**CS3491 ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING**

**UNIT 1**

PROBLEM SOLVING

Introduction to AI - AI Applications - Problem solving agents – search algorithms – uninformed search strategies – Heuristic search strategies – Local search and optimization problems – adversarial search – constraint satisfaction problems (CSP)

## INTRODUCTION

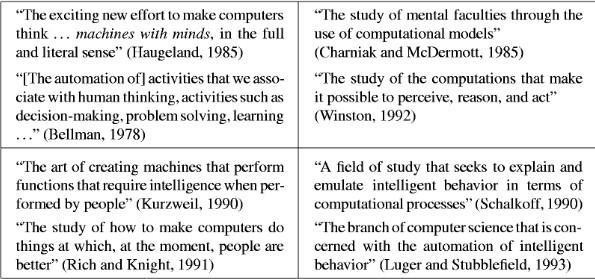
|  |  |
| --- | --- |
| **INTELLIGENCE** | **ARTIFICIAL INTELLIGENCE** |
| It is a natural process. | It is programmed by humans. |
| It is actually hereditary. | It is not hereditary. |
| Knowledge is required for intelligence. | KB and electricity are required to generate output. |
| No human is an expert. We may get better solutions from other humans. | Expert systems are made which aggregate many person’s experience and ideas. |

* 1. **DEFINITION**

The study of how to make computers do things at which at the moment, people are better.

### “Artificial Intelligence is the ability of a computer to act like a human being”.

* + - Systems that think like humans
    - Systems that act like humans
    - Systems that think rationally. Systems that act rationally**.**



**Figure 1.1 Some definitions of artificial intelligence, organized into four categories**

1. **Intelligence -** Ability to apply knowledge in order to perform better in an environment.
2. **Artificial Intelligence -** Study and construction of agent programs that perform well in a given environment, for a given agent architecture**.**
3. **Agent -** An entity that takes action in response to precepts from an environment**.**
4. **Rationality** - property of a system which does the “right thing” given what it knows.
5. **Logical Reasoning -** A process of deriving new sentences from old, such that the new sentences are necessarily true if the old ones are true.

Four Approaches of Artificial Intelligence:

* Acting humanly: The Turing test approach.
* Thinking humanly: The cognitive modelling approach.
* Thinking rationally: The laws of thought approach.
* Acting rationally: The rational agent approach.

In today's world, technology is growing very fast, and we are getting in touch with different new technologies day by day.

Here, one of the booming technologies of computer science is Artificial Intelligence which is ready to create a new revolution in the world by making intelligent machines.The Artificial Intelligence is now all around us. It is currently working with a variety of subfields, ranging from general to specific, such as self-driving cars, playing chess, proving theorems, playing music, Painting, etc.

AI is one of the fascinating and universal fields of Computer science which has a great scope in future. AI holds a tendency to cause a machine to work as a human.

Artificial Intelligence is composed of two words **Artificial** and **Intelligence**, where Artificial defines *"man- made,"* and intelligence defines *"thinking power"*, hence AI means *"a man-made thinking power."*

So, we can define AI as:

"It is a branch of computer science by which we can create intelligent machines which can behave like a human, think like humans, and able to make decisions."

Artificial Intelligence exists when a machine can have human based skills such as learning, reasoning, and solving problems

With Artificial Intelligence you do not need to preprogram a machine to do some work, despite that you can create a machine with programmed algorithms which can work with own intelligence, and that is the awesomeness of AI.

It is believed that AI is not a new technology, and some people says that as per Greek myth, there were Mechanical men in early days which can work and behave like humans.

**Why Artificial Intelligence?**

Before Learning about Artificial Intelligence, we should know that what is the importance of AI and why

should we learn it. Following are some main reasons to learn about AI:

* 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*

Following are the main goals of Artificial Intelligence:

1. Replicate human intelligence
2. Solve Knowledge-intensive tasks
3. An intelligent connection of perception and action
4. 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
5. 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?**

Artificial Intelligence is not just a part of computer science even it's so vast and requires lots of other factors which can contribute to it. To create the AI first we should know that how intelligence is composed, so the Intelligence is an intangible 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

Following are some main 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

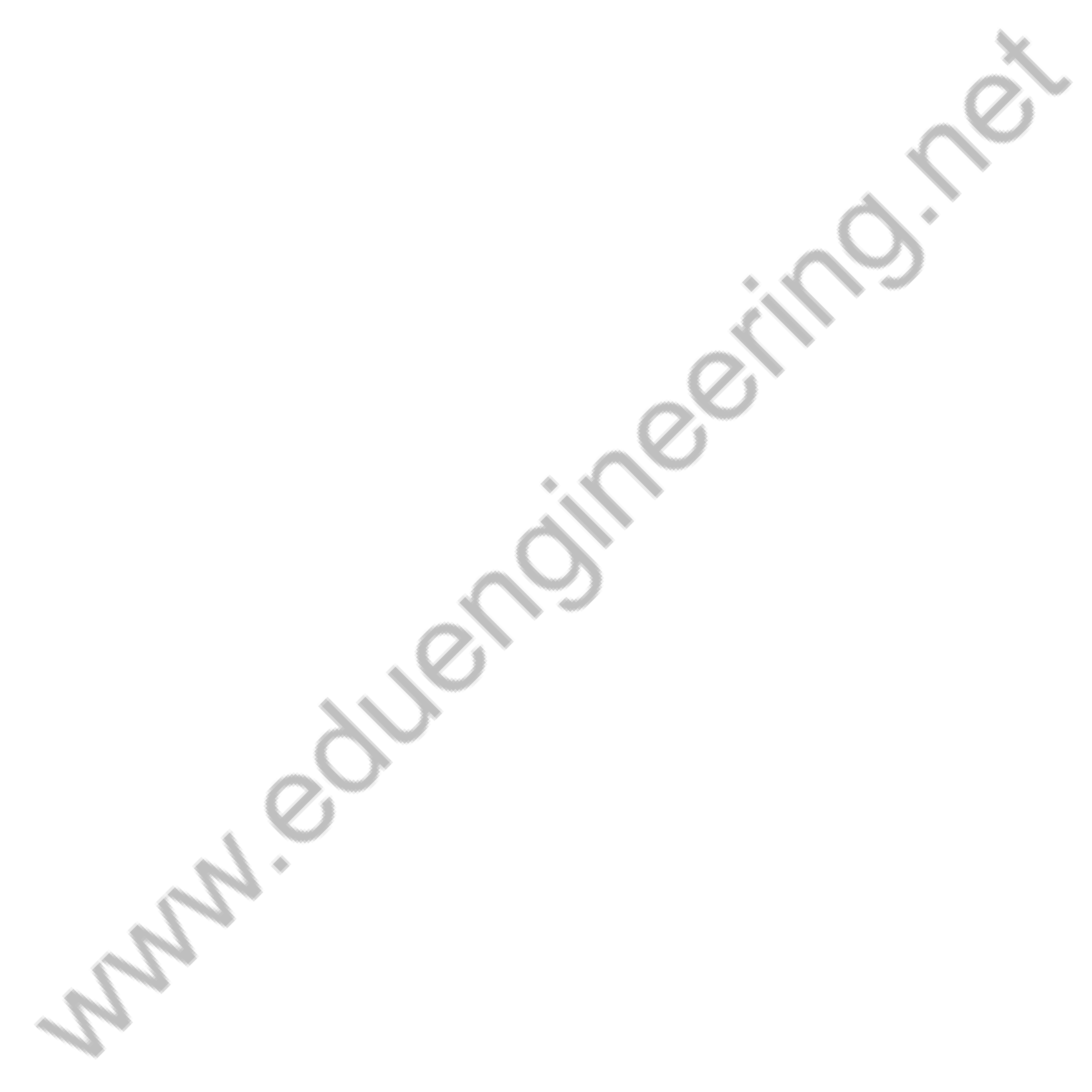
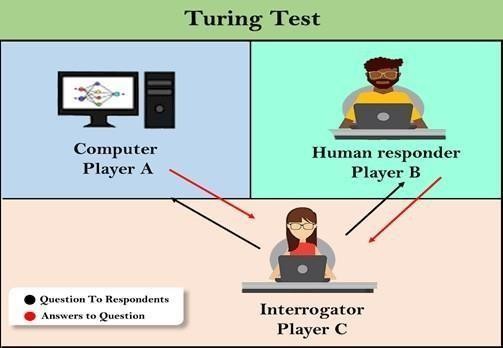
Every technology has some disadvantages, and thesame goes for Artificial intelligence. Being so advantageous technology still, it has some disadvantages which we need to keep in our mind while creating an AI system. Following are the disadvantages of AI:

* + **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.

#### Prerequisite

Before learning about Artificial Intelligence, you must have the fundamental knowledge of following so that you can understand the concepts easily:

* + Any computer language such as C, C++, Java, Python, etc.(knowledge of Python will be an advantage)
  + Knowledge of essential Mathematics such as derivatives, probability theory, etc.





## ACTING HUMANLY: THE TURING TEST APPROACH

The Turing Test, proposed by Alan Turing (1950), was designed to provide a satisfactory operational definition of intelligence. A computer passes the test if a human interrogator, after posing some written questions, cannot tell whether the written responses come from a person or from a computer.

* + - **natural language processing** to enable it to communicate successfully in English;
    - **knowledge representation** to store what it knows or hears;
    - **automated reasoning** to use the stored information to answer questions and to draw new conclusions
    - **machine learning** to adapt to new circumstances and to detect and extrapolate patterns.

**Total Turing Test** includes a video signal so that the interrogator can test the subject’s perceptual abilities, as well as the opportunity for the interrogator to pass physical objects “through the hatch.” To pass the total Turing Test, the computer will need

* + - * **computer vision** to perceive objects, and **robotics** to manipulate objects and move about**.**

Thinking humanly: The cognitive modelling approach

Analyse how a given program thinks like a human, we must have some way of determining how humans think. The interdisciplinary field of **cognitive science** brings together computer models from AI and experimental techniques from psychology to try to construct precise and testable theories of the workings of the human mind.

Although cognitive science is a fascinating field in itself, we are not going to be discussing it all that much in this book. We will occasionally comment on similarities or differences between AI techniques and human cognition. Real cognitive science, however, is necessarily based on experimental investigation of actual humans or animals, and we assume that the reader only has access to a computer for experimentation. We will simply note that AI and cognitive science continue to fertilize each other, especially in the areas of vision, natural language, and learning.

Thinking rationally: The “laws of thought” approach

The Greek philosopher Aristotle was one of the first to attempt to codify ``right thinking,'' that is, irrefutable reasoning processes. His famous syllogisms provided patterns for argument structures that always gave correct conclusions given correct premises.

For example, ``Socrates is a man; all men are mortal; therefore Socrates is mortal.''

These laws of thought were supposed to govern the operation of the mind, and initiated the field of ***logic.***

Acting rationally: The rational agent approach

Acting rationally means acting so as to achieve one's goals, given one's beliefs. An agent is just something that perceives and acts.

The right thing: that which is expected to maximize goal achievement, given the available information

Does not necessary involve thinking.

For Example - blinking reflex- but should be in the service of rational action.

## FUTURE OF ARTIFICIAL INTELLIGENCE

* + - **Transportation:** Although it could take a decade or more to perfect them, autonomous cars will one day ferry us from place to place.
    - **Manufacturing:** AI powered robots work alongside humans to perform a limited range of tasks like assembly and stacking, and predictive analysis sensors keep equipment running smoothly.
    - **Healthcare:** In the comparatively AI-nascent field of healthcare, diseases are more quickly and accurately diagnosed, drug discovery is sped up and streamlined, virtual nursing assistants monitor patients and big data analysis helps to create a more personalized patient experience.
    - **Education:** Textbooks are digitized with the help of AI, early-stage virtual tutors assist human instructors and facial analysis gauges the emotions of students to help determine who’s struggling or bored and better tailor the experience to their individual needs.
    - **Media:** Journalism is harnessing AI, too, and will continue to benefit from it. Bloomberg uses Cyborg technology to help make quick sense of complex financial reports. The Associated Press employs the natural language abilities of Automated Insights to produce 3,700 earning reports stories per year — nearly four times more than in the recent past
    - **Customer Service:** Last but hardly least, Google is working on an AI assistant that can place human-like calls to make appointments at, say, your neighborhood hair salon. In addition to words, the system understands context and nuance.

## CHARACTERISTICS OF INTELLIGENT AGENTS

### Situatedness

The agent receives some form of sensory input from its environment, and it performs some action that changes its environment in some way.

Examples of environments: the physical world and the Internet.

* + - Autonomy

The agent can act without direct intervention by humans or other agents and that it has control over its own actions and internal state.

* + - Adaptivity

The agent is capable of

1. reacting flexibly to changes in its environment;
2. taking goal-directed initiative (i.e., is pro-active), when appropriate; and
3. Learning from its own experience, its environment, and interactions with others.
   * + Sociability

#### Theagent is capable of interacting in a peer-to-peer manner with other agents or humans What is Artificial Intelligence?

Future of Artificial Intelligence:

1. Health Care Industries

India is 17.7% of the worlds’ population that makes it the second-largest country in terms of China’s population. Health care facilities are not available to all individuals living in the country. It is because of the lack of good doctors, not having good infrastructure, etc. Still, there are people who couldn’t reach to doctors/ hospitals. AI has the ability to provide the facility to detect disease based on symptoms; even if you don’t go to the doctor, AI would read the data from Fitness band/medical history of an individual to analyze the pattern and suggest proper medication and even deliver it on one’s fingertips just through cell-phone.

As mentioned earlier Google’s deep mind has already beaten doctors in detecting fatal diseases like breast cancer. It’s not far away when AI will be detecting common disease as well as providing proper suggestions for medication. The consequences of this could be: no need for doctors in the long term result in JOB reduction.

1. AI in Education

The development of a country depends on the quality of education youth is getting. Right now, we can see there are lots of courses are available on AI. But in the future AI is going to transform the classical way of education. Now the world doesn’t need skilled labourers for manufacturing industries, which is mostly replaced by robots and automation. The education system could be quite effective and can be according to the individual’s personality and ability. It would give chance brighter students to shine and to imbecile a better way to cop up.

Right Education can enhance the power of individuals/nations; on the other hand, misuse of the same could lead to devastating results.

1. AI in Finance

Quantification of growth for any country is directly related to its economic and financial condition. As AI has enormous scope in almost every field, it has great potential to boost individuals’ economic health and a nation. Nowadays, the AI algorithm is being used in managing equity funds.

An AI system could take a lot number of parameters while figuring out the best way to manage funds. It would perform better than a human manager. AI-driven strategies in the field of finance are going to change the classical way of trading and investing. It could be devastating for some fund managing firms who cannot

afford such facilities and could affect business on a large scale, as the decision would be quick and abrupt. The competition would be tough and on edge all the time.

1. AI in Military and Cybersecurity

AI-assisted Military technologies have built autonomous weapon systems, which won’t need humans at all hence building the safest way to enhance the security of a nation. We could see robot Military in the near future, which is as intelligent as a soldier/ commando and will be able to perform some tasks.

AI-assisted strategies would enhance mission effectiveness and will provide the safest way to execute it. The concerning part with AI-assisted system is that how it performs algorithm is not quite explainable. The deep neural networks learn faster and continuously keep learning the main problem here would be explainable AI. It could possess devastating results when it reaches in the wrong hands or makes wrong decisions on its own.

## AGENTS AND ITS TYPES

An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators.

* + - Human Sensors:
    - Eyes, ears, and other organs for sensors.
    - Human Actuators:
    - Hands, legs, mouth, and other body parts.
    - Robotic Sensors:
    - Mic, cameras and infrared range finders for sensors
    - Robotic Actuators:
    - Motors, Display, speakers etc An agent can be:

**Human-Agent:** A human agent has eyes, ears, and other organs which work for sensors

and hand, legs, vocal tract work for actuators.

**Robotic Agent:** A robotic agent can have cameras, infrared range finder, NLP for sensors and various motors for actuators.

**Software Agent:** Software agent can have keystrokes, file contents as sensory input and act on those inputs and display output on the screen.

Hence the world around us is full of agents such as thermostat, cell phone, camera, and even we are also agents. Before moving forward, we should first know about sensors, effectors, and actuators.

**Sensor:** Sensor is a device which detects the change in the environment and sends the information to other electronic devices. An agent observes its environment through sensors.

**Actuators:** Actuators are the component of machines that converts energy into motion. The actuators are only responsible for moving and controlling a system. An actuator can be an electric motor, gears, rails, etc.

**Effectors:** Effectors are the devices which affect the environment. Effectors can be legs, wheels, arms, fingers, wings, fins, and display screen.

## PROPERTIES OF ENVIRONMENT

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.

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.

**Example:** chess – the board is fully observable, as are opponent’s moves. Driving – what is around the next bend is not observable and hence partially observable.

### 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.

### 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.

### 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.

### 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.

### Discrete vs Continuous

* + If in an environment there are a finite number of precepts 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 game comes 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.

### 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.

### 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.

**Task environments**, which are essentially the "problems" to which rational agents are the "solutions."

**PEAS:** Performance Measure, Environment, Actuators, Sensors

#### Performance

The output which we get from the agent. All the necessary results that an agent gives after processing comes under its performance.

#### Environment

All the surrounding things and conditions of an agent fall in this section. It basically consists of all the things under which the agents work.

#### Actuators

The devices, hardware or software through which the agent performs any actions or processes any information to produce a result are the actuators of the agent.

#### Sensors

The devices through which the agent observes and perceives its environment are the sensors of the agent.

Eg. Crossing road. Here first perception occurs on both sides and then only action. No perception occurs in **Degenerate Agent**.

Eg. Clock. It does not view the surroundings. No matter what happens outside. The clock works based on inbuilt program.

* + **Ideal Agent** describes by ideal mappings. “Specifying which action an agent ought to take in response to any given percept sequence provides a design for ideal agent”.
  + **Eg.** SQRT function calculation in calculator.
  + Doing actions in order to modify future precepts-sometimes called **information gathering**- is an important part of rationality.
  + A rational agent should be **autonomous**-it should learn from its own prior knowledge (experience).

### The Structure of Intelligent Agents

#### Agent = Architecture + Agent Program

Architecture = the machinery that an agent executes on. (Hardware)

Agent Program = an implementation of an agent function. (Algorithm, Logic – Software)

## TYPES OF AGENTS

Agents can be grouped into four classes based on their degree of perceived intelligence and capability :

* + - Simple Reflex Agents
    - Model-Based Reflex Agents
    - Goal-Based Agents
    - Utility-Based Agents
    - Learning Agent

### The Simple reflex agents

* 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 (**past State).**
* 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.

**Condition-Action Rule** − It is a rule that maps a state (condition) to an action.

**Ex:** if car-in-front-is-braking then initiate- braking.

### Model Based Reflex Agents

* 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 **(“Level of Happiness”)** 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:
  1. **Learning element:** It is responsible for making improvements by learning from environment
  2. **Critic:** Learning element takes feedback from critic which describes that how well the agent is doing with respect to a fixed performance standard.
  3. **Performance element:** It is responsible for selecting external action
  4. **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.

## PROBLEM SOLVING APPROACH TO TYPICAL AI PROBLEMS

### 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, wewill learn various problem-solving search algorithms.

* + - Some of the most popularly used problem solving with the help of artificial intelligence are:

1. Chess.
2. Travelling Salesman Problem.
3. Tower of Hanoi Problem.
4. Water-Jug Problem.
5. N-Queen Problem.

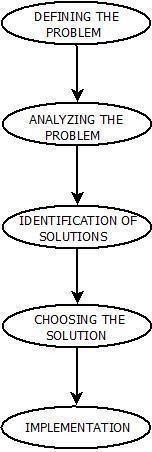
### Problem Searching

* + - In general, searching refers to as finding information one needs.
    - Searching is the most commonly used technique of problem solving in artificial intelligence.
    - The searching algorithm helps us to search for solution of particular problem.

**Problem:** Problems are the issues which comes across any system. A solution is needed to solve that particular problem.

### Steps : Solve Problem Using Artificial Intelligence

* + - The process of solving a problem consists of five steps. These are:



**Figure 1.11 Problem Solving in Artificial Intelligence**

**Defining The Problem**: The definition of the problem must be included precisely. It should contain the possible initial as well as final situations which should result in acceptable solution.

1. **Analyzing The Problem**: Analyzing the problem and its requirement must be done as few features can have immense impact on the resulting solution.
2. **Identification Of Solutions**: This phase generates reasonable amount of solutions to the given problem in a particular range.
3. **Choosing a Solution**: From all the identified solutions, the best solution is chosen basis on the results produced by respective solutions.
4. **Implementation**: After choosing the best solution, its implementation is done.

### Measuring problem-solving performance

We can evaluate an algorithm’s performance in four ways:

**Completeness:** Is the algorithm guaranteed to find a solution when there is one?

**Optimality:** Does the strategy find the optimal solution?

**Time complexity**: How long does it take to find a solution?

**Space complexity:** How much memory is needed to perform the search?

### 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:
  1. Search Space: Search space represents a set of possible solutions, which a system may have.
  2. Start State: It is a state from where agent begins the search.
  3. 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.

### Example Problems

A **Toy Problem** is intended to illustrate or exercise various problem-solving methods.

A**real- world problem** is one whose solutions people actually care about.

### Toy Problems

Vacuum World

**States:** The state is determined by both the agent location and the dirt locations. The agent is in one of the 2 locations, each of which might or might not contain dirt. Thus there are 2\*2^2=8 possible world states.

**Initial state**: Any state can be designated as the initial state.

**Actions**: In this simple environment, each state has just three actions: *Left*, *Right*, and

*Suck*. Larger environments might also include *Up* and *Down*.

**Transition model**: The actions have their expected effects, except that moving *Left* in the leftmost squ are, moving *Right* in the rightmost square, and *Suck*ing in a clean square have no effect. The complete state space is shown in Figure.

**Goal test**: This checks whether all the squares are clean.

**Path cost:** Each step costs 1, so the path cost is the number of steps in the path.

**States**: A state description specifies the location of each of the eight tiles and the blank in one of the nine squares.**Initial state**: Any state can be designated as the initial state. Note that any given goal can be reached from exactly half of the possible initial states.

The simplest formulation defines the actions as movements of the blank space *Left*, *Right*, *Up*, or *Down*. Different subsets of these are possible depending on where the blank is.

**Transition model**: Given a state and action, this returns the resulting state; for example, if we apply *Left* to the start state in Figure 3.4, the resulting state has the 5 and the blank switched.

**Goal test**: This checks whether the state matches the goal configuration shown in Figure. **Path cost:** Each step costs 1, so the path cost is the number of steps in the path.

### Queens Problem

* **States**: Any arrangement of 0 to 8 queens on the board is a state.
* **Initial state**: No queens on the board.
* **Actions**: Add a queen to any empty square.
* **Transition model**: Returns the board with a queen added to the specified square.
* **Goal test**: 8 queens are on the board, none attacked.

## PROBLEM SOLVING BY SEARCH

An important aspect of intelligence is ***goal-based*** problem solving.

The **solution** of many **problems** can be described by finding a **sequence of actions** that lead to a desirable **goal.** Each action changes the ***state*** and the aim is to find the sequence of actions and states that lead from the initial (start) state to a final (goal) state.

A well-defined problem can be described by: Initial state

* + - **Operator or successor function** - for any state x returns s(x), the set of states reachable from x with one action
    - **State space** - all states reachable from initial by any sequence of actions
    - **Path** - sequence through state space
    - **Path cost** - function that assigns a cost to a path. Cost of a path is the sum of costs of individual actions along the path
    - **Goal test** - test to determine if at goal state

### What is Search?

**Search** is the systematic examination of **states** to find path from the **start/root state** to the **goal state.**

The set of possible states, together with *operators* defining their connectivity constitute the *search space*.

