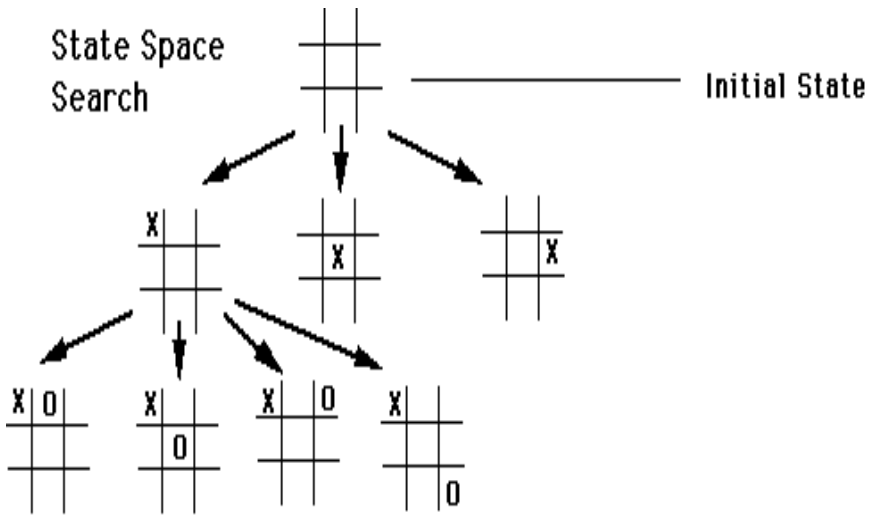
Problem-solving agents:

- 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. In this topic, we will learn various problem solving search algorithms.
- Search Algorithm Terminologies:
- **Search:** Searching is a step by step procedure to solve a search-problem in a given search space. A search problem can have three main factors:
    - **Search Space:** Search space represents a set of possible solutions, which a system may have.
    - **Start State:** It is a state from where agent begins **the search**.
    - **Goal test:** It is a function which observe the current state and returns whether the goal state is achieved or not.

- **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.
- **Actions:** It gives the description of all the available actions to the agent.
- **Transition model:** A description of what each action do, can be represented as a transition model.
- **Path Cost:** It is a function which assigns a numeric cost to each path.
- **Solution:** It is an action sequence which leads from the start node to the goal node.
- **Optimal Solution:** If a solution has the lowest cost among all solutions.

Search Algorithm Terminologies:

- Search: Searching is a step by step procedure to solve a search-problem in a given search space. A search problem can have three main factors:
  - ✓ Search Space: Search space represents a set of possible solutions, which a system may have.
  - ✓ Start State: It is a state from where agent begins the search.
  - ✓ Goal test: It is a function which observe the current state and returns whether the goal state is achieved or not.



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.

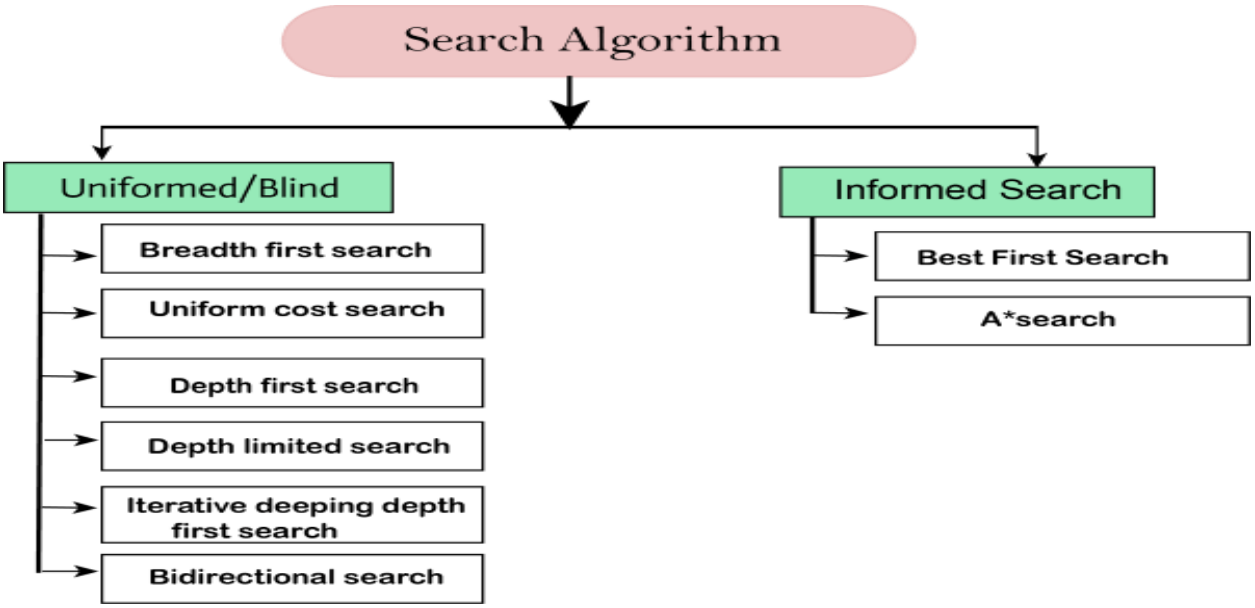
Optimality: If a solution found for an algorithm is guaranteed to be the best solution (lowest path cost) among all other solutions, then such a solution for is said to be an optimal solution.

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:

The uninformed search does not contain any domain knowledge such as closeness, the location of the goal. It operates in a brute-force way as it only includes information about how to traverse the tree and how to identify leaf and goal nodes. Uninformed search applies a way in which search tree is searched without any information about the search space like initial state operators and test for the goal, so it is also called blind search.It examines each node of the tree until it achieves the goal node.

It can be divided into five main types:

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

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.

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

- Greedy Search

- A* 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.

Algorithm:

Step 1: SET STATUS = 1 (ready state) for each node in G

Step 2: Enqueue the starting node A and set its STATUS = 2 (waiting state)

Step 3: Repeat Steps 4 and 5 until QUEUE is empty

Step 4: Dequeue a node N. Process it and set its STATUS = 3 (processed state).

Step 5: Enqueue all the neighbours of N that are in the ready state (whose STATUS = 1) and set their STATUS = 2(waiting state)

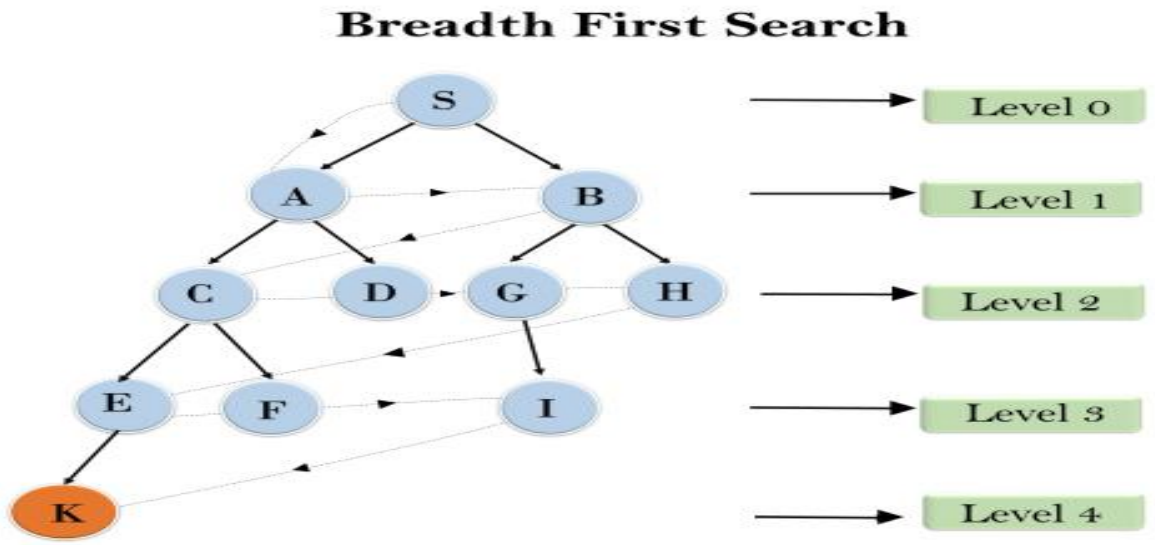
[END OF LOOP]

Time complexity=$O(b^d)$, where b is the branch factor and d is the depth

Space complexity=$O(b^d)$

Example: In the below tree structure, we have shown the traversing of the tree using BFS algorithm from the root node S to goal node K. BFS search algorithm traverse in layers, so it will follow the path which is shown by the dotted arrow, and the traversed path will be:

- S---> A--->B---->C--->D---->G--->H--->E---->F---->I---->K



Advantages:

- BFS will provide a solution if any solution exists.
- If there are more than one solutions for a given problem, then BFS will provide the minimal solution which requires the least number of steps.

Disadvantages:

- It requires lots of memory since each level of the tree must be saved into memory to expand the next level.
- BFS needs lots of time if the solution is far away from the root node.

Depth-first Search:

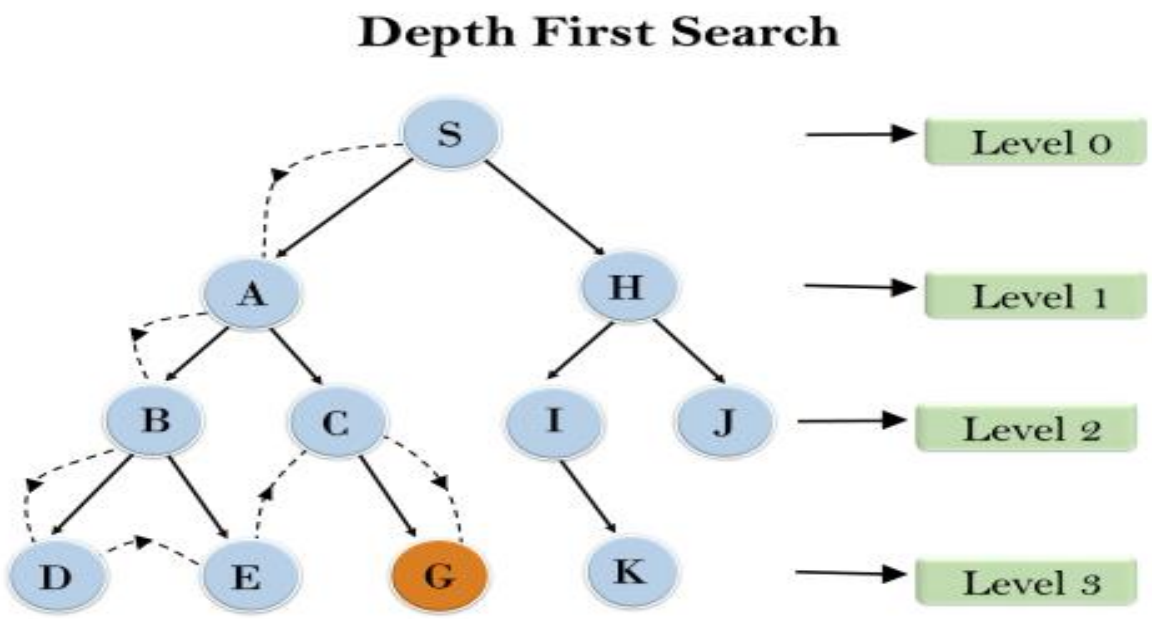
- Depth-first search is a recursive algorithm for traversing a tree or graph data structure.
- It is called the depth-first search because it starts from the root node and follows each path to its greatest depth node before moving to the next path.
- DFS uses a stack data structure for its implementation.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

Example:

In the below search tree, we have shown the flow of depth-first search, and it will follow the order as:

Root node--->Left node ----> right node.

It will start searching from root node S, and traverse A, then B, then D and E, after traversing E, it will backtrack the tree as E has no other successor and still goal node is not found. After backtracking it will traverse node C and then G, and here it will terminate as it found goal node.



Algorithm

Step 1: SET STATUS = 1 (ready state) for each node in G

Step 2: Push the starting node A on the stack and set its STATUS = 2 (waiting state)

Step 3: Repeat Steps 4 and 5 until STACK is empty

Step 4: Pop the top node N. Process it and set its STATUS = 3 (processed state)

Step 5: Push on the stack all the neighbors of N that are in the ready state (whose STATUS = 1) and set their STATUS = 2 (waiting state)

[END OF LOOP]

Step 6: EXIT

Time complexity=$O(b^d)$, where b is the branch factor and d is the depth

Space complexity=O(b*d)

Advantage:

- DFS requires very less memory as it only needs to store a stack of the nodes on the path from root node to the current node.
- It takes less time to reach to the goal node than BFS algorithm (if it traverses in the right path).

Disadvantage:

- There is the possibility that many states keep re-occurring, and there is no guarantee of finding the solution.
- DFS algorithm goes for deep down searching and sometime it may go to the infinite loop.

Depth-Limited Search Algorithm:

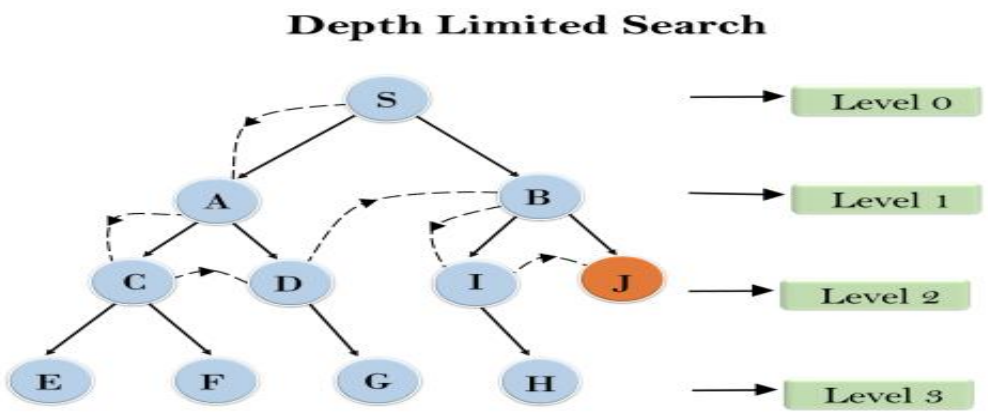
- A depth-limited search algorithm is similar to depth-first search with a predetermined limit. Depth-limited search can solve the drawback of the infinite path in the Depth-first search.
- In this algorithm, the node at the depth limit will treat as it has no successor nodes further.
- Depth-limited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

Advantages:

- Depth-limited search is Memory efficient.

Disadvantages:

- Depth-limited search also has a disadvantage of incompleteness.
- It may not be optimal if the problem has more than one solution.



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

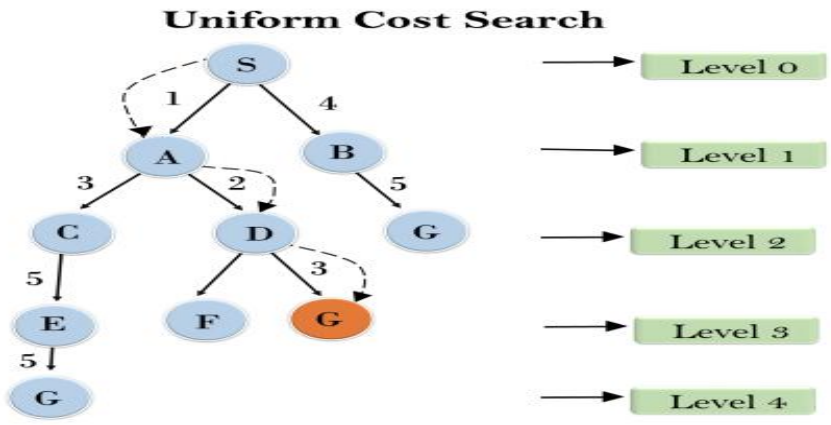
Time Complexity: Time complexity of DLS algorithm is $\mathbf{O(b^{\ell})}$.

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

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.
- Uniform-cost search expands nodes according to their path costs form the root node.
- It can be used to solve any graph/tree where the optimal cost is in demand.
- A uniform-cost search algorithm is implemented by the priority queue.
- It gives maximum priority to the lowest cumulative cost.

- Uniform cost search is equivalent to BFS algorithm if the path cost of all edges is the same.



Advantages:

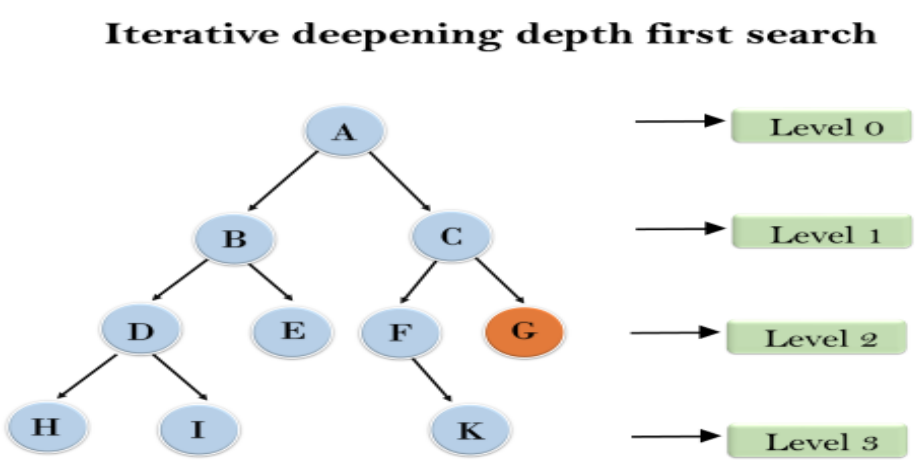
Uniform cost search is optimal because at every state the path with the least cost is chosen.

Disadvantages:

It does not care about the number of steps involve in searching and only concerned about path cost. Due to which this algorithm may be stuck in an infinite loop.

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.



1'st Iteration-----> A
2'nd Iteration----> A, B, C
3'rd Iteration------>A, B, C, D, E, F, G
In the third iteration, the algorithm will find the goal node.

Completeness: This algorithm is complete is if the 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.

Advantages:

It combines the benefits of BFS and DFS search algorithm in terms of fast search and memory efficiency.

Disadvantages:

The main drawback of IDDFS is that it repeats all the work of the previous phase.

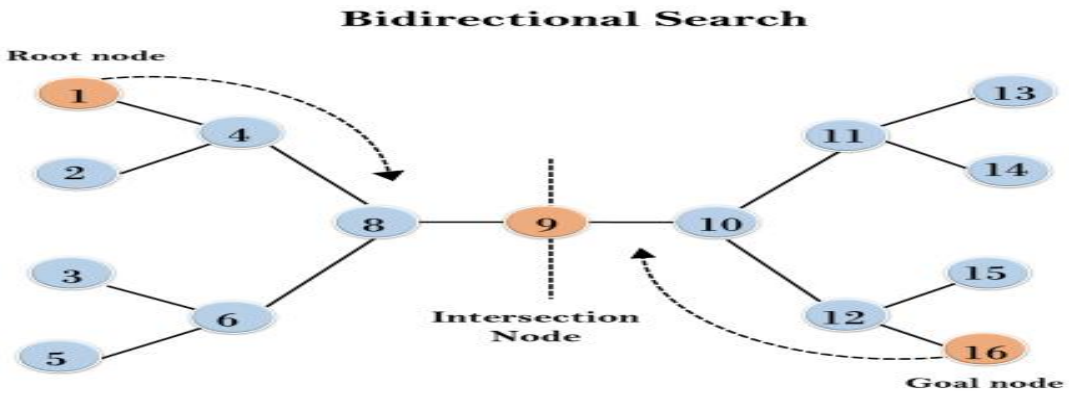
Bidirectional Search Algorithm:

- 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.

Example:

In the below search tree, bidirectional search algorithm is applied. This algorithm divides one graph/tree into two sub-graphs. It starts traversing from node 1 in the forward direction and starts from goal node 16 in the backward direction.

The algorithm terminates at node 9 where two searches meet.



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.

Advantages:

- Bidirectional search is fast.
- Bidirectional search requires less memory

Disadvantages:

- Implementation of the bidirectional search tree is difficult.
- **In bidirectional search, one should know the goal state in advance.**

Informed search algorithm

- Informed search algorithm contains an array of knowledge such as **how far we are from the goal**, **path cost**, **how to reach to goal node**, etc.
- This knowledge help agents to explore less to the search space and find more efficiently the goal node.
- 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 search:

- A **heuristics** in the most common sense is a method involving adapting the approach to a problem based on **previous solutions to similar problems**.
- Heuristic search is class of method which is used in order to search a solution space for **an optimal solution** for a problem.
- The heuristic here uses some method to search the solution space while assessing where in the space the solution is most likely to be and focusing the search on that area.
    - Hill Climbing

- Simulated annealing
- Best First Search
- Iterative deepening A* search

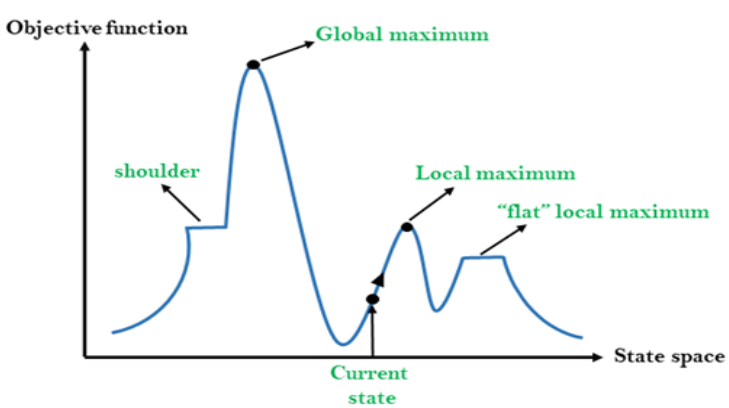
Hill Climbing Algorithm

- 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:

- **Local search algorithm:** Explores and evaluates different solutions (search space) by applying local changes until an optimal solution is achieved or certain iterations are computed.
- **Greedy approach:** Hill-climbing algorithm search moves in the direction which optimizes the cost.
- **No backtracking:** It does not backtrack the search space, as it does not remember the previous states.

State-space Diagram for Hill Climbing:



Different regions in the state space landscape:

- **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** (plateau) **:** 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.

Types of Hill Climbing Algorithm:

- Simple hill Climbing:
- Steepest-Ascent hill-climbing:
- Stochastic hill Climbing:

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

<u>Algorithm for Simple Hill Climbing:</u>

Step 1: Evaluate the initial state.

Step 2: Loop Until a solution is found or there is no new operator left to apply.

Step 3: Select and apply a new operator.

Step 4: Evaluate the new state:

a. If it is goal state, then quit.
b. Else if it is better than the current state then it is new current state.
c. Else if not better than the current state, then return to step2.

Step 5: Exit.

<u>Steepest-Ascent hill climbing:</u>

- 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 stop, else make current<-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:
    i. Apply the new operator and generate a new state.
    ii. Evaluate the new state.
    iii. If it is goal state, then return it and quit, else compare it to the SUCC.
    iv. If it is better than SUCC, then set new state as SUCC.
    v. If the SUCC is better than the current state, then set current state to SUCC.

Step 3: Exit.

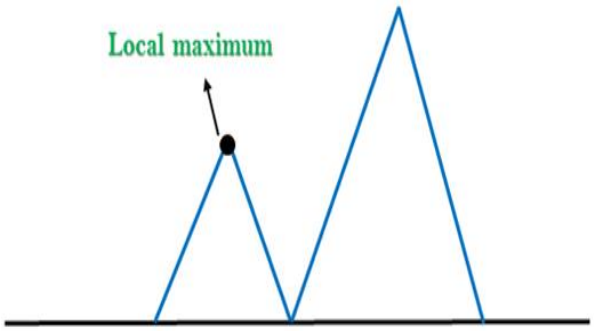
Stochastic hill climbing:

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:

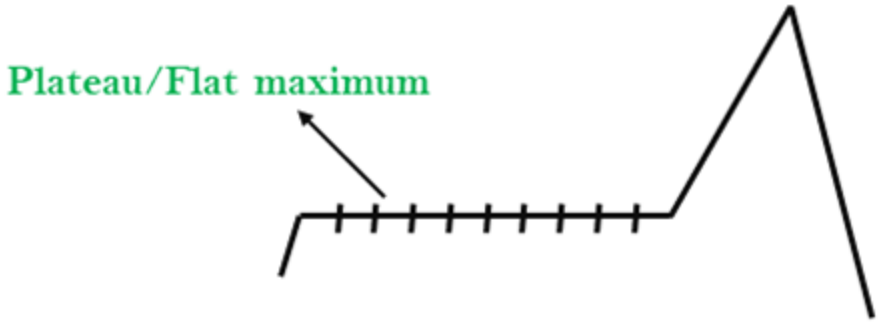
1. Local Maximum: A local maximum is a peak state in the landscape which is better than each of its neighboring states, but there is another state also present which is higher than the local maximum.

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.



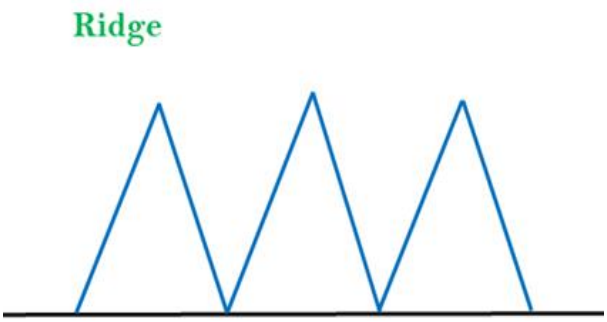
2. Plateau: A plateau is the flat area of the search space in which all the neighbor states of the current state contains the same value, because of this algorithm does not find any best direction to move. A hill-climbing search might be lost in the plateau area.

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.

Advantage:

- Easy to code for complex problems also.
- Give good solution.
- Statistically guarantees finding an optimal solution.

Disadvantage:

- Slow process.
- Can’t tell whether an optimal solution is found.

Simple hill climbing vs. simulated annealing

Simulated annealing	**simple hill climbing**
Annealing schedule is maintained	X
Moves to worst state may be accepted	X
Best state found so far is also maintained.	X

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). where, h(n)= estimated cost from node n to the goal.
- The greedy best first algorithm is implemented by the priority queue.

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.

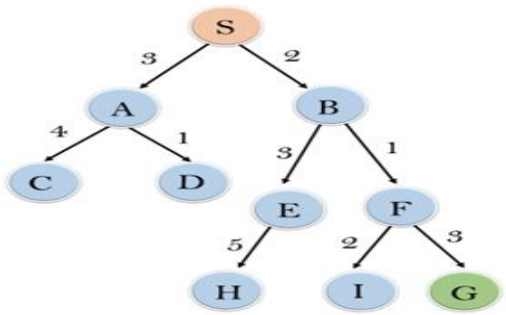
Step 5: Check each successor of node n, and find whether any node is a goal node or not. If any successor node is goal node, then return success and terminate the search, else proceed to Step 6.

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.

Example:

Consider the below search problem, and we will traverse it using greedy best-first search. At each iteration, each node is expanded using evaluation function f(n)=h(n) , which is given in the below table.



node	H (n)
A	12
B	4
C	7
D	3
E	8
F	2
H	4
I	9
S	13
G	0

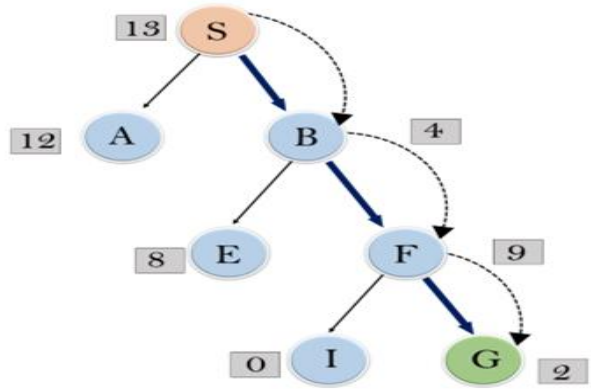
Expand the nodes of S and put in the CLOSED list

Initialization: Open [A, B], Closed [S]

Iteration 1: Open [A], Closed [S, B]

Iteration 2: Open [E, F, A], Closed [S, B]
: Open [E, A], Closed [S, B, F]

Iteration 3: Open [I, G, E, A], Closed [S, B, F]
: Open [I, E, A], Closed [S, B, F, G]



Hence the final solution path will be: **S----> B----->F----> G**

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.

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.

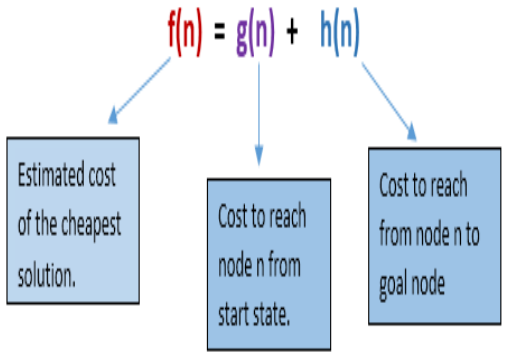
Disadvantages:

- 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.

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**.



<u>Algorithm:</u>

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.

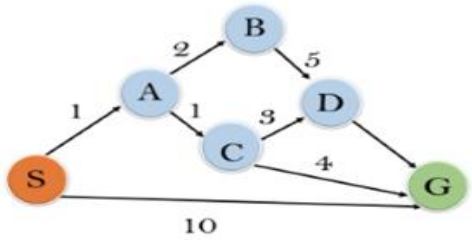
Step 5: Else if node n' is already in OPEN and CLOSED, then it should be attached to the back pointer which reflects the lowest g(n') value.

Step 6: Return to **Step 2**.

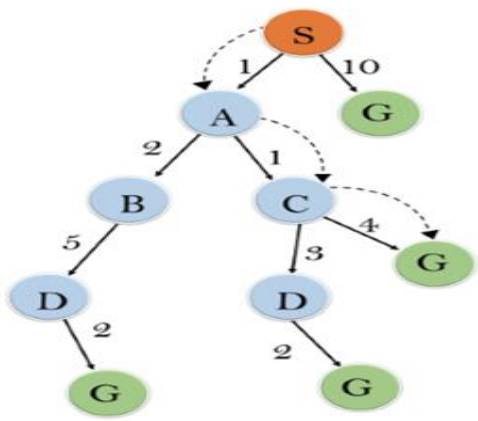
Example:

In this example, we will traverse the given graph using the A* algorithm. The heuristic value of all states is given in the below table so we will calculate the f(n) of each state using the formula f(n)= g(n) + h(n), where g(n) is the cost to reach any node from start state.

Here we will use OPEN and CLOSED list.



State	h(n)
S	5
A	3
B	4
C	2
D	6
G	0



Initialization: {(S, 5)}

Iteration1: {(S--> A, 4), (S-->G, 10)}

Iteration2: {(S--> A-->C, 4), (S--> A-->B, 7), (S-->G, 10)}

Iteration3: {(S--> A-->C--->G, 6), (S--> A-->C--->D, 11), (S--> A-->B, 7), (S-->G, 10)}

Iteration 4 will give the final result, as **S--->A--->C--->G** it provides the optimal path with cost 6.

Points to remember:

A* algorithm returns the path which occurred first, and it does not search for all remaining paths.

The efficiency of A* algorithm depends on the quality of heuristic.

A* algorithm expands all nodes which satisfy the condition f(n)<="" li="">

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)**

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.

Iterative deepening A*

- **Iterative deepening A*** (**IDA***) is a graph traversal and path search algorithm that can find the shortest path between a designated start node and any member of a set of goal nodes in a weighted graph.
- It is similar to iterative deepening depth first search algorithm.
- But the difference between IDA* algorithm and IDDFS is that the depth cutoff is used is F-cost(g+h) rather than depth only.
- Since it is a depth-first search algorithm, its memory usage is lower than in A*, but unlike ordinary iterative deepening search, it concentrates on exploring the most promising nodes and thus does not go to the same depth everywhere in the search tree.

While the standard iterative deepening depth-first search uses search depth as the cutoff for each iteration, the IDA* uses the more informative f(n)=g(n)+h(n), where g(n) is the cost to travel from the root to node n and h(n) is a problem-specific heuristic estimate of the cost to travel from n to the goal.

Algorithm

Step 1. first set the threshold a, the list value of f(n), where f(n)=g(n)+h(n)

Step 2. Now by checking if the threshold >= child then expand else check for the least f(n)

Step 3. Continue it until you reach to the goal node

Step 4. All rejected and stopped value added into f value list

Advantage:

- Finds an optimal solution(shortest number of steps)
- Less storage space

Disadvantage:

Computational time is more

Constrain satisfaction problem (CSP)

- CSP problem is composed of a finite set of variables, each of which has a finite domain of values, and a set of constraints.
- Each constraint is defined over some subset of the original set of variables and restricts the values these variables can simultaneously take.
- The task is to find an assignment of a value for each variable such that the assignments satisfy all the constraints, in some problems, the goal is to find all such assignments.
- Therefore, we can define CSP as a set of objects whose state must satisfy a number of constraints or limitations.

Example:

- N-queens problem
- graph coloring problem
- crossword puzzle

CSP consist of three components V, D , C

V is the set of variables $\{V_1, V_2, \ldots, V_n\}$

D is the set of domains $\{D_1, D_2, \ldots, D_n\}$ one for each variable

C is the set of constraints the specifies allowable combination of values

Example: Map-Coloring

Variables *WA, NT, Q, NSW, V, SA, T*

Domains D_i = {red, green, blue}

Constraints: adjacent regions must have different colors

e.g., WA ≠ NT

Solutions are **complete** and **consistent** assignments,

e.g., WA = red,

NT = green,

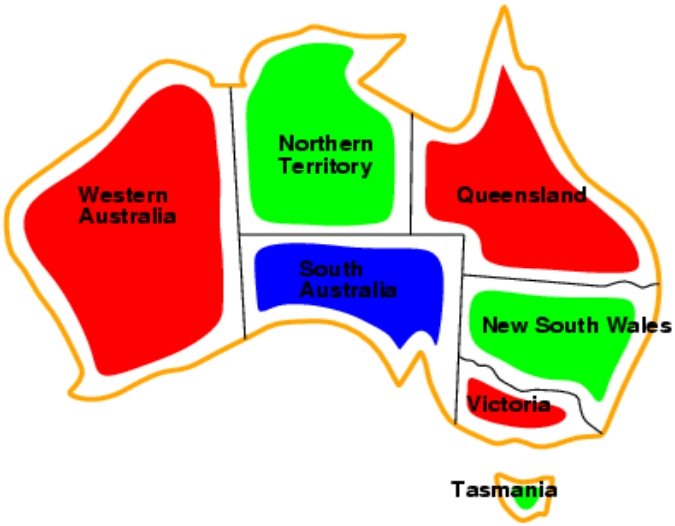
Q = red,

NSW = green,

V = red,

SA = blue,

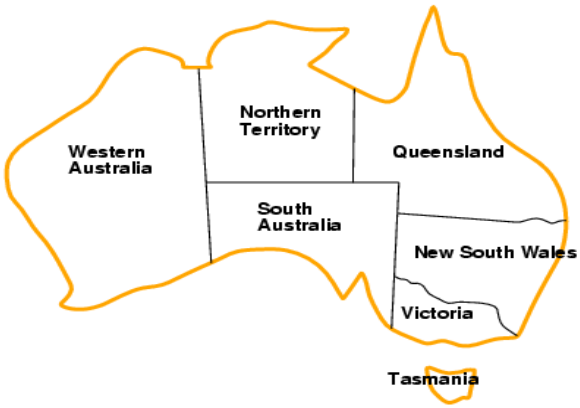
T = green

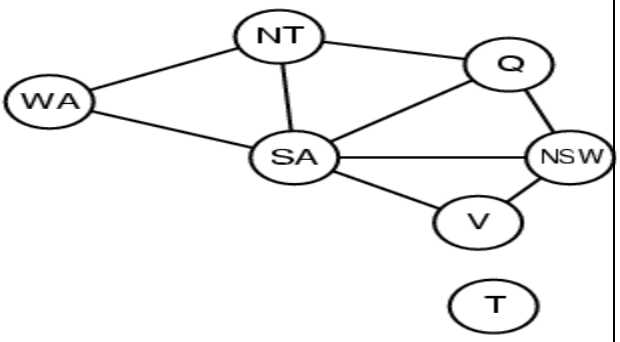


Binary CSP: each constraint relates two variables

Constraint graph: nodes are variables, arcs are constraints

Constraint graph





N-Queens Problem:

A classic combinational problem is to place n queens on a n*n chess board so that no two attack, i.,e notwo queens are on the same row, column or diagonal.

If we take n=4then the problem is called 4 queens problem.

If we take n=8 then the problem is called as 8 queens problem.

4-Queens problem:

Consider a 4*4 chessboard. Let there are 4 queens. The objective is place there 4 queens on 4*4 chessboard in such a way that no two queens should be placed in the same row, same column or diagonal position.

The explicit constraints are 4 queens are to be placed on 4*4 chessboards in 44 ways.

The implicit constraints are no two queens are in the same row column or diagonal. Let{x1, x2, x3, x4} be the solution vector where x1 column on which the queen i is placed.First queen is placed in first row and first column.

1			

The second queen should not be in first row and second column. It should be placed in second row and in second, third or fourth column. It we place in second column, both will be in same diagonal, so place it in third column.

1			
•	•		

1			
•	•	2	
•	•	•	•

We are unable to place queen 3 in third row, so go back to queen 2 and place it somewhereelse

1			
			2

1			
			2
	3		

Now the fourth queen should be placed in 4th row and 3rd column but there will be a diagonal attack from queen 3. So go back, remove queen 3 and place it in the next column. But it is not possible, so move back to queen 2 and remove it to next column but it is not possible. So go back to queen 1 and move it to next column.

	1		

	1		
			2

	1		
			2
3			

1			
			2
3			
		4	

Hence the solution of to 4-queens's problem is x1=2, x2=4, x3=1, x4=3, i.,e first queen is placed in 2nd column, second queen is placed in 4th column and third queen is placed in first column and fourth queen is placed in third column.

8 queen problem:

A classic combinatorial problem is to place 8 queens on a 8*8 chess board so that no two attack, i.,e no two queens are to the same row, column or diagonal.

Now, we will solve 8 queens problem by using similar procedure adapted for 4 queensproblem.The algorithm of 8 queens problem can be obtained by placing n=8, in N queens algorithm.

If two queens are placed at positions (i,j) and (k,l). They are on the same diagonal only ifi-

j=k-l.....(1)or

i+j=k+l .....(2).

From (1) and (2) implies j-l=i-k and j-l=k-i

Two queens lie on the same diagonal iff |j-l|=|i-k|

The solution of 8 queens problem can be obtained similar to the solution of 4 queens.
problem.X1=3, X2=6, X3=2, X4=7, X5=1, X6=4, X7=8, X8=5,
The solution can be shown as

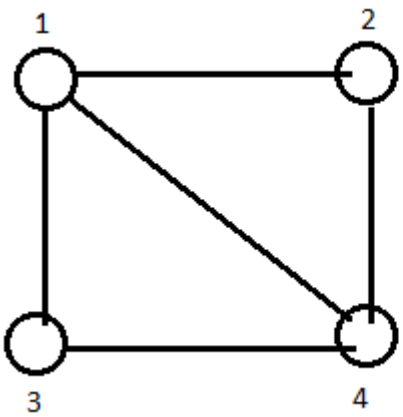
		1					
					2		
	3						
						4	
5							
			6				
							7
				8			

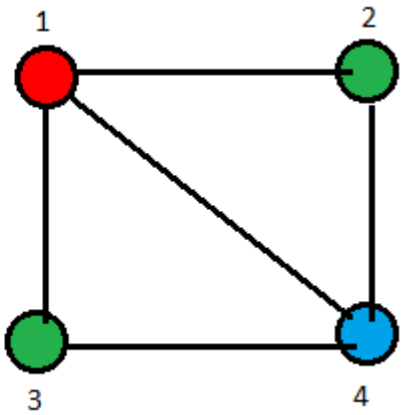
Vertex coloring problem

V={1, 2, 3, 4}

D={Red, Green, Blue}

C={1≠2, 1≠3, 1≠4, 2≠4, 3≠4}





Solution:

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