Unit - 1

Introduction

1.1 Introduction to Artificial Intelligence

What is Artificial Intelligence?

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 pre-program 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:

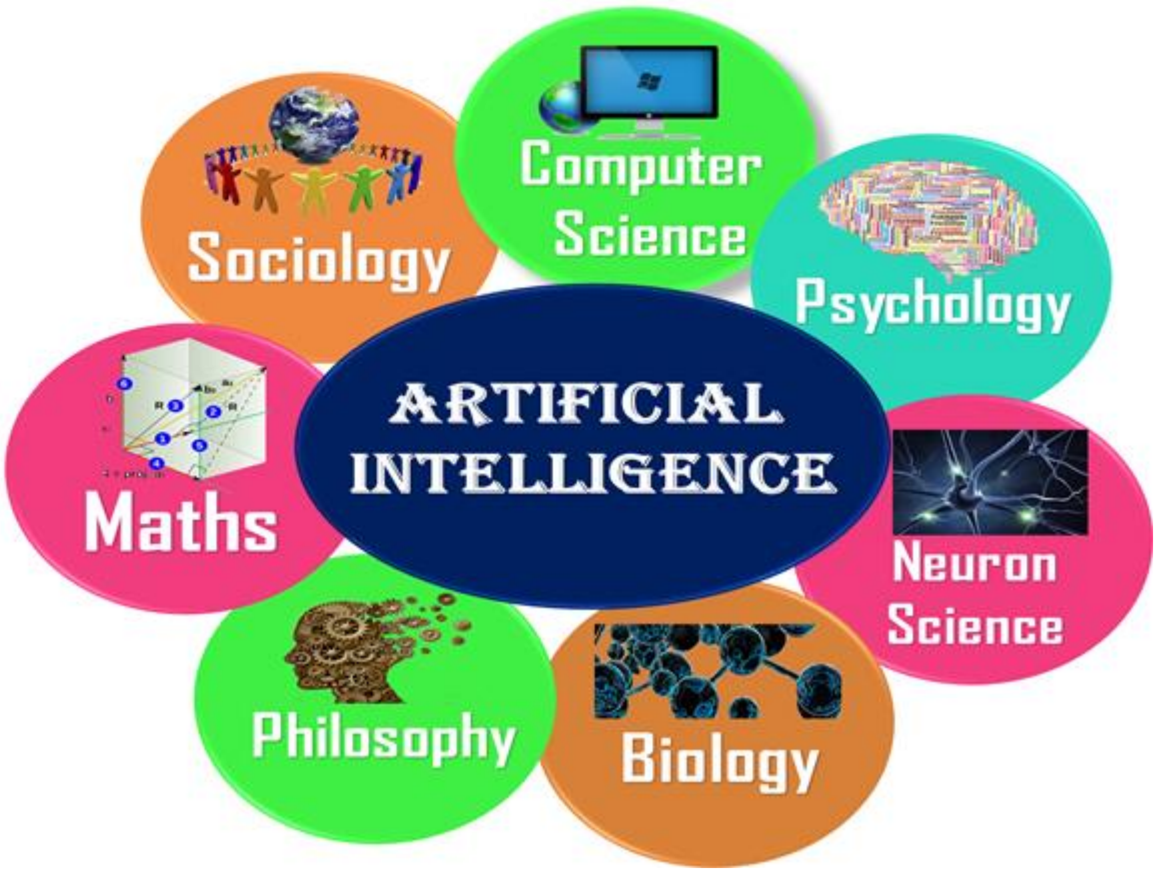
1. Replicate human intelligence
2. Solve Knowledge-intensive tasks
3. An intelligent connection of perception and action
4. Building a machine which can perform tasks that requires human intelligence such as:
   - Proving a theorem
   - Playing chess
   - Plan some surgical operation
   - Driving a car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

What Comprises 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 the same 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.

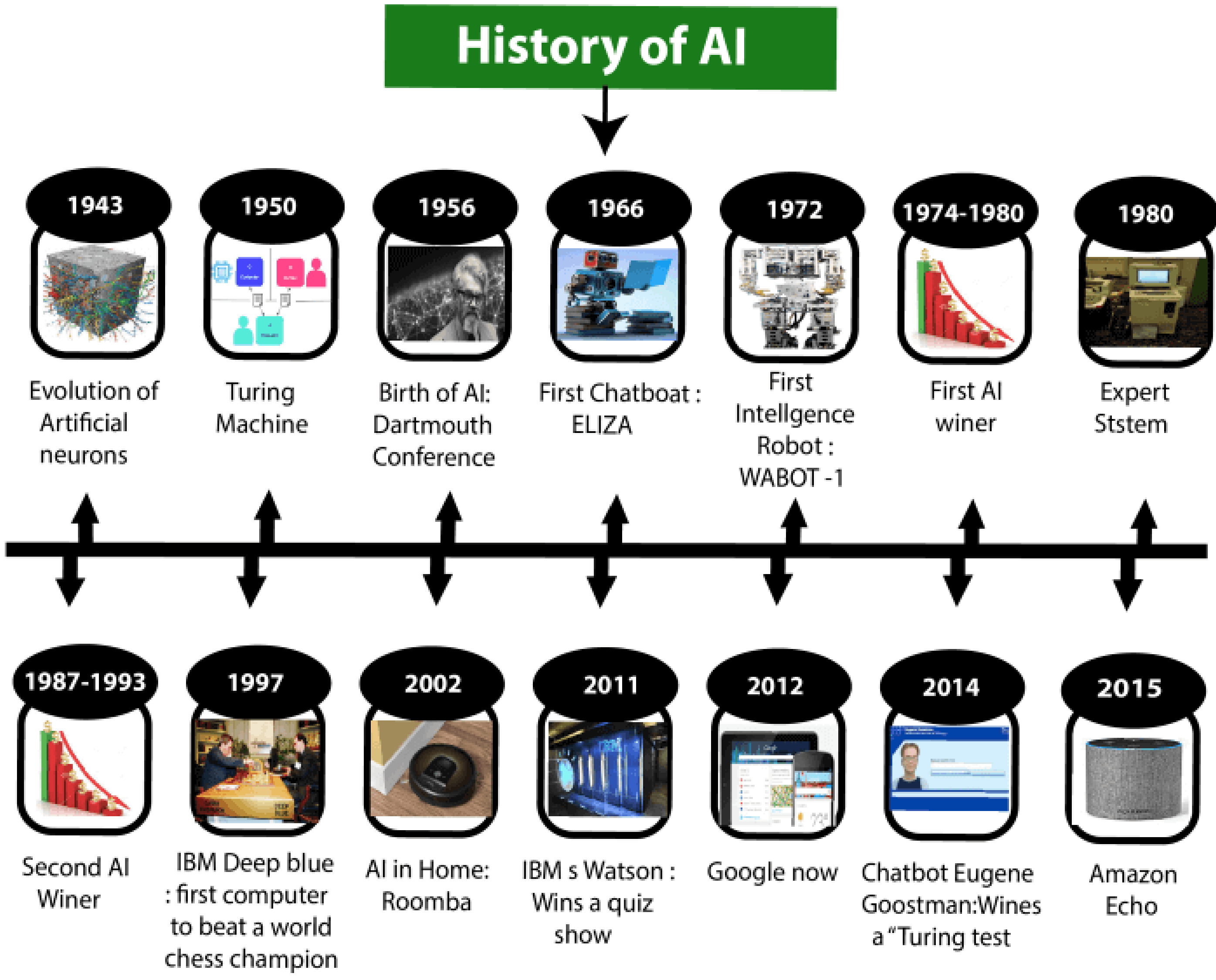
Key takeaway

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.

1.2 Foundations and History of Artificial Intelligence

Artificial Intelligence is not a new word and not a new technology for researchers. This technology is much older than you would imagine. Even there are the myths of Mechanical men in Ancient Greek and Egyptian

Myths. Following are some milestones in the history of AI which defines the journey from the AI generation to till date development.


History of AI
1943
Evolution of Artificial neurons
1950
Turing Machine
1956
Birth of AI: Dartmouth Conference
1966
First Chatboat : ELIZA
1972
First Intellgence Robot : WABOT -1
1974-1980
First AI winer
1980
Expert Ststem
1987-1993
Second AI Winer
1997
IBM Deep blue : first computer to beat a world chess champion
2002
AI in Home: Roomba
2011
IBM s Watson : Wins a quiz show
2012
Google now
2014
Chatbot Eugene Goostman:Wines a "Turing test
2015
Amazon Echo

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.

Now AI has developed to a remarkable level. The concept of Deep learning, big data, and data science are now trending like a boom. Nowadays companies like Google, Facebook, IBM, and Amazon are working with AI and creating amazing devices. The future of Artificial Intelligence is inspiring and will come with high intelligence.

Key takeaway

Artificial Intelligence is not a new word and not a new technology for researchers. This technology is much older than you would imagine. Even there are the myths of Mechanical men in Ancient Greek and Egyptian Myths. Following are some milestones in the history of AI which defines the journey from the AI generation to till date development.

1.3 State of the Art

Researchers now have access to new tools that can help them achieve critical objectives, and these technologies are excellent starting points in and of themselves. The following are some specific domains that have been achieved in recent years:

- Machine learning;
- Reinforcement learning;
- Deep learning;
- Natural language processing.

Machine learning

Machine learning is a subtype of artificial intelligence that employs statistical approaches to enable machines to absorb data without being explicitly instructed to do so. This is referred to as 'training' a'model' with a learning 'algorithm,' which improves performance on a given task over time. Researchers have been inspired to push further on the accelerator as a result of their triumphs in this field.