The output of a search algorithm is a solution, that is, a path from the initial state to a state that satisfies the goal test.

### Problem-solving agents

A Problem solving agent is a **goal-based** agent. It decide what to do by finding sequence of actions that lead to desirable states. The agent can adopt a goal and aim at satisfying it.

To illustrate the agent’s behavior, let us take an example where our agent is in the city of Arad, which is in Romania. The agent has to adopt a **goal** of getting to Bucharest.

**Goal formulation**, based on the current situation and the agent’s performance measure, is the first step in problem solving.

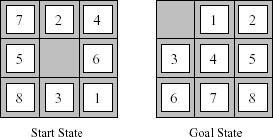
The agent’s task is to find out which sequence of actions will get to a goal state.

**Problem formulation** is the process of deciding what actions and states to consider given a goal.

**The 8-puzzle**

An 8-puzzle consists of a 3x3 board with eight numbered tiles and a blank space. A tile adjacent to the balank space can slide into the space. The object is to reach the goal state, as shown in Figure 2.4

### Example: The 8-puzzle



**Figure 2.2 A typical instance of 8-puzzle**

The problem formulation is as follows:

* **States** : A state description specifies the location of each of the eight tiles and the blank in one of the nine squares.
* **Initial state** : Any state can be designated as the initial state. It can be noted that any given goal can be reached from exactly half of the possible initial states.
* **Successor function** : This generates the legal states that result from trying the four actions(blank moves Left, Right, Up or down).
* **Goal Test** : This checks whether the state matches the goal configuration shown in Figure(Other goal configurations are possible)
* **Path cost** : Each step costs 1,so the path cost is the number of steps in the path.

**The 8**-**puzzle** belongs to the family **of sliding-block puzzles**, which are often used as test problems for new search algorithms in AI. This general class is known as NP-complete. The **8**-**puzzle** has 9!/2 = 181,440 reachable states and is easily solved.

The **15 puzzle** ( 4 x 4 board ) has around 1.3 trillion states, an the random instances can be solved optimally in few milli seconds by the best search algorithms.

The **24-puzzle** (on a 5 x 5 board) has around 1025 states and random instances are still quite difficult to solve optimally with current machines and algorithms.

### 8-Queens problem

The goal of 8-queens problem is to place 8 queens on the chessboard such that no queen attacks any other.(A queen attacks any piece in the same row, column or diagonal).

An **Incremental formulation** involves operators that augments the state description, starting with an empty state. For 8-queens problem, this means each action adds a queen to the state. A **complete-state formulation** starts with all 8 queens on the board and move them around. In either case the path cost is of no interest because only the final state counts.

The first incremental formulation one might try is the following:

* **States**: Any arrangement of 0 to 8 queens on board is a state.
* **Initial state**: No queen on the board.
* **Successor function**: Add a queen to any empty square.
* **Goal Test**: 8 queens are on the board, none attacked.

In this formulation, we have 64.63…57 = 3 x 1014 possible sequences to investigate.

A better formulation would prohibit placing a queen in any square that is already attacked.

* **States** : Arrangements of n queens ( 0 <= n < = 8 ),one per column in the left most columns, with no queen attacking another are states.
* **Successor function** : Add a queen to any square in the left most empty column such that it is not attacked by any other queen.

This formulation reduces the 8-queen state space from 3 x 1014 to just 2057,and solutions are easy to find.

For the 100 queens the initial formulation has roughly 10400 states whereas the improved formulation has about 1052 states. This is a huge reduction, but the improved state space is still too big for the algorithms to handle.

**Different Search Algorithm**

UNINFORMED SEARCH STRATGES

**Uninformed Search Strategies** have no additional information about states beyond that provided in the **problem definition**.

**Strategies** that know whether one non goal state is “more promising” than another are called

**Informed search or heuristic search** strategies.

There are five uninformed search strategies as given below.

* Breadth-first search
* Uniform-cost search
* Depth-first search
* Depth-limited search
* Iterative deepening search

### Breadth-first search

* + Breadth-first search is a simple strategy in which the root node is expanded first, then all successors of the root node are expanded next, then their successors, and so on. In general, all the nodes are expanded at a given depth in the search tree before any nodes at the next level are expanded.
  + Breath-first-search is implemented by calling TREE-SEARCH with an empty fringe that is a first-in-first-out (FIFO) queue, assuring that the nodes that are visited first will be expanded first. In otherwards, calling TREE-SEARCH (problem, FIFO-QUEUE()) results in breadth-first-search. The FIFO queue puts all newly generated successors at the end of the queue, which means that Shallow nodes are expanded before deeper nodes.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| **Figure 2.5 Breadth-first search on a simple binary tree. At each stage, the node to be expanded next is indicated by a marker.** | | | |

**Time complexity for BFS**

Assume every state has b successors. The root of the search tree generates b nodes at the first level, each of which generates b more nodes, for a total of b2 at the second level. Each of these generates b more nodes, yielding b3 nodes at the third level, and so on. Now suppose, that the solution is at depth d. In the worst case, we would expand all but the last node at level d, generating bd+1 - b nodes at level d+1.

Then the total number of nodes generated is b + b2 + b3 + …+ bd + ( bd+1 + b) = O(bd+1).

Every node that is generated must remain in memory, because it is either part of the fringe or is an ancestor of a fringe node. The space compleity is, therefore, the same as the time complexity

**Step 1:** Take an Empty Queue.

**Step 2:** Select a starting node (visiting a node) and insert it into the Queue.

**Step 3:** Provided that the Queue is not empty, extract the node from the Queue and insert its child nodes (exploring a node) into the Queue.

**Step 4:** Print the extracted node.

## UNIFORM-COST SEARCH

Instead of expanding the shallowest node, **uniform-cost search** expands the node n with the lowest path cost. Uniform-cost search does not care about the number of steps a path has, but only about their total cost.

## DEPTH-FIRST-SEARCH

Depth-first-search always expands the deepest node in the current fringe of the search tree. The progress of the search is illustrated in Figure 1.31. The search proceeds immediately to the deepest level of the search tree, where the nodes have no successors. As those nodes are expanded, they are dropped from the fringe, so then the search “backs up” to the next shallowest node that still has unexplored successors.

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| **Figure 2.7 Depth-first-search on a binary tree. Nodes that have been expanded and have node scendants in the fringe can be removed from the memory; these are shown in black. Nodes at depth 3 are assumed to have no successors and M is the only goal node.** | | |

This strategy can be implemented by TREE-SEARCH with a last-in-first-out (LIFO) queue, also known as a stack.

Depth-first-search has very modest memory requirements. It needs to store only a single path from the root to a leaf node, along with the remaining unexpanded sibling nodes for each node on the path. Once the node has been expanded, it can be removed from the memory, as soon as its descendants have been fully explored (Refer Figure 2.7).

function DFS((V,E))

mark each node in V with 0

count ← 0

for each vertex in V do

if vertex is marked then

DFSExplore(vertex)

function DFSExplore(vertex)

count ← count + 1

mark vertex with count

for each edge (vertex, neighbour) do

if neighbour is marked with 0 then

DFSExplore(neighbour)

For a state space with a branching factor b and maximum depth m, depth-first-search requires storage of only bm + 1 nodes.

Using the same assumptions as Figure, and assuming that nodes at the same depth as the goal node have no successors, we find the depth-first-search would require 118 kilobytes instead of 10 petabytes, a factor of 10 billion times less space.

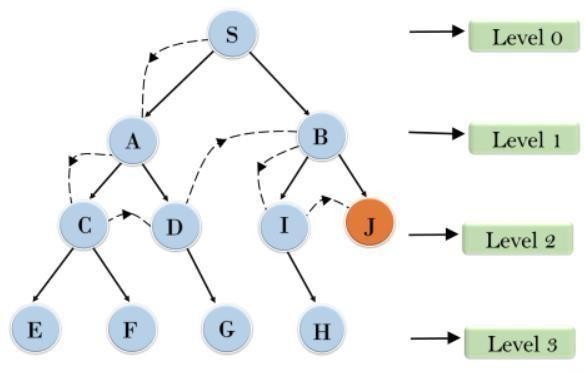
### Drawback of Depth-first-search

The drawback of depth-first-search is that it can make a wrong choice and get stuck going down very long(or even infinite) path when a different choice would lead to solution near the root of the search tree. For example, depth-first-search will explore the entire left subtree even if node C is a goal node.

## BACKTRACKING SEARCH

A variant of depth-first search called backtracking search uses less memory and only one successor is generated at a time rather than all successors.; Only O(m) memory is needed rather than O(bm)

## DEPTH-LIMITED-SEARCH (DLS)



**Figure 2.8 Depth-limited-search**

The problem of unbounded trees can be alleviated by supplying depth-first-search with a pre- determined depth limit l. That is, nodes at depth l are treated as if they have no successors. This approach is called **depth-limited-search**. The depth limit solves the infinite path problem.

Depth limited search will be nonoptimal if we choose l > d. Its time complexity is O(bl) and its space complete is O(bl). Depth-first-search can be viewed as a special case of depth- limited search with l = oo Sometimes, depth limits can be based on knowledge of the problem. For, example, on the map of Romania there are 20 cities. Therefore, we know that if there is a solution, it must be of length 19 at the longest, So l = 10 is a possible choice. However, it can be shown that any city can be reached from any other city in at most 9 steps. This number known as the **diameter** of the state space, gives us a better depth limit.

Depth-limited-search can be implemented as a simple modification to the general tree- search algorithm or to the recursive depth-first-search algorithm. The pseudocode for recursive depth- limited-search is shown in Figure.

It can be noted that the above algorithm can terminate with two kinds of failure : the standard *failure* value indicates no solution; the *cutoff*value indicates no solution within the depth limit. Depth-limited search = depth-first search with depth limit l,returns cut off if any path is cut off by depth limit

**function** Depth-Limited-Search( problem, limit) **returns** a solution/fail/cutoff **return**

Recursive-DLS(Make-Node(Initial-State[problem]), problem, limit) **function** Recursive-

DLS(node, problem, limit) returns solution/fail/cutoff cutoff-occurred? false

**if** Goal-Test(problem,State[node]) then **return** Solution(node)

**else if** Depth[node] = limit **then return** cutoff

**else for each** successor **in** Expand(node, problem) **do** result

Recursive-DLS(successor, problem, limit) **if** result = cutoff then cutoff\_occurred?true

**else if** result not = failure **then return** result

**if**cutoff\_occurred? then return cutoff **else return** failure

**Figure 2.9 Recursive implementation of Depth-limited-search**

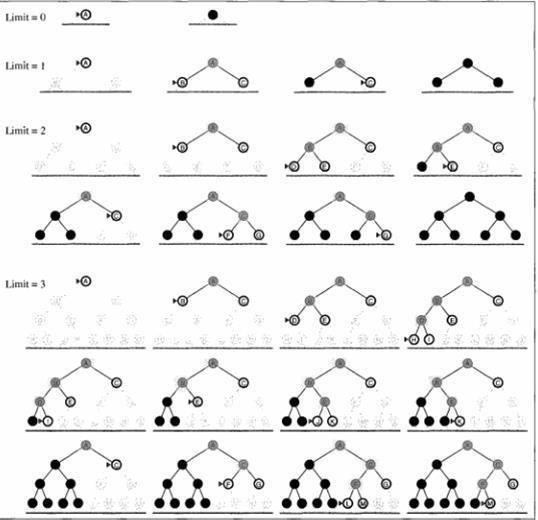
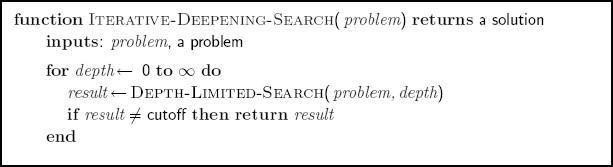
## ITERATIVE DEEPENING DEPTH-FIRST SEARCH

Iterative deepening search (or iterative-deepening-depth-first-search) is a general strategy often used in combination with depth-first-search, that finds the better depth limit. It does this by gradually increasing the limit – first 0,then 1,then 2, and so on – until a goal is found. This will occur when the depth limit reaches d, the depth of the shallowest goal node. The algorithm is shown in Figure.

Iterative deepening combines the benefits of depth-first and breadth-first-search Like depth-first-search, its memory requirements are modest; O(bd) to be precise.

Like Breadth-first-search, it is complete when the branching factor is finite and optimal when the path cost is a non decreasing function of the depth of the node.

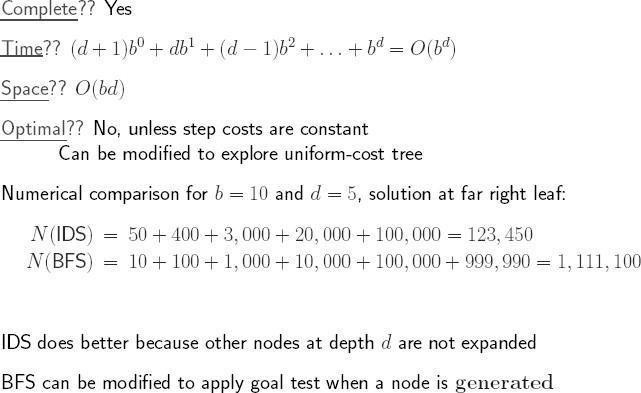
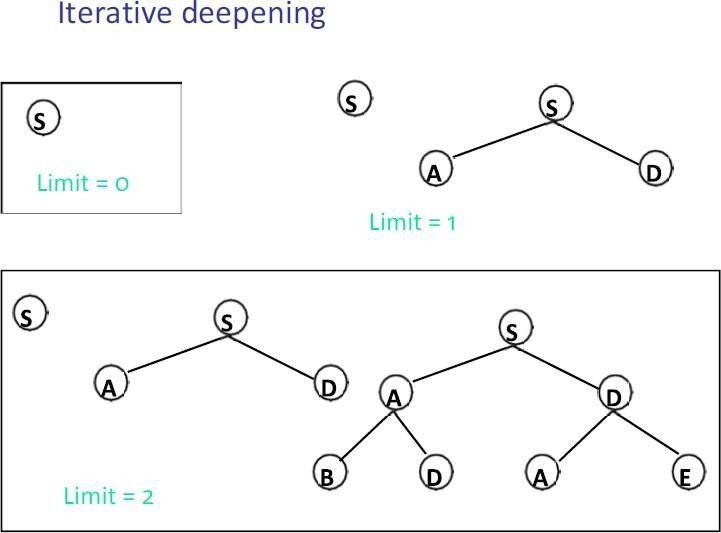
Figure shows the four iterations of ITERATIVE-DEEPENING\_SEARCH on a binary search tree, where the solution is found on the fourth iteration.



**Figure 2.10 The iterative deepening search algorithm, which repeatedly applies depth-limited- search with increasing limits. It terminates when a solution is found or if the depth limited search returns *failure*, meaning that no solution exists.**

**Figure 2.11 Four iterations of iterative deepening search on a binary tree**

Iterative search is not as wasteful as it might seem



**Figure 2.12 Iterative search is not as wasteful as it might seem**

Properties of iterative deepening search

**Figure 2.13 Properties of iterative deepening search**

### Best-first search

**Best-first search** is an instance of general TREE-SEARCH or GRAPH-SEARCH algorithm in which a node is selected for expansion based on an **evaluation function** f(n). The node with lowest evaluation is selected for expansion, because the evaluation measures the distance to the goal.

This can be implemented using a priority-queue, a data structure that will maintain the fringe in ascending order of f-values.

## HEURISTIC FUNCTIONS

A **heuristic function** or simply a **heuristic** is a function that ranks alternatives in various search algorithms at each branching step basing on an available information in order to make a decision which branch is to be followed during a search.

The key component of Best-first search algorithm is a **heuristic function**, denoted by h(n): h(n) = estimated cost of the **cheapest path** from node n to a **goal node**.

For example, in Romania, one might estimate the cost of the cheapest path from Arad to Bucharest via a **straight-line distance** from Arad to Bucharest (Figure 2.19).

Heuristic function are the most common form in which additional knowledge is imparted to the search algorithm.

### Greedy Best-first search

**Greedy best-first search** tries to expand the node that is closest to the goal, on the grounds that this is likely to a solution quickly.

It evaluates the nodes by using the heuristic function f(n) = h(n).

**A\* SEARCH**

**A\* Search** is the most widely used form of best-first search. The evaluation function f(n) is obtained by combining

1. **g(n) =** the cost to reach the node, and
2. **h(n) =** the cost to get from the node to the **goal** :

f(n) = g(n) + h(n).

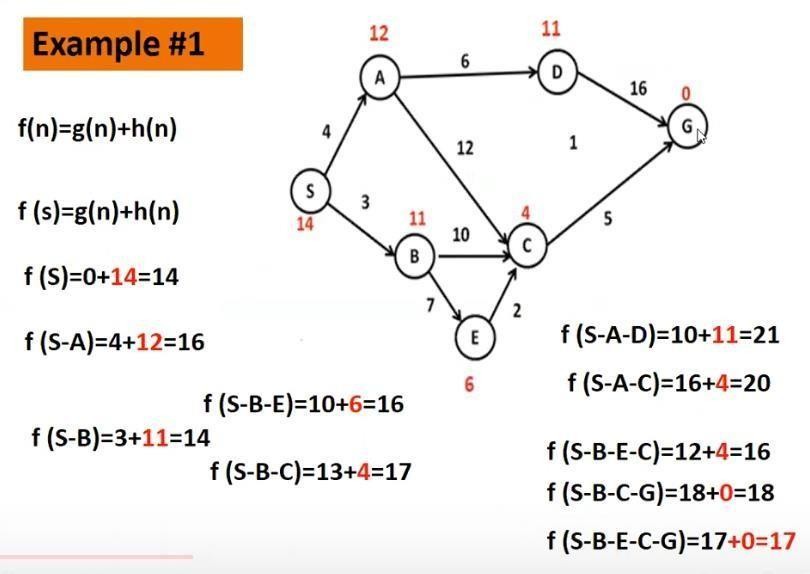
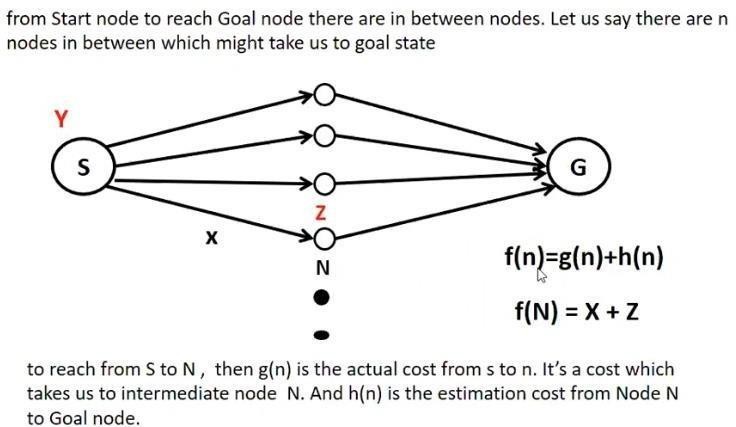
A\* Search is both optimal and complete. A\* is optimal if h(n) is an admissible heuristic. The obvious example of admissible heuristic is the straight-line distance hSLD. It cannot be an overestimate.

A\* Search is optimal if h(n) is an admissible heuristic – that is, provided that h(n) never overestimates the cost to reach the goal.

An obvious example of an admissible heuristic is the straight-line distance hSLD that we used in getting to Bucharest. The progress of an A\* tree search for Bucharest is shown in Figure

The values of ‘g ‘ are computed from the step costs shown in the Romania map(figure).

Also the values of hSLD are given in Figure



**LOCAL SEARCH ALGORITHMS AND OPTIMIZATION PROBLEMS**

**Figure 2.22 Example A\* Search**

**Figure 2.21 A\* Search**

### 1.24

* In many optimization problems, the path to the goal is irrelevant; the goal state itself is the solution
* For example, in the 8-queens problem, what matters is the final configuration of queens, not the order in which they are added.
* In such cases, we can **use local search algorithms.** They operate using a **single current state** (rather than multiple paths) and generally move only to neighbors of that state.
* The important applications of these class of problems are (a) integrated-circuit design,

(b) Factory-floor layout, (c) job-shop scheduling, (d) automatic programming, (e) telecommunications network optimization, (f) Vehicle routing, and (g) portfolio management.

Key advantages of Local Search Algorithms

1. They use very little memory – usually a constant amount; and
2. they can often find reasonable solutions in large or infinite(continuous) state spaces for which systematic algorithms are unsuitable.

## OPTIMIZATION PROBLEMS

In addition to finding goals, local search algorithms are useful for solving pure **optimization problems**, in which the aim is to find the **best state** according to an **objective function**.

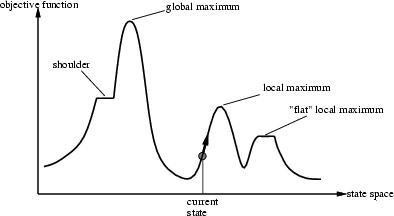
### State Space Landscape

To understand local search, it is better explained using **state space landscape** as shown in Figure.

A landscape has both “**location**” (defined by the state) and “**elevation**” (defined by the value of the heuristic cost function or objective function).

If elevation corresponds to **cost**, then the aim is to find the **lowest valley** – a **global minimum**; if elevation corresponds to an **objective function**, then the aim is to find the **highest peak** – a **global maximum.**

Local search algorithms explore this landscape. A complete local search algorithm always finds a **goal** if one exists; an **optimal** algorithm always finds a **global minimum/maximum**.



**Figure 2.23 A one dimensional state space landscape in which elevation corresponds to the objective function. The aim is to find the global maximum. Hill climbing search modifies the current state to try to improve it, as shown by the arrow. The various topographic features are defined in the text**

### Hill-climbing search

The **hill-climbing** search algorithm as shown in figure, is simply a loop that continually moves in the direction of increasing value – that is, **uphill**. It terminates when it reaches a “**peak**” where no neighbor has a higher value.

**function** HILL-CLIMBING( *problem*) **return** a state that is a local maximum

**input:** *problem*, a problem

**local variables:** *current***, a**

**node.**

*neighbor***, a node.**

*current ←*MAKE-NODE(INITIAL-STATE[*problem*])

**loop do**

*neighbor ←* a highest valued successor of *current*

**if** VALUE [*neighbor*] *≤* VALUE[*current*] **then return** STATE[*current*]

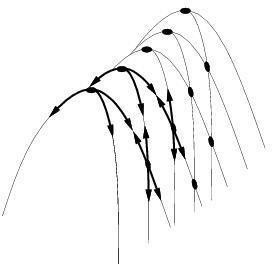
*current ←neighbor*

**Figure 2.24 The hill-climbing search algorithm (steepest ascent version), which is the most basic local search technique. At each step the current node is replaced by the best neighbor; the neighbor with the highest VALUE. If the heuristic cost estimate h is used, we could find the neighbor with the lowest h.**

Hill-climbing is sometimes called greedy local search because it grabs a good neighbor state without thinking ahead about where to go next. Greedy algorithms often perform quite well. **Problems with hill-climbing**

Hill-climbing often gets stuck for the following reasons :

* **Local maxima**: a local maximum is a peak that is higher than each of its neighboring states, but lower than the global maximum. Hill-climbing algorithms that reach the vicinity of a local maximum will be drawn upwards towards the peak, but will then be stuck with nowhere else to go
* **Ridges**: A ridge is shown in Figure 2.10. Ridges results in a sequence of local maximathat is very difficult for greedy algorithms to navigate.
* **Plateaux**: A plateau is an area of the state space landscape where the evaluation function is flat. It can be a flat local maximum, from which no uphill exit exists, or a shoulder, from which it is possible to make progress.



**Figure 2.25 Illustration of why ridges cause difficulties for hill-climbing. The grid of states(dark circles) is superimposed on a ridge rising from left to right, creating a sequence of local maxima that are not directly connected to each other. From each local maximum, all the available options point downhill.**

Hill-climbing variations

* + Stochastic hill-climbing
    - Random selection among the uphill moves.
    - The selection probability can vary with the steepness of the uphill move.
  + First-choice hill-climbing
    - cfr. stochastic hill climbing by generating successors randomly until a better one is found.
  + Random-restart hill-climbing
    - Tries to avoid getting stuck in local maxima.

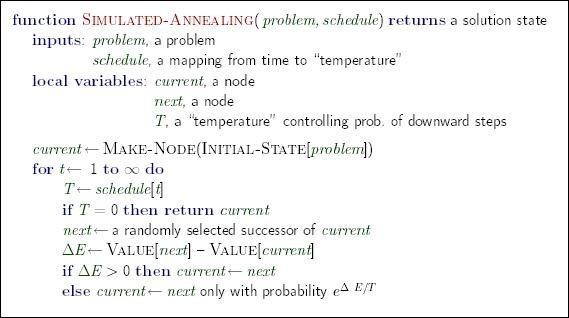
### Simulated annealing search

A hill-climbing algorithm that never makes “downhill” moves towards states with lower value (or higher cost) is guaranteed to be incomplete, because it can stuck on a local maximum. In contrast, a purely random walk –that is, moving to a successor choosen uniformly at random from the set of successors – is complete, but extremely inefficient.

Simulated annealing is an algorithm that combines hill-climbing with a random walk in someway that yields both efficiency and completeness.

Figure shows simulated annealing algorithm. It is quite similar to hill climbing. Instead of picking the best move, however, it picks the random move. If the move improves the situation, it is always accepted. Otherwise, the algorithm accepts the move with some probability less than 1. The probability decreases exponentially with the “badness” of the move – the amount E by which the evaluation is worsened.

Simulated annealing was first used extensively to solve VLSI layout problems in the early 1980s. It has been applied widely to factory scheduling and other large-scale optimization tasks.



**Figure 2.26 The simulated annealing search algorithm, a version of stochastic hill climbing where some downhill moves are allowed.**

## CONSTRAINT SATISFACTION PROBLEMS(CSP)

A **Constraint Satisfaction Problem**(or CSP) is defined by a set of **variables,**X1,X2,….Xn, and a set of constraints C1,C2,…,Cm. Each variable Xi has a nonempty **domain** D,of possible **values**.

Each constraint Ci involves some subset of variables and specifies the allowable combinations of values for that subset.

A **State** of the problem is defined by an **assignment** of values to some or all of the variables,{Xi = vi,Xj = vj,…}. An assignment that does not violate any constraints is called a **consistent** or **legal assignment.** A complete assignment is one in which every variable is mentioned, and a **solution** to a CSP is a complete assignment that satisfies all the constraints.

Some CSPs also require a solution that maximizes an **objective function**.

### Example for Constraint Satisfaction Problem

Figure shows the map of Australia showing each of its states and territories. We are given the task of coloring each region either red, green, or blue in such a way that the neighboring regions have the same color. To formulate this as CSP, we define the variable to be the regions

:WA,NT,Q,NSW,V,SA, and T. The domain of each variable is the set

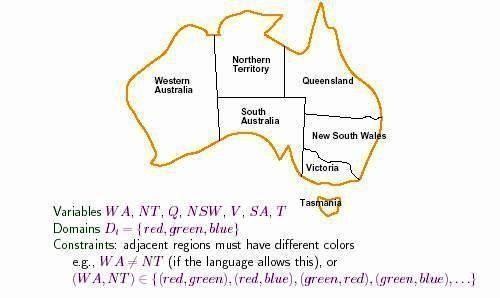
{red,green,blue}.The constraints require neighboring regions to have distinct colors; for example, the allowable combinations for WA and NT are the pairs

{(red,green),(red,blue),(green,red),(green,blue),(blue,red),(blue,green)}.

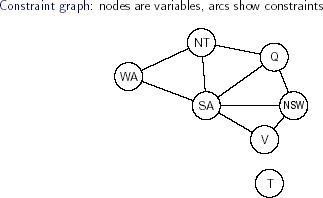
The constraint can also be represented more succinctly as the inequality WA not = NT, provided the constraint satisfaction algorithm has some way to evaluate such expressions.) There are many possible solutions such as

{ WA = red, NT = green, Q = red, NSW = green, V = red,SA = blue,T = red}.

It is helpful to visualize a CSP as a constraint graph, as shown in Figure 2.29. The nodes of the graph corresponds to variables of the problem and the arcs correspond to constraints.



**Figure 2.29 Principle states and territories of Australia. Coloring this map can be viewed as a constraint satisfaction problem. The goal is to assign colors to each region so that no neighboring regions have the same color.**



### Figure 2.30 Mapping Problem

CSP can be viewed as a standard search problem as follows:

* **Initial state**: the empty assignment {},in which all variables are unassigned.
* **Successor function**: a value can be assigned to any unassigned variable, provided that it does not conflict with previously assigned variables.
* **Goal test**: the current assignment is complete.
* **Path cost**: a constant cost(E.g.,1) for every step.

Every solution must be a complete assignment and therefore appears at depth n if there are n variables.

Depth first search algorithms are popular for CSPs

### Varieties of CSPs

1. **Discrete variables Finite domains**

The simplest kind of CSP involves variables that are **discrete** and have **finite domains.** Map coloring problems are of this kind. The 8-queens problem can also be viewed as finite- domain

CSP, where the variables Q1,Q2,…..Q8 are the positions each queen in columns 1,….8 and each variable has the domain {1,2,3,4,5,6,7,8}. If the maximum domain size of any

variable in a CSP is d, then the number of possible complete assignments is O(dn) – that is, exponential in the number of variables. Finite domain CSPs include **Boolean CSPs**, whose variables can be either *true* or *false*. **Infinite domains**

Discrete variables can also have **infinite domains** – for example, the set of integers or the set of strings. With infinite domains, it is no longer possible to describe constraints by enumerating all allowed combination of values. Instead a constraint language of algebric inequalities such as Startjob1 + 5 <= Startjob3.

### CSPs with continuous domains

CSPs with continuous domains are very common in real world. For example in operation research field, the scheduling of experiments on the Hubble Telescope requires very precise timing of observations; the start and finish of each observation and manoeuvre are continuous-valued variables that must obey a variety of astronomical, precedence and power constraints. The best known category of continuous-domain CSPs is that of **linear programming** problems, where the constraints must be linear inequalities forming a *convex* region. Linear programming problems can be solved in time polynomial in the number of variables.

### Varieties of constraints

1. **unary constraints** involve a single variable.

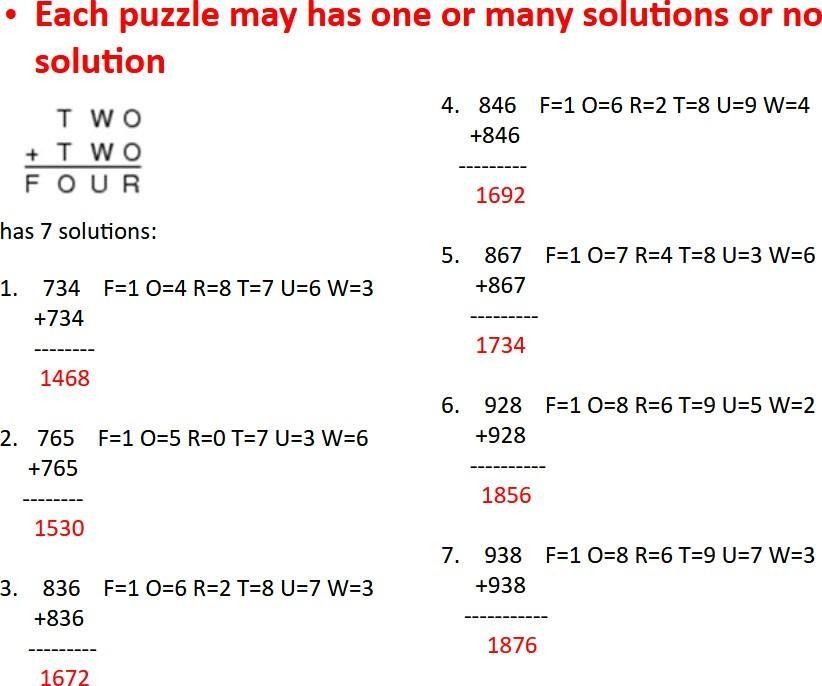
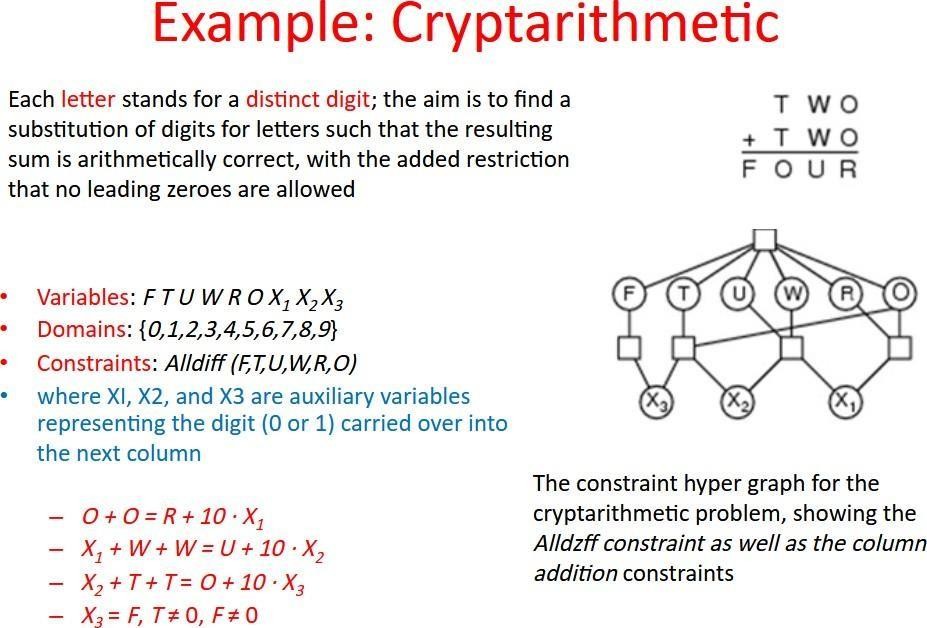
Example : SA # green

1. Binary constraints involve paris of variables.

Example : SA # WA

1. Higher order constraints involve 3 or more variables. Example :cryptarithmetic puzzles.

**Figure 2.31 cryptarithmetic puzzles**.

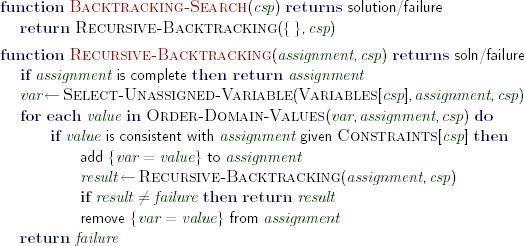
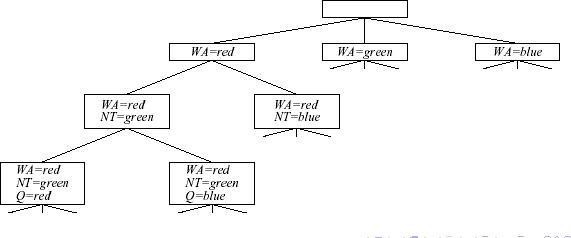
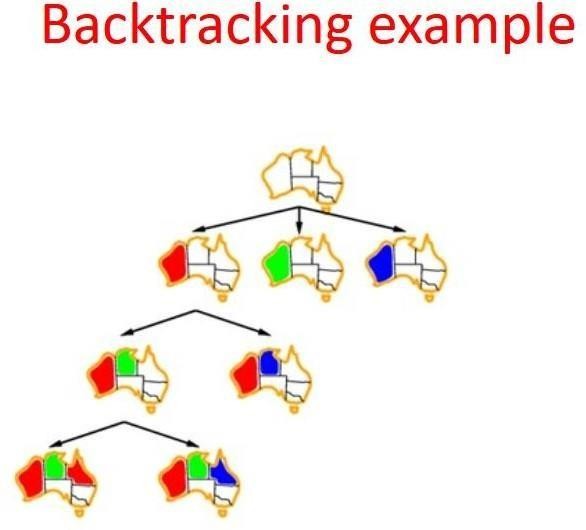


**Figure 2.33 Numerical Solution**

**Figure 2.32 Cryptarithmetic puzzles-Solution**

### Backtracking Search for CSPs

The term **backtracking search** is used for depth-first search that chooses values for one variable at a time and backtracks when a variable has no legal values left to assign. The algorithm is shown in figure



**Figure 2.34 A simple backtracking algorithm for constraint satisfaction problem. The algorithm is modeled on the recursive depth-first search**

**Figure 2.34 Part of the search tree generated by simple backtracking for the map- coloring problem**

**Figure 2.35 Part of search tree generated by simple backtracking for the map**

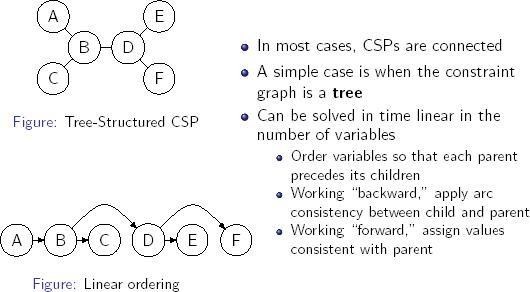
**coloring problem.**

### Forward checking

One way to make better use of constraints during search is called forward checking. Whenever a variable X is assigned, the forward checking process looks at each unassigned variable Y that is connected to X by a constraint and deletes from Y ’s domain any value that is inconsistent with the value chosen for X. Figure 5.6 shows the progress of a map-coloring search with forward checking.

**k-Consistency**

**Tree-Structured CSPs**



**Figure 2.40 Tree-Structured CSPs**

## 1.29 ADVERSARIAL SEARCH

Competitive environments, in which the agent’s goals are in conflict, give rise to

**adversarial search** problems – often known as **games. Games**

Mathematical **Game Theory**, a branch of economics, views any **multiagent environment** as a **game** provided that the impact of each agent on the other is “significant”, regardless of whether the agents are cooperative or competitive. In, **AI**, “games” are deterministic, turn-taking, two-player, zero-sum games of perfect information. This means deterministic, fully observable environments in which there are two agents whose actions must alternate and in which the **utility values** at the end of the game are always equal and opposite. For example, if one player wins the game of chess(+1),the other player necessarily loses(-1). It is this opposition between the agents’ utility functions that makes the situation **adversarial.**

### Formal Definition of Game

We will consider games with two players, whom we will call **MAX** and **MIN**. MAX moves first, and then they take turns moving until the game is over. At the end of the game, points are awarded to the winning player and penalties are given to the loser. A **game** can be formally defined as a **search problem** with the following components:

* The **initial state**, which includes the board position and identifies the player to move.
* A **successor function**, which returns a list of (*move, state*) pairs, each indicating a legal move and the resulting state.
* **terminal test**, which describes when the game is over. States where the game has ended are called **terminal states**.
* A **utility function** (also called an objective function or payoff function), which give a numeric value for the terminal states. In chess, the outcome is a win, loss, or draw, with values+1,-1, or 0. he payoffs in backgammon range from +192 to -192.

### Game Tree

The **initial state** and **legal moves** for each side define the **game tree** for the game.

### Optimal Decisions in Games

In normal search problem, the **optimal solution** would be a sequence of move leading to a **goal state** – a terminal state that is a win. In a game, on the other hand, MIN has something

to say about it, MAX therefore must find a contingent **strategy**, which specifies MAX’s move in the **initial state**, then MAX’s moves in the states resulting from every possible response by MIN, then MAX’s moves in the states resulting from every possible response by MIN those moves, and so on. An **optimal strategy** leads to outcomes at least as good as any other strategy when one is playing an infallible opponent.