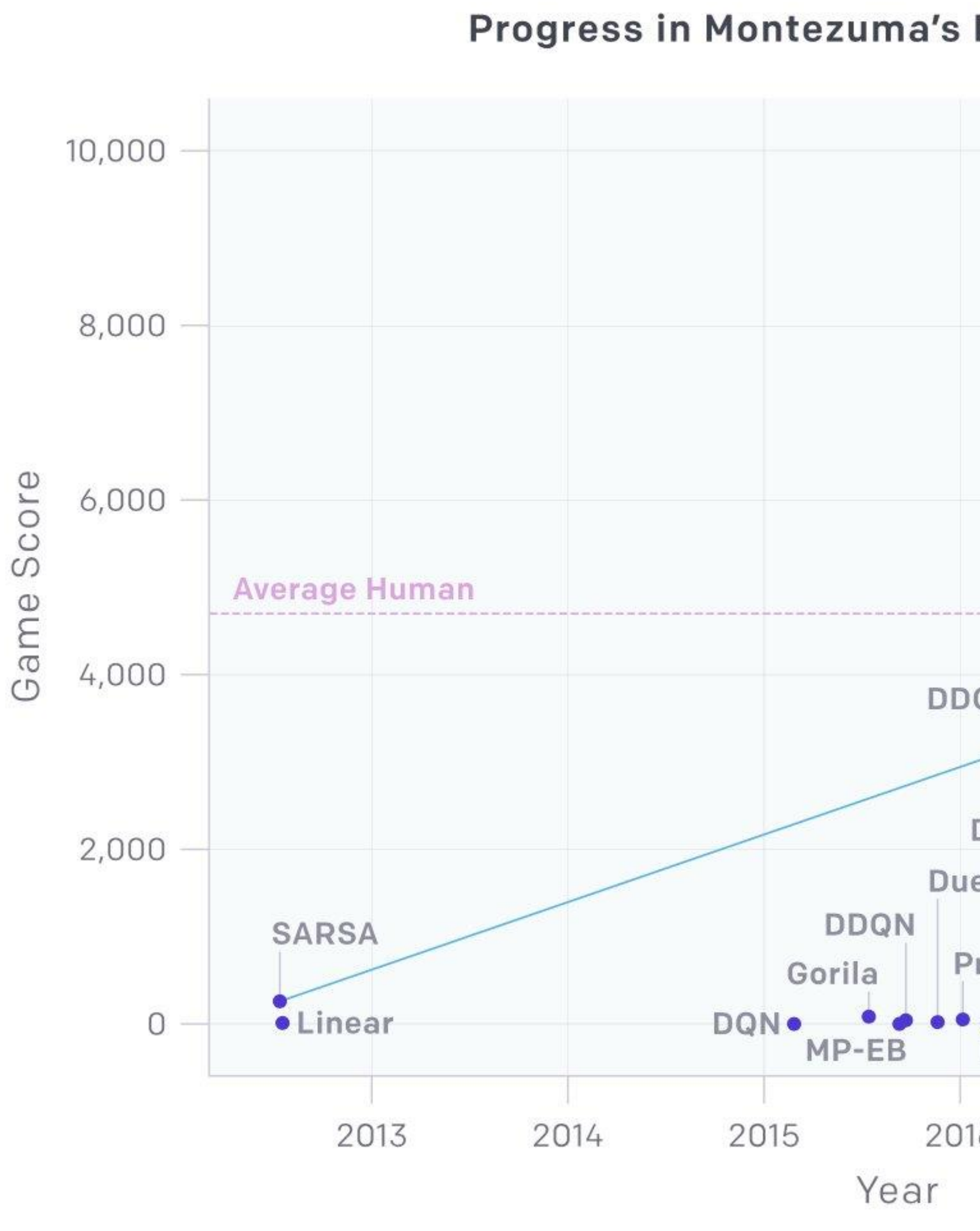
The capacity of machines to learn how to create molecules is one of these achievements. It was feasible to train the computers with roughly 12.4 million reactions using a system made up of three neural networks and a search tree technique dubbed 'Monte Carlo,' which solved various retrosynthetic studies. This process is far faster than the current one, which involves computerised molecular synthesis planning. In fact, it solves more than 80% of a single molecular test, with a maximum time restriction of 5 seconds as a target for each unique molecule being examined.

Research on improving techniques like hyperparameters and neural networks, which are fixed parameters provided to computers as beginning values for learning, is also continuing. By reducing the complexity and size of the calculation, new algorithms can help to maximise network performance. LEAF (Learning Evolutionary AI Framework) is an example of this, since it uses these methods to precisely optimise hyperparameters and network designs by welding together smaller, more effective networks.

Reinforcement Learning (RL)

Reinforcement learning is a branch of machine learning that deals with software agents that learn 'goal-oriented' behaviour by attempting and failing in settings that reward them for their actions (called 'Policy') toward accomplishing the goals.

This is the field that has piqued the interest of researchers the most in the recent decade. In 2018, OpenAI, a non-profit artificial intelligence research organisation dedicated to promoting and creating friendly AI, achieved significant results in the game Montezuma's Revenge. A technique called Random Network Distillation (RND) was used to achieve superhuman performance by encouraging the RL agent to explore unanticipated states. The graph below demonstrates how far this strategy outperformed the game's other AI algorithms.



Progress in Montezuma's I
Game Score
10,000
8,000
6,000
4,000
2,000
0
Average Human
SARSA
Linear
DQN
Gorila
MP-EB
DDQN
Due
Pr
DD
D
2013
2014
2015
201
Year

This is only one of a number of examples of findings received in 2019. DeepMind's AlphaStar is another AI worth considering. It used multi-agent algorithm training to defeat the five-time world champion in the real-time strategy game StarCraft 2. It began by forcing agents to compete against one another, allowing it to learn about the vast strategic space. Later, a new agent was created that integrated the greatest methods that people had devised. Multiple agents that independently learnt and operated together to compete against one another achieved a level of performance equal to humans in Quake 3 Arena.

Deep Learning

Deep learning, another type of machine learning, is inspired by the action of neurons in the brain to learn how to discern complicated patterns from taught data. This is because algorithms, mostly statistical calculations, are used. The term 'deep' alludes to the huge number of levels of neurons that ML models at the same time, allowing for the acquisition of rich data representations and performance increases.

The year 2019 marked a watershed moment for deep learning and its applications in a variety of fields, particularly medicine. For example, a technique known as "two phases" has resulted in expert-level diagnosis and therapy prescriptions for a variety of eye illnesses. The first stage used a computerised 3D scanner to reconstruct an eye tissue map, and the second stage used this map to make predictions about the severity of the condition. A deep learning model that was employed in 54 thousand ECG traces is another example. It can identify 12 different types of arrhythmias.

Even more significant is what the researchers hope future research will reveal, namely the possibility of recovering speech in paralysed patients and movement in quadriplegics.

In the first case, Columbia University researchers were able to synthesise voice with a vocoder employing ways of analysing the brain activity of five epileptic patients. Although the system has achieved 75 percent accuracy, a deep learning model has shown a 65 percent improvement in voice comprehensibility.

In the second, even more astonishing example, researchers implanted a microelectrode in a paraplegic patient's hand and arm portion of the left side of the primary motor cortex. They told the neural network to use the voltage signals inside the patient's skull to forecast what the patient's arm movement intentions would be. Without monitoring, the patient was able to maintain a high-accuracy resuscitation of his paralysed forearm with functional electrical stimulation for almost a year.

Natural Language Processing (NLP)

Natural language processing (NLP) is the process by which computers learn to interpret, comprehend, and alter textual material. With Google AI's BERT and Transformer, Allen Institute's ELMo, OpenAI's Transformer, Ruder and Howard's ULMFit, and Microsoft's MT-DNN, 2019 was a banner year for NPL. All of these studies have demonstrated that pre-taught language models can increase performance on a wide range of NLP tasks.

Take the OpenAI GPT-2 pre-educated language model for example:

INPUT: Miley Cyrus was busted shoplifting on Hollywood Boulevard from Abercrombie and Fitch.

OUTPUT (written totally by machine on the second attempt): Security guards were recorded on camera escorting the 19-year-old singer out of the store. The singer was dressed in a black hoodie with the labels 'Blurred lines' and 'Fashion Police' on the front and back, respectively. The video can be found by scrolling below.

Miley Cyrus was busted shoplifting today on Hollywood Boulevard from Abercrombie & Fitch (pictured).

1.4 Risks and Benefits of AI

Artificial intelligence is dreaded by many experts and ordinary individuals alike, despite being heralded as a gift for humanity by tech pundits. This dread has been shown on film countless times in the shape of dystopian futures generated by AI computers that have taken over the world. The Matrix and the Terminator are two of the most well-known examples.

AI is Unsustainable

Intelligent machines are known for their great computational power, which is provided by a slew of processors. Selenium, a rare earth metal, is a major component of these computer chips. Furthermore, such robots' batteries are made of Lithium, another rare element found in the earth's crust. Increased mining of these materials is rapidly and irreversibly harming our ecosystem. Furthermore, they consume a large amount of power to operate, putting a strain on our power plants and, once again, damaging the environment.

Lesser Jobs

Machines, without a doubt, perform routine and repetitious activities far better than people. Many firms would prefer machines replace humans in order to boost their profits, lowering the number of employment available for humans.

A threat to Humanity(?)

Elon Musk is widely regarded as one of the most intelligent people working on artificial intelligence today. He has also stated openly that artificial intelligence is the greatest future threat to human civilisation. This suggests that the dismal future depicted in science fiction films is not implausible. In addition, Stephen Hawking has long expressed his opposition to AI advancements.

The most serious danger connected with AI is that computers will develop consciousness and turn against humans if they become self-aware.

Benefits of AI

Artificial Intelligence is a computer program's ability to learn and think. Everything that includes a programme doing something that we would ordinarily associate with human intelligence is termed artificial intelligence.

Artificial intelligence applications offer huge benefits and have the potential to disrupt any industry. Let's take a look at a few of them.

1) Reduction in Human Error:

Because humans make mistakes from time to time, the term "human error" was coined. Computers, on the other hand, do not make these errors if they are correctly programmed. Artificial intelligence makes choices based on previously obtained data and a set of algorithms. As a result, errors are decreased, and the prospect of achieving better precision and accuracy is increased.

For example, AI has removed the majority of human mistake in weather forecasting.

2) Takes risks instead of Humans:

One of the most significant advantages of artificial intelligence is this. By constructing an AI Robot that can do the risky tasks for us, we can transcend many of humanity's risky limits. It can be utilised effectively in every type of natural or man-made disaster, whether it is going to Mars, defusing a bomb, exploring the deepest regions of the oceans, mining for coal and oil.

Have you heard about the explosion at the Chernobyl nuclear power facility in Ukraine? There were no AI-powered robots available at the time to assist us in minimising the effects of radiation by controlling the fire early on, as any human who came close to the core died in minutes. They eventually used helicopters to drop sand and boron from a safe distance.

AI Robots can be utilised in situations when human intervention is risky.

3) Available 24x7:

Without breaks, an average human will labour for 4–6 hours every day. Humans are created in such a way that they can take time off to replenish themselves and prepare for a new day at work, and they even have weekly off days to keep their professional and home lives separate. But, unlike humans, we can use AI to make machines work 24 hours a day, seven days a week with no breaks, and they don't get bored.

For example, educational institutions and helpline centres receive a large number of requests and difficulties that AI can successfully address.

4) Helping in Repetitive Jobs:

We will be doing a lot of repetitious labour in our day-to-day work, such as writing thank-you emails, double-checking documents for flaws, and so on. We can use artificial intelligence to efficiently automate these monotonous operations and even remove "boring" duties from humans' schedules, allowing them to be more creative.

For example, at banks, we frequently see numerous document verifications in order to obtain a loan, which is a time-consuming task for the bank's owner. The owner can use AI Cognitive Automation to speed up the process of document verification, which will benefit both the customers and the owner.

5) Digital Assistance:

Digital assistants are used by some of the most modern enterprises to engage with people, reducing the requirement for human personnel. Many websites also use digital assistants to supply things that users seek. We can discuss what we're searching for with them. Some chatbots are created in such a way that it's difficult to tell whether we're conversing with a machine or a human.

For example, we all know that businesses have a customer service team that is responsible for answering customers' questions and concerns. Organizations can use AI to create a voice bot or a chatbot that can assist customers with all of their questions. Many firms have already begun to implement them on their websites and mobile applications.

6) Faster Decisions:

We can make computers make decisions and carry out activities faster than humans by combining AI with other technologies. While a human will consider numerous aspects, both emotionally and practically, before making a decision, an AI-powered machine will focus on what it has been designed to do and will produce results more quickly.

For instance, we've all played Chess games on Windows. Because of the AI in the game, beating the CPU in hard mode is practically impossible. According to the algorithms utilised, it will take the best feasible step in the shortest amount of time.

7) Daily Applications:

Apple's Siri, Microsoft's Cortana, and Google's OK Google are all commonplace in our daily lives, whether it's for finding a location, taking a selfie, making a phone call, or responding to an email.

For example, when we wanted to go somewhere 20 years ago, we used to ask someone who had already been there for instructions. All we have to do now is ask Google, "OK Google, where is Visakhapatnam?" It will show you the location of Visakhapatnam on a Google map as well as the best route between you and Visakhapatnam.

8) New Inventions:

Many technologies in practically every domain are powered by AI, which will aid humans in solving the majority of complicated problems.

For instance, using powerful AI-based technologies, clinicians can now identify breast cancer in women at an early stage.

1.5 Intelligent Agents, Agents and Environments

Agents in Artificial Intelligence

An AI system can be defined as the study of the rational agent and its environment. The agents sense the environment through sensors and act on their environment through actuators. An AI agent can have mental properties such as knowledge, belief, intention, etc.

What is an Agent?

An agent can be anything that perceive its environment through sensors and act upon that environment through actuators. An Agent runs in the cycle of **perceiving**, **thinking**, and **acting**. 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.

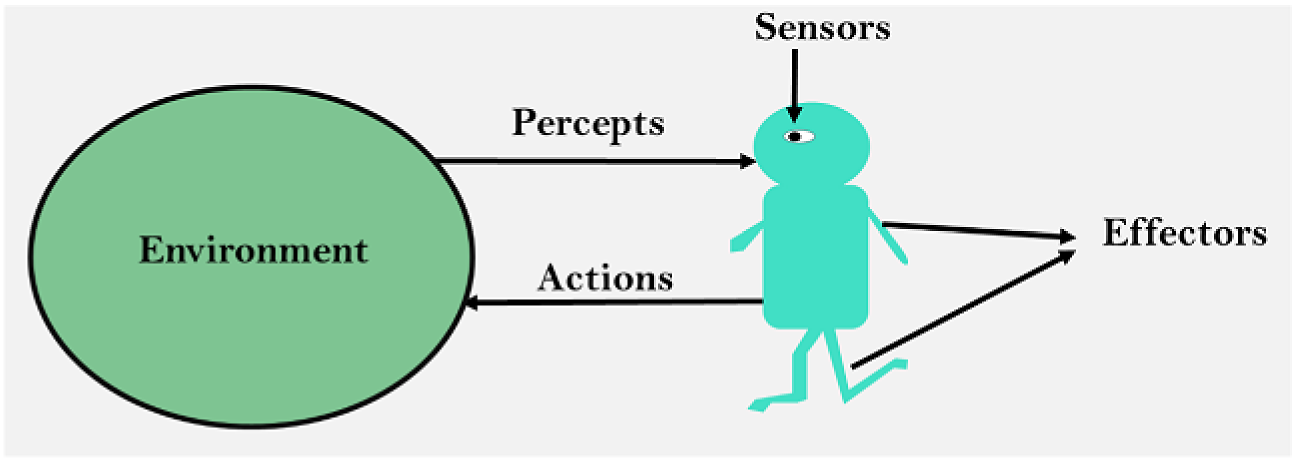
Hence the world around us is full of agents such as thermostat, cell phone, camera, and even we are also agents.

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.



Intelligent Agents:

An intelligent agent is an autonomous entity which act upon an environment using sensors and actuators for achieving goals. An intelligent agent may learn from the environment to achieve their goals. A thermostat is an example of an intelligent agent.

Following are the main four rules for an AI agent:

- **Rule 1:** An AI agent must have the ability to perceive the environment.
- **Rule 2:** The observation must be used to make decisions.
- **Rule 3:** Decision should result in an action.
- **Rule 4:** The action taken by an AI agent must be a rational action.

Rational Agent:

A rational agent is an agent which has clear preference, models uncertainty, and acts in a way to maximize its performance measure with all possible actions.

A rational agent is said to perform the right things. AI is about creating rational agents to use for game theory and decision theory for various real-world scenarios.

For an AI agent, the rational action is most important because in AI reinforcement learning algorithm, for each best possible action, agent gets the positive reward and for each wrong action, an agent gets a negative reward.

Note: Rational agents in AI are very similar to intelligent agents.

Rationality:

The rationality of an agent is measured by its performance measure. Rationality can be judged on the basis of following points:

- Performance measure which defines the success criterion.
- Agent prior knowledge of its environment.
- Best possible actions that an agent can perform.
- The sequence of percepts.

Note: Rationality differs from Omniscience because an Omniscient agent knows the actual outcome of its action and act accordingly, which is not possible in reality.

Structure of an AI Agent

The task of AI is to design an agent program which implements the agent function. The structure of an intelligent agent is a combination of architecture and agent program. It can be viewed as:

Agent = Architecture + Agent program

Following are the main three terms involved in the structure of an AI agent:

Architecture: Architecture is machinery that an AI agent executes on.

Agent Function: Agent function is used to map a percept to an action.

f:P* → A

Agent program: Agent program is an implementation of agent function. An agent program executes on the physical architecture to produce function f.

PEAS Representation

PEAS is a type of model on which an AI agent works upon. When we define an AI agent or rational agent, then we can group its properties under PEAS representation model. It is made up of four words:

- **P:** Performance measure
- **E:** Environment
- **A:** Actuators
- **S:** Sensors

Here performance measure is the objective for the success of an agent's behavior.

PEAS for self-driving cars:

Let's suppose a self-driving car then PEAS representation will be:

Performance: Safety, time, legal drive, comfort

Environment: Roads, other vehicles, road signs, pedestrian

Actuators: Steering, accelerator, brake, signal, horn

Sensors: Camera, GPS, speedometer, odometer, accelerometer, sonar.

Example of Agents with their PEAS representation

Agent	**Performance measure**	**Environment**	**Actuators**	**Sensors**
1. Medical Diagnose	• Healthy patient • Minimized cost	• Patient • Hospital • Staff	• Tests • Treatments	Keyboard (Entry of symptoms)
2. Vacuum Cleaner	• Cleanness • Efficiency • Battery life • Security	• Room • Table • Wood floor • Carpet • Various obstacles	• Wheels • Brushes • Vacuum Extractor	• Camera • Dirt detection sensor • Cliff sensor • Bump Sensor • Infrared Wall Sensor

3. Part - picking Robot	● Percentage of parts in correct bins.	● Conveyor belt with parts, ● Bins	● Jointed Arms ● Hand	● Camera ● Joint angle sensors.

1.6 Good Behavior: Concept of Rationality, Nature of Environments

The state of being reasonable, sensible, and having a good sense of judgment is known as rationality.

Rationality is concerned with the actions and consequences that can be foreseen based on the agent's views. Taking actions with the objective of obtaining useful knowledge is a fundamental part of reason.

The rationality of an agent is determined by its performance metric. To determine reasonableness, utilize the following criteria:

- The success criterion is defined by a performance metric.
- The agent has prior knowledge of its surroundings.
- The most effective activities that an agent can take.
- The order in which percepts appear.

The Nature of Environments

Some programs operate in an entirely artificial environment, relying solely on keyboard input, databases, computer file systems, and character output on a screen.

On the other hand, some software agents (software robots or softbots) exist in rich, unlimited softbots domains. The simulator's environment is incredibly detailed and complex. The software agent must choose among a vast number of possibilities in real time. In both a real and an artificial setting, a softbot examines a customer's internet interests and offers appealing things to them.

The Turing Test is the most well-known artificial scenario, in which one real and other artificial agents are put to the test on an equal footing. Because a software agent cannot perform as effectively as a human, this is a challenging environment to work in.

Turing test

The Turing Test can be used to determine whether a system's intelligent behavior is successful.

Two humans will participate in the test, as well as a machine that will be checked. One of the two individuals is assigned to the role of tester. They're all in separate rooms. The tester has no way of knowing who is human and who is not. He types the questions and transmits them to both intelligences, who react with typed answers.

The goal of this test is to deceive the tester. The machine is deemed to be intelligent if the tester is unable to distinguish the machine's response from the human response.

Key takeaway

An AI system can be defined as the study of the rational agent and its environment. The agents sense the environment through sensors and act on their environment through actuators. An AI agent can have mental properties such as knowledge, belief, intention, etc.

1.7 Structure of Agents

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.

Characteristics of intelligent agents

Intelligent agents have the following distinguishing characteristics:

- They have some level of autonomy that allows them to perform certain tasks on their own.
- They have a learning ability that enables them to learn even as tasks are carried out.
- They can interact with other entities such as agents, humans, and systems.
- New rules can be accommodated by intelligent agents incrementally.
- They exhibit goal-oriented habits.
- They are knowledge-based. They use knowledge regarding communications, processes, and entities.

The structure of intelligent agents

The IA structure consists of three main parts: architecture, agent function, and agent program.

1. **Architecture:** This refers to machinery or devices that consists of actuators and sensors. The intelligent agent executes on this machinery. Examples include a personal computer, a car, or a camera.
2. **Agent function:** This is a function in which actions are mapped from a certain percept sequence. Percept sequence refers to a history of what the intelligent agent has perceived.

3. **Agent program:** This is an implementation or execution of the agent function. The agent function is produced through the agent program's execution on the physical architecture.

Categories of intelligent agents

There are 5 main categories of intelligent agents. The grouping of these agents is based on their capabilities and level of perceived intelligence.

Simple reflex agents

These agents perform actions using the current percept, rather than the percept history. The condition-action rule is used as the basis for the agent function. In this category, a fully observable environment is ideal for the success of the agent function.

Model-based reflex agents

Unlike simple reflex agents, model-based reflex agents consider the percept history in their actions. The agent function can still work well even in an environment that is not fully observable. These agents use an internal model that determines the percept history and effect of actions. They reflect on certain aspects of the present state that have been unobserved.

Goal-based agents

These agents have higher capabilities than model-based reflex agents. Goal-based agents use goal information to describe desirable capabilities. This allows them to choose among various possibilities. These agents select the best action that enhances the attainment of the goal.

Utility-based agents

These agents make choices based on utility. They are more advanced than goal-based agents because of an extra component of utility measurement. Using a utility function, a state is mapped against a certain measure of utility. A rational agent selects the action that optimizes the expected utility of the outcome.

Learning agents

These are agents that have the capability of learning from their previous experience.

Learning agents have the following elements.

- **The learning element:** This element enables learning agents to learn from previous experiences.
- **The critic:** It provides feedback on how the agent is doing.
- **The performance element:** This element decides on the external action that needs to be taken.
- **The problem generator:** This acts as a feedback agent that performs certain tasks such as making suggestions (new) and keeping history.

How intelligent agents work

Intelligent agents work through three main components: sensors, actuators, and effectors. Getting an overview of these components can improve our understanding of how intelligent agents work.

- **Sensors:** These are devices that detect any changes in the environment. This information is sent to other devices. In artificial intelligence, the environment of the system is observed by intelligent agents through sensors.

- **Actuators:** These are components through which energy is converted into motion. They perform the role of controlling and moving a system. Examples include rails, motors, and gears.

- **Effectors:** The environment is affected by effectors. Examples include legs, fingers, wheels, display screen, and arms.

Key takeaway

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.

1.8 Case Study: Kroger: How This U.S. Retail Giant Is Using AI And Robots to Prepare for the 4th Industrial Revolution

Kroger, one of the major grocery companies in the United States, has decided to embrace technology in order to survive and grow in the fourth industrial revolution. Kroger wants to use its data, shopper insights, and scale to help it remain a leader in the future marketplace, with 2,782 grocery shops under nearly two dozen names in 35 states. According to the Food Marketing Institute, by 2022, online grocery will account for 20% of all grocery retail and generate $100 billion in consumer sales, so Kroger and its competitors would be well to figure out how to leverage technology.

Restock Kroger Initiative

Kroger announced an ambitious three-year $9 billion plan called Restock Kroger in the fall of 2017, with the objective of expanding its e-commerce, digital, and omnichannel operations and redefining the consumer experience. The retailer already sends out 3 billion tailored recommendations per year, but they'll step up their efforts to "create unique client experiences." Shoppers will receive not just essential digital content, but also "inspiration" in the form of product-related content and recipes. The Restock Kroger effort also includes the expansion of Kroger's Scan, Bag, Go trial technology, which allows users to scan products while shopping with their smartphone. It will be revealed to 400 stores by the end of 2018 after being tested in 20 stores. Kroger's operations will be made more efficient by investing in Internet of Things (IoT) sensors, machine learning, and artificial intelligence.

Delivery by autonomous vehicles

We can get groceries delivered today, but Kroger is trying the delivery of the future: driverless vehicles delivering groceries. On its trial programme, Kroger collaborated with Nuro, a Silicon Valley startup that specialised in autonomous delivery cars. Customers can place same-day delivery orders via Kroger's ClickList ordering system and Nuro's app, but customers must be home to get their groceries from the car after entering a unique code to unlock the doors. There's no indication yet on which locations will be chosen to test autonomous deliveries, but local trade is expected to be disrupted.

Automated warehouses

Kroger and Ocado, a British online-only grocer, have formed a relationship that is anticipated to help Kroger automate its warehouses and employ artificial intelligence to boost its bottom line. Ocado claims to have the world's most advanced automated grocery warehouses and has tested delivery options with Uber and Instacart, and it's this know-how that Kroger hopes to capitalise on with its investment. The companies announced that Ocado would operate three new warehouses, with another 17 to come in the next three years. Ocado's warehouses are run by robots that explore the warehouse and pick products for orders using machine learning algorithms. Kroger will be able to get products to shops faster thanks to this investment and access to Ocado's technologies.

Marketing gets a boost from analytics

Kroger Precision Marketing, which uses customer purchase data from Kroger's 60 million shopper households to conduct marketing campaigns across a digital spectrum, was launched by Kroger's in-house analytics business 84.51. This not only improves personalization for customers, but it also provides product producers with fantastic marketing opportunities on Kroger.com, branded digital media, and the MyMagazine Sharing Network.

Machine learning

In a project called Embedded Machine Learning, 84.51 has made it a goal to enable and embed machine learning into Kroger's operations, where a "machine learning machine" can generate and deploy a large number of models with very little human intervention. With an aim to "enable, empower, and engage" machine learning throughout the business, this was a comprehensive approach to machine learning, with three phases to machine learning methodology: Solution Engineering, Model Development, and Model Deployment.

Smart shelves

When a Kroger consumer travels down the aisle with the Kroger app open, sensors recognise the shopper and use smart shelves technology to deliver personalised pricing and highlight products the customer might be interested in. Smart shelves benefit not only the customer, but they also assist the business check inventory to ensure that expired products aren't on the shelves and that everything is stocked correctly—and this has an impact on the customer experience. Kroger has been testing this technology since 2015, and while adoption has been slow, the retailer is attempting to improve it before deploying it.

Unit - 2

Problem-solving

2.1 Solving Problems by Searching, Problem-Solving Agents

In Artificial Intelligence, search techniques are universal problem-solving procedures. To solve a given problem and deliver the optimum outcome, rational agents or problem-solving agents in AI used these search techniques or algorithms. Problem-solving agents are goal-based agents that use atomic representation. In this subject, we will explore a variety of problem-solving search approaches.

Searching Algorithms Terminologies:

- Search: In a given search space, searching is a step-by-step procedure for addressing a search problem. There are three primary variables that can influence the outcome of a search:

  - Search space: A search space is a collection of probable solutions that a system could have.
  - Start state: It's the starting point for the agent's quest.
  - Goal test: It's a function that looks at the current state and returns whether or not the goal state has been reached.

- Search tree: Search tree is a tree representation of a search problem. The root node, which corresponds to the initial state, is at the top of the search tree.

- Actions: It provides the agent with a list of all available actions.

- Transition model: A transition model is a description of what each action does.

- Path cost: It's a function that gives each path a numerical cost.
- Solutions: It is an action sequence that connects the start and end nodes.
- Optimal solutions: If a solution is the cheapest of all the options.

Formulating problems

The problem of getting to Bucharest had been previously defined in terms of the starting state, activities, transition model, goal test, and path cost. Despite the fact that this formulation appears to be reasonable, it is still a model—an abstract mathematical description—rather than the real thing.

Compare our simple state description, In (Arad), to a cross-country trip, where the state of the world includes a wide range of factors such as the traveling companions, what's on the radio, the scenery out the window, whether there are any law enforcement officers nearby, how far it is to the next rest stop, the road condition, the weather, and so on.

All of these factors aren't mentioned in our state descriptions because they have nothing to do with finding a route to Bucharest. The process of removing details from a representation is known as abstraction.

In addition to the state description, we must abstract both the state description and the actions themselves. A driving action can result in a number of different outcomes. It takes time, consumes fuel, emits emissions, and changes the agent, in addition to shifting the vehicle's and its occupants' positions (as they say, travel is broadening). In our system, only the change in position is taken into account.

Can we be more precise about the proper level of abstraction? Consider the abstract states and actions we've selected as large collections of detailed world states and action sequences. Consider a path from Arad to Sibiu to Rimnicu Vilcea to Pitesti to Bucharest as an example of a solution to the abstract problem. A wide number of more detailed paths correlate to this abstract solution.

Searching for solution

We must now address the issues we've identified. Because a solution consists of a series of activities, search algorithms consider a number of different action sequences. The SEARCH TREE possible action sequences start with the starting state and build a search tree with the initial state NODE at the root; the branches are actions, and the nodes are states in the problem's state space.

Figure depicts the first few steps in developing a search tree for finding a route from Arad to Bucharest. The root node of the tree represents the initial state (Arad). The first stage is to determine whether or not this is a goal state. (Of course, it isn't, but it's worth double-checking to avoid problems like "starting in Arad, get to Arad.") Then we must weigh a number of possibilities. This is done by extending the current state; in other words, each legal action is applied to the existing state, resulting in the production of a new set of states. In this scenario, we create three additional child nodes from the parent node In(Arad): In(Sibiu), In(Timisoara), and In(Sibiu) (Zerind).

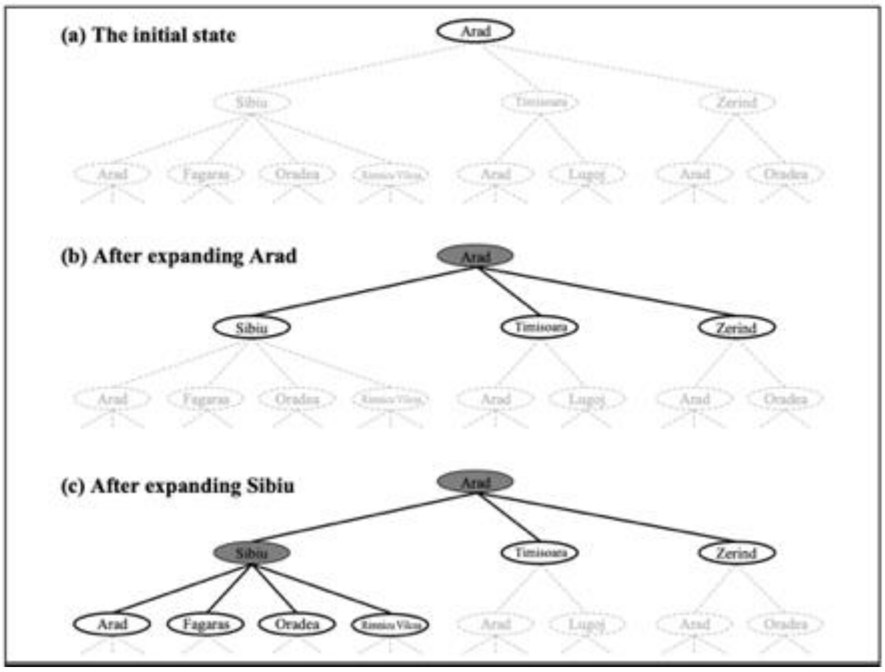


Fig: Partial search tree

This is the essence of searching: pursuing one option at a time while putting the others on hold in the event that the first does not yield a solution. Let's use Sibiu as an example. We first check to see whether it's a target state (it isn't), then expand it to get In(Arad), In(Fagaras), In(Oradea), and In(Oradea) (RimnicuVilcea). Then we can choose one of these four possibilities, or we can go back to LEAF NODE and choose Timisoara or Zerind. Each of these six nodes in the tree is a leaf node, which means it has no progeny.

At any given FRONTIER point, the frontier is the collection of all leaf nodes that are available for expansion. Each tree's border is defined by the nodes with bold outlines in Figure.

The process of choosing and expanding nodes on the frontier continues until a solution is found or no more states can be added to the frontier. Informally, the TREE-SEARCH algorithm is depicted in the figure. All search algorithms have the same core basis; the difference is in how they choose which state to expand next—the so-called search strategy.

Function TREE-SEARCH (problem) **returns** a solution, or failure

Initialize the frontier using the initial state of problem

Loop do

```
If the frontier is empty then return failure
Choose a leaf node and remove it from the frontier
If the node contains a goal state then return the corresponding solution
Expand the chosen node, adding the resulting nodes to the frontier
Function GRAPH-SEARCH(problem) returns a solution, or failure
Initialize the explored set to be empty
Loop do
If the frontier is empty then return failure
Choose a leaf node and remove it from the frontier
If the node contains a goal state then return the corresponding solution
Add the node to the explored set
Expand the chosen node, adding the resulting nodes to frontier
Only if not in the frontier or explored set
```

Problem solving agents

By defining problems and their various solutions, the problem-solving agent performs exactly.

In the field of Artificial Intelligence, a problem-solving agent is a goal-based agent that focuses on goals. It is one embodiment of a combination of algorithms and strategies to tackle a well-defined problem. These agents differ from reflex agents in that they only have to map states into actions and are unable to map when storing and learning are both larger.

The following are the stages that problem-solving agents go through to reach a desired state or solution:

- Articulating or expressing the desired goal and the problem is tried upon, clearly.
- Explore and examine
- Find the solution from the various algorithms on the table.
- The final step is Execution!

The steps taken by the problem-solving agent

- Goal formulation: is the first and most basic step in solving a problem. It organises the steps/sequence needed to construct a single goal from many goals, as well as the actions needed to achieve that goal. The existing circumstance and the agent's performance measure are used to formulate goals.
- Problem formulation: is the most crucial step in the problem-solving process because it determines what activities should be taken to attain the stated goal. In the formulation of a problem, there are five elements to consider:
- First State: This is the agent's initial state, or first step toward its goal.
- Actions: This is a list of the several options available to the agent.
- Transition Model: The Transition Model explains what each step does.
- Goal test: It determines whether or not a given state is a desired state.
- Path cost: Each path that leads to the goal is given a numerical cost. A cost function is chosen by the problem-solving agent to reflect its

performance measure. Remember that the optimal option has the cheapest path cost of all the alternatives.

Note that the problem's state-space is implicitly defined by the initial state, actions, and transition model. A problem's state-space is a collection of all states that can be achieved by following any series of activities from the beginning state. The state-space is represented as a directed map or graph, with nodes representing states, links between nodes representing actions, and a path representing a series of states connected by a series of actions.

- **Search** - It determines the best potential sequence of actions to get from the current condition to the goal state. It receives an issue as input and returns a solution as output.
- **Solution** - It selects the best algorithm from a set of algorithms that can be demonstrated to be the most optimal solution.
- **Execution** - It uses the best optimal solution found by the searching algorithms to get from the present state to the goal state.

Key takeaway

Search techniques are universal problem-solving strategies in Artificial Intelligence.

The problem of traveling to Bucharest was previously formulated in terms of the beginning state, activities, transition model, goal test, and path cost.

A solution is a series of actions, search algorithms explore a variety of alternative action sequences.

2.2 Example Problems

In general, there are two sorts of problem-solving strategies:

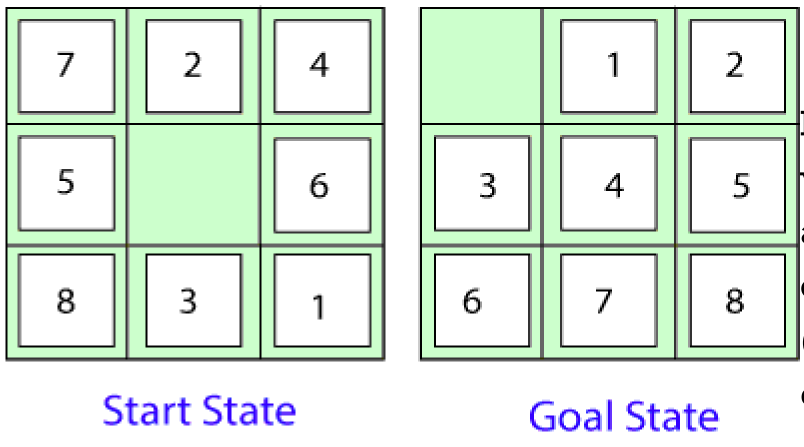
- **Toy problem** - Researchers use it to compare the performance of algorithms because it provides a succinct and clear explanation of the problem.
- **Real world problem** - The problems that need to be solved are those that occur in the real world. It does not require descriptions, unlike a toy problem, yet we can have a generic formulation of the problem.

1. Some Toy Problems

8 Puzzle Problem: A 33 matrix with moveable tiles numbered 1 to 8 and a vacant area is shown. The tile to the left of the vacant spot can be slid into it. The goal is to achieve a goal state that is similar to the one indicated in the diagram below.

Our goal is to slide digits into the blank space in the figure to change the current state to the goal state.

Fig: 8 Puzzle problem



By sliding digits into the vacant space in the above diagram, we can change the current (Start) state to the desired state.

The following is the problem statement:

- States: It shows where each numbered tile and the blank tile are located.
- Initial State: Any state can be used as the starting point.
- Actions: The blank space's actions are defined here, i.e., left, right, up, or down.
- Transition model: It returns the final state, which is determined by the provided state and actions.
- Goal test: It determines if we have arrived at the correct goal-state.
- Path cost: The path cost is the number of steps in a path where each step costs one dollar.

8-queens problem: The goal of this issue is to arrange eight queens on a chessboard in such a way that none of them can attack another queen. A queen can attack other queens in the same row and column or diagonally.

We can grasp the problem and its correct solution by looking at the diagram below.

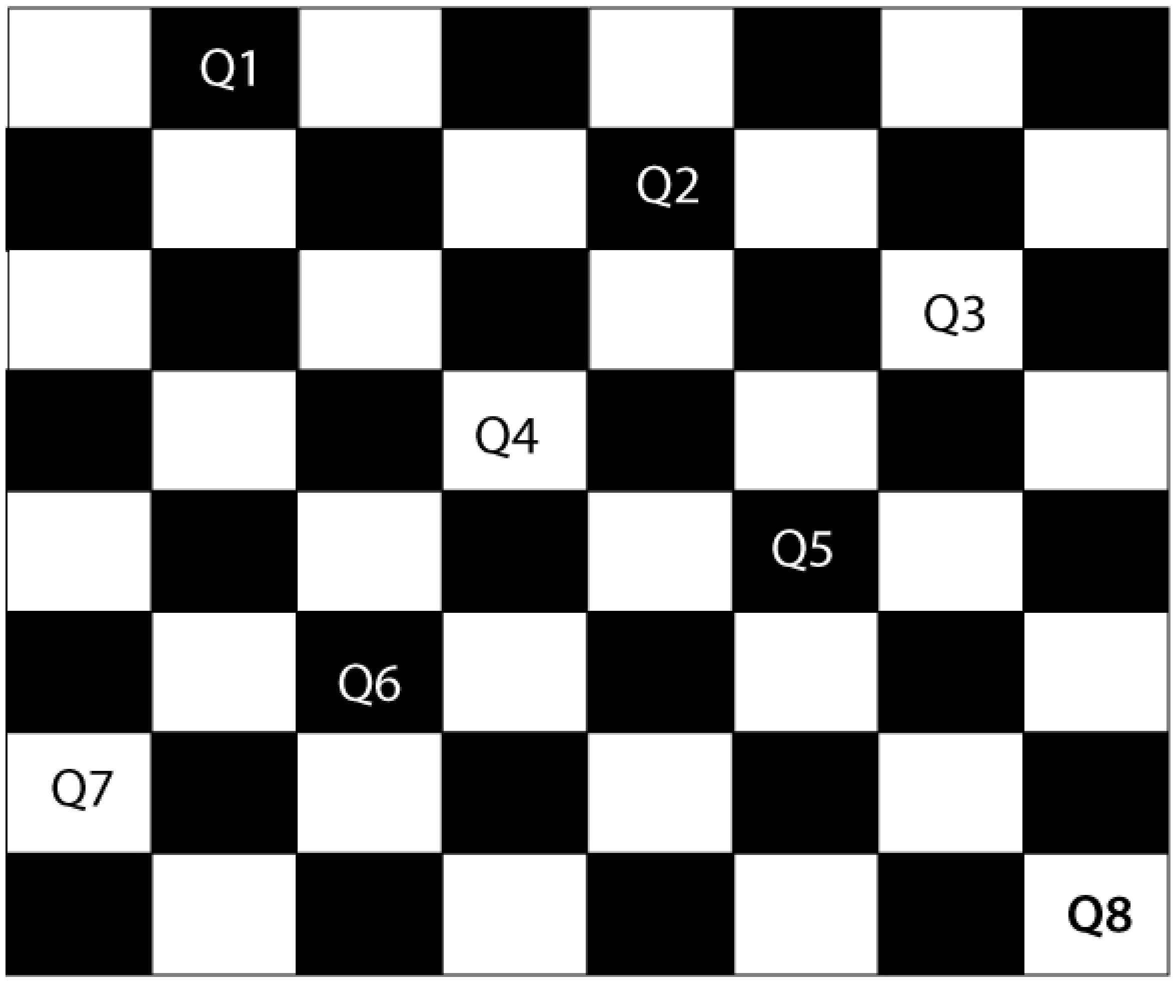


Fig: 8 Queen puzzle

As can be seen in the diagram above, each queen is put on the chessboard in such a way that no other queen is placed diagonally, in the same row or column. As a result, it is one viable solution to the eight-queens dilemma.

For this problem, there are two main kinds of formulation:

Incremental formulation: It begins with an empty state and progresses in steps, with the operator augmenting a queen at each step.

Following steps are involved in this formulation:

- States: Arrangement of any 0 to 8 queens on the chessboard.
- Initial State: An empty chessboard
- Actions: Add a queen to any empty box.
- Transition model: Returns the chessboard with the queen added in a box.
- Goal test: Checks whether 8-queens are placed on the chessboard without any attack.
- Path cost: There is no need for path cost because only final states are counted.

In this formulation, there is approximately 1.8 x 1014 possible sequence to investigate.

Complete-state formulation: It starts with all the 8-queens on the chessboard and moves them around, saving from the attacks.

In this formulation, the steps are as follows:

States: Each of the eight queens is arranged in a column, with no queen assaulting the other.

Actions: Move the queen to a spot where it will be secure from attacks.

This formulation is superior to the incremental formulation since it shrinks the state space from 1.8 x 1014 to 2057 and makes finding solutions much easier.

2. Some Real-world problems

Traveling salesperson problem(TSP): It's a problem of touring, because the salesman can only visit each city once. The goal is to discover the shortest tour and sell out all of the merchandise in each place.

VLSI Layout problem: Millions of components and connections are placed on a chip in order to reduce area, circuit delays, and stray capacitances while increasing manufacturing yield.

The layout problem is split into two parts:

- **Cell layout:** The circuit's primitive components are arranged into cells, each of which performs a distinct purpose. Each cell is the same size and shape. The goal is to arrange the cells on the chip so that they do not overlap.

- **Channel routing:** It determines a unique path through the spaces between the cells for each wire.

- **Protein Design: T**he goal is to develop an amino acid sequence that will fold into a 3D protein with the ability to treat an illness.

2.3 Search Algorithms

Problem-solving agents:

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

Search Algorithm Terminologies:

● 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.

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.

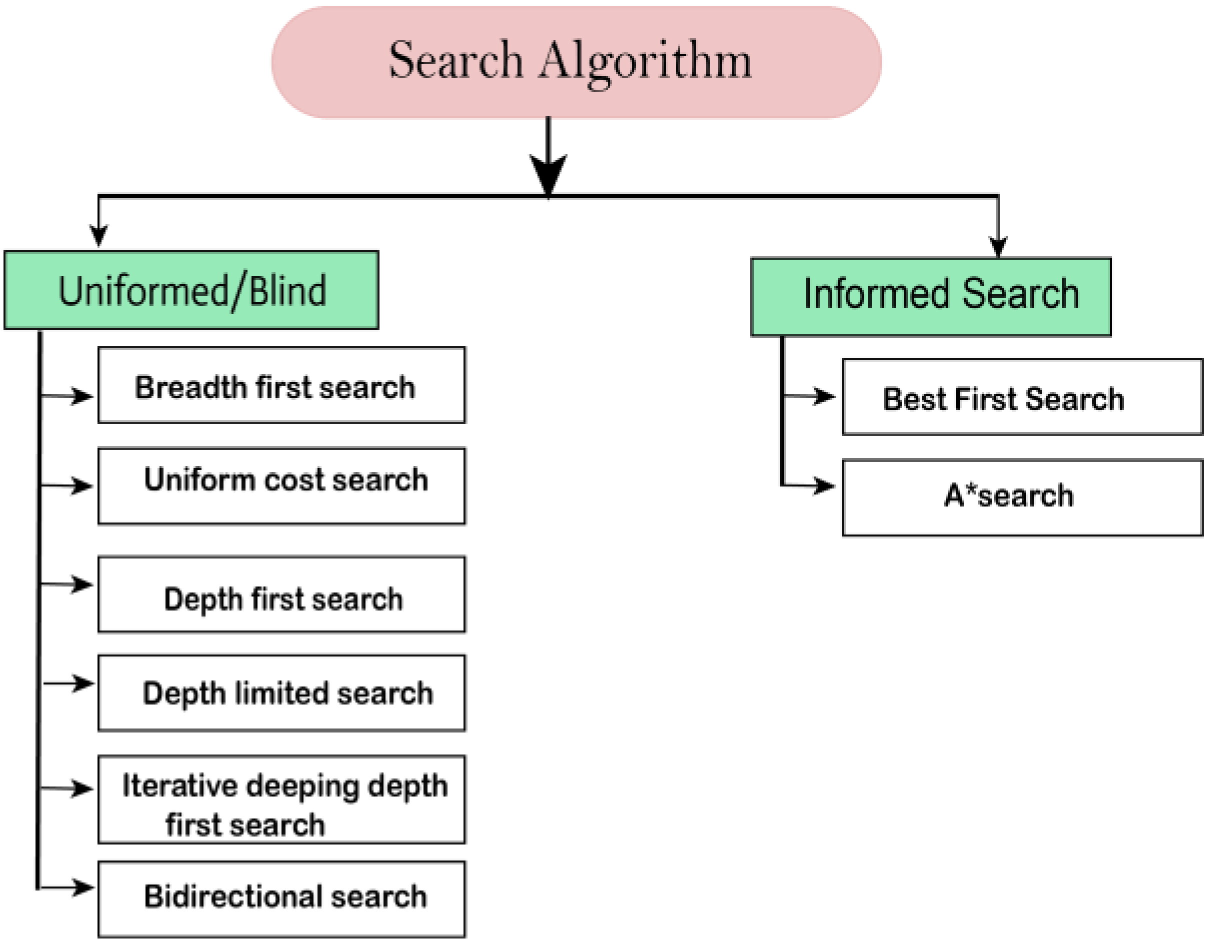


Fig: Search algorithm

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.

1. Greedy Search
2. A* Search

Uninformed Search Algorithms

Uninformed search is a class of general-purpose search algorithms which operates in brute force-way. Uninformed search algorithms do not have additional information about state or search space other than how to traverse the tree, so it is also called blind search.

Following are the various types of uninformed search algorithms:

1. Breadth-first Search
2. Depth-first Search
3. Depth-limited Search
4. Iterative deepening depth-first search
5. Uniform cost search
6. Bidirectional Search

1. 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.

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.

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:

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

Time Complexity: Time Complexity of BFS algorithm can be obtained by the number of nodes traversed in BFS until the shallowest Node. Where the d= depth of shallowest solution and b is a node at every state.

T (b) = $1+b^2+b^3+.......+ b^d= O (b^d)$

Space Complexity: Space complexity of BFS algorithm is given by the Memory size of frontier which is $O(b^d)$.

Completeness: BFS is complete, which means if the shallowest goal node is at some finite depth, then BFS will find a solution.

Optimality: BFS is optimal if path cost is a non-decreasing function of the depth of the node.

2. Depth-first Search

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

Note: Backtracking is an algorithm technique for finding all possible solutions using recursion.

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.

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.

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

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

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

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

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

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

3. 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.

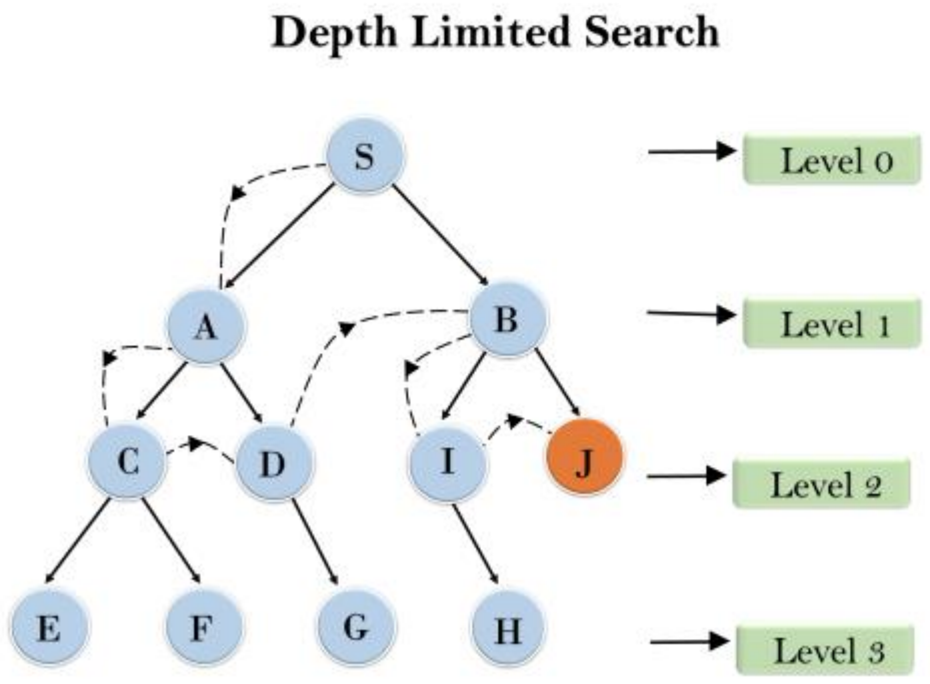
Advantages:

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.

Example:



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

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

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

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

4. Uniform-cost Search Algorithm:

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

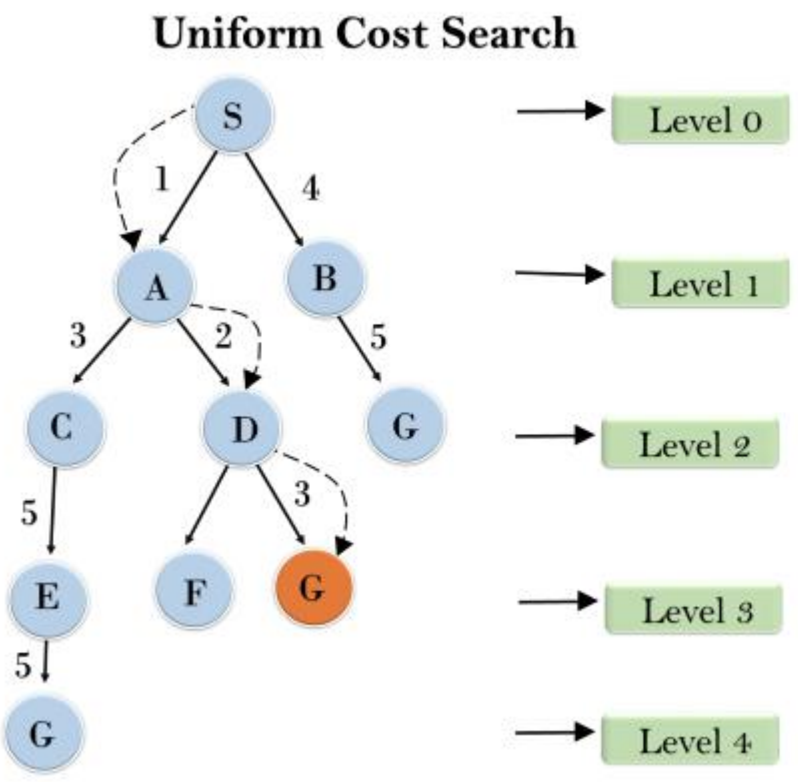
Advantages:

- 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.

Example:



Completeness:

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

Time Complexity:

Let C* is Cost of the optimal solution, and ε is each step to get closer to the goal node. Then the number of steps is = C*/ε+1. Here we have taken +1, as we start from state 0 and end to C*/ε.

Hence, the worst-case time complexity of Uniform-cost search is$O(b^{1 + [C^*/\varepsilon]})$/.

Space Complexity:

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

Optimal:

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

5. Iterative 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.

Advantages:

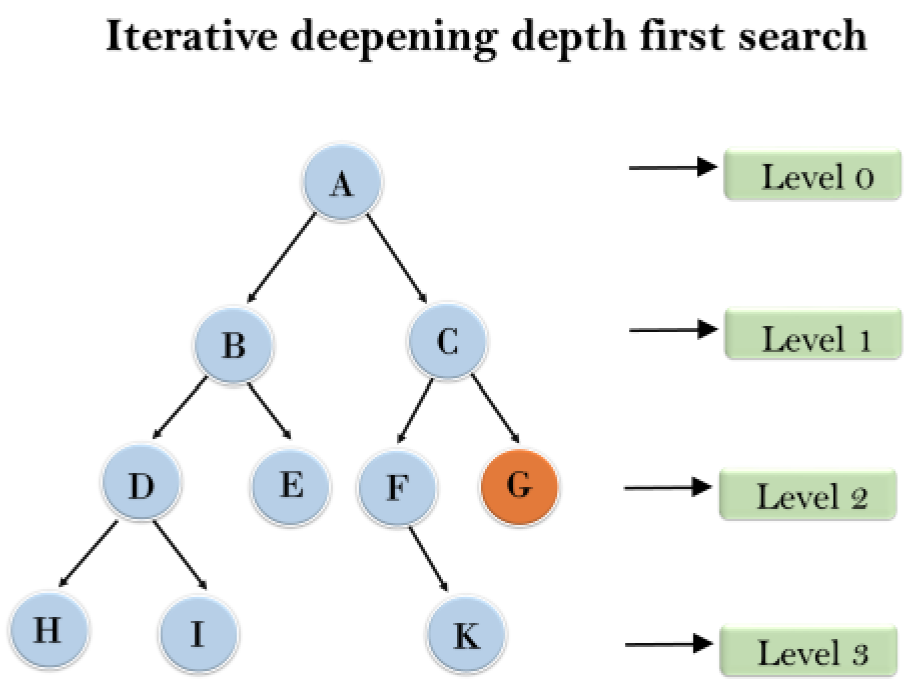
- It Combines the benefits of BFS and DFS search algorithm in terms of fast search and memory efficiency.

Disadvantages:

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

Example:

Following tree structure is showing the iterative deepening depth-first search. IDDFS algorithm performs various iterations until it does not find the goal node. The iteration performed by the algorithm is given as:



1'st Iteration----> A

2'nd Iteration----> A, B, C

3'rd Iteration------>A, B, D, E, C, F, G

4'th Iteration------>A, B, D, H, I, E, C, F, K, G

In the fourth 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.

6. Bidirectional Search Algorithm:

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

Bidirectional search can use search techniques such as BFS, DFS, DLS, etc.

Advantages:

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

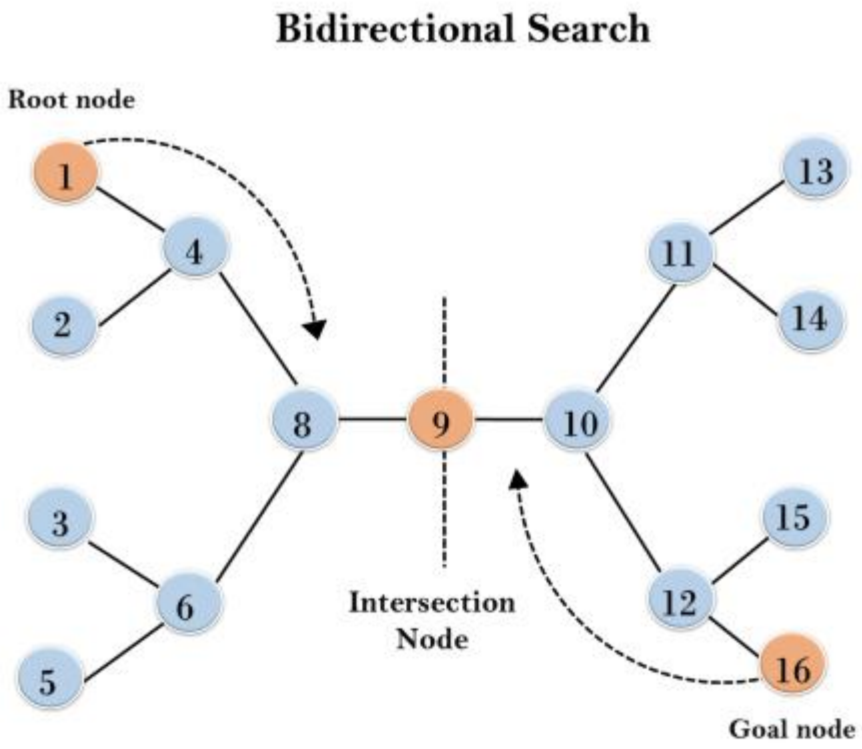
Disadvantages:

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

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.

Key takeaway

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.

2.4 Uninformed Search Strategies

Uninformed search, also known as blind, exhaustive, or brute-force search, is a type of search that doesn't use any information about the problem to guide it and so isn't very efficient.

Breadth first search

The breadth first search technique is a general strategy for traversing a graph. A breadth first search takes up more memory, but it always discovers the shortest path first. In this type of search, the state space is represented as a tree. The answer can be discovered by traveling the tree. The tree's nodes indicate the initial value of starting state, multiple intermediate states, and the end state.

This search makes use of a queue data structure, which is traversed level by level. In a breadth first search, nodes are extended in order of their distance from the root. It's a path-finding algorithm that, if one exists, can always find a solution. The answer that is discovered is always the alternative option. This task demands a significant amount of memory. At each level, each node in the search tree is increased in breadth.

Algorithm:

Step 1: Place the root node inside the queue.

Step 2: If the queue is empty then stops and returns failure.

Step 3: If the FRONT node of the queue is a goal node, then stop and return success.
Step 4: Remove the FRONT node from the queue. Process it and find all its neighbours that are in a ready state then place them inside the queue in any order.

Step 5: Go to Step 3.

Step 6: Exit.

Implementations

Let us implement the above algorithm of BFS by taking the following suitable example

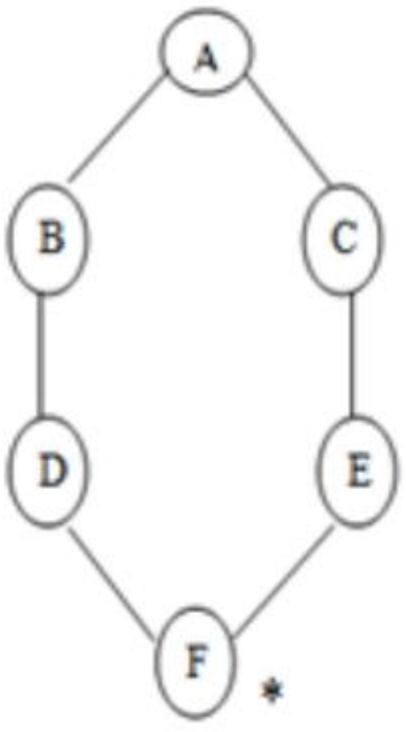


Fig: Example

Consider the graph below, where A is the initial node and F is the destination node (*).

Step 1: Place the root node inside the queue i.e. A

	A	

Step 2: The queue is no longer empty, and the FRONT node, i.e. A, is no longer our goal node. So, proceed to step three.

Step 3: Remove the FRONT node, A, from the queue and look for A's neighbors, B and C.

	C	B	A

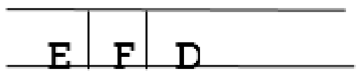
Step 4: Now, b is the queue's FRONT node. As a result, process B and look for B's neighbors, i.e., D.

C	D	B

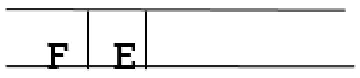
Step 5: Find out who C's neighbors are, which is E.

	D	E	C

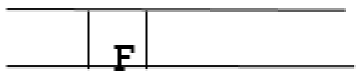
Ste 6: As D is the queue's FRONT node, find out who D's neighbors are.



Step 7: E is now the queue's first node. As a result, E's neighbor is F, which is our objective node.



Step 8: Finally, F is the FRONT of the queue, which is our goal node. As a result, leave.



Advantages

- The goal will be achieved in whatever way possible using this strategy.
- It does not take any unproductive pathways for a long time.
- It finds the simplest answer in the situation of several pathways.

Disadvantages

- BFS consumes a significant amount of memory.
- It has a greater level of time complexity.
- It has long pathways when all paths to a destination have about the same search depth.

Key takeaway

Breadth-first search is a search method in which the highest layer of a decision tree is entirely searched before moving on to the next layer (BFS).

Because no feasible solutions are omitted in this technique, an optimal solution is certain to be found.

When the search space is large, this method is frequently impractical.

2.5 Informed (Heuristic) Search Strategies

A heuristic is a technique for making our search process more efficient. Some heuristics assist in the direction of a search process without claiming completeness, whereas others do. A heuristic is a piece of problem-specific knowledge that helps you spend less time looking for answers. It's a strategy that works on occasion, but not all of the time.

The heuristic search algorithm uses the problem information to help steer the way over the search space. These searches make use of a number of functions that, assuming the function is efficient, estimate the cost of going from where you are now to where you want to go.

A heuristic function is a function that converts problem situation descriptions into numerical measures of desirability. The heuristic function's objective is to direct the search process in the most profitable routes by recommending which path to take first when there are multiple options.

The following procedures should be followed when using the heuristic technique to identify a solution:

1. Add domain—specific information to select what is the best path to continue searching along.

2. Define a heuristic function h(n) that estimates the 'goodness' of a node n. Specifically, h(n) = estimated cost (or distance) of minimal cost path from n to a goal state.

3. The term heuristic means 'serving to aid discovery' and is an estimate, based on domain specific information that is computable from the current state description of how close we are to a goal.

A search problem in which multiple search orders and the use of heuristic knowledge are clearly understood is finding a path from one city to another.

1. State: The current city in which the traveller is located.

2. Operators: Roads linking the current city to other cities.

3. Cost Metric: The cost of taking a given road between cities.

4. Heuristic information: The search could be guided by the direction of the goal city from the current city, or we could use airline distance as an estimate of the distance to the goal.

Key takeaway

Informed Search also called heuristic or intelligent search, this uses information about the problem to guide the search—usually guesses the distance to a goal state and is therefore efficient, but the search may not always be possible.

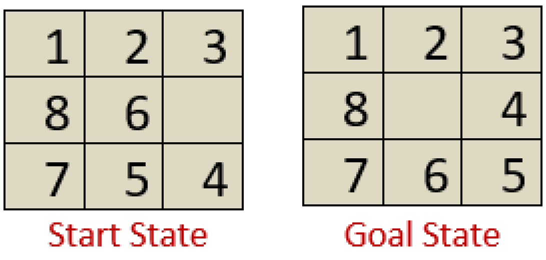
The purpose of heuristic function is to guide the search process in the most profitable directions by suggesting which path to follow first when more than is available.

2.6 Heuristic Functions

As we've seen, an informed search employs heuristic functions in order to get at the destination node in a more conspicuous manner. As a result, there are various ways to get from the present node to the goal node in a search tree. It is undeniably important to choose a decent heuristic function. The efficiency of a heuristic function determines its usefulness. The more knowledge about the problem there is, the longer it takes to solve it.

A heuristic function can help solve some toy problems more efficiently, such as 8-puzzle, 8-queen, tic-tac-toe, and so on. Let's have a look at how:

Consider the eight-puzzle issue below, which has a start and a target state. Our goal is to slide the tiles from the current/start state into the goal state in the correct order. There are four possible movements: left, right, up, and down. There are various ways to transform the current/start state to the desired state, but we can solve the problem more efficiently by using the heuristic function h(n).

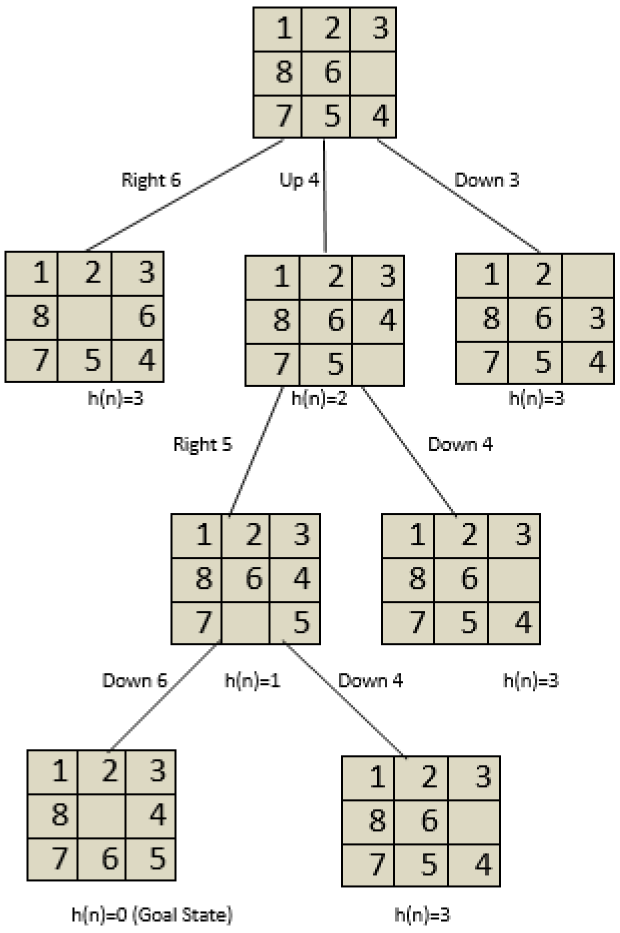


The following is a heuristic function for the 8-puzzle problem:

h(n)=Number of tiles out of position.

So, there are three tiles that are out of place, namely 6, 5, and 4. The empty tile in the goal state is not counted). h(n)=3 in this case. The value of h(n) =0 must now be minimised.

To reduce the h(n) value to 0, we can build a state-space tree as shown below:



The objective state is minimised from h(n)=3 to h(n)=0, as seen in the state space tree above. However, depending on the requirement, we can design and employ a number of heuristic functions. A heuristic function h(n) can alternatively be defined as the knowledge needed to solve a problem more efficiently, as shown in the previous example. The information can be related to the nature of the state, the cost of changing states, the characteristics of target nodes, and so on, and is stated as a heuristic function.

Key takeaway

As we've seen, an informed search employs heuristic functions in order to get at the destination node in a more conspicuous manner. As a result, there are various ways to get from the present node to the goal node in a search tree. It is undeniably important to choose a decent heuristic function. The efficiency of a heuristic function determines its usefulness. The more knowledge about the problem there is, the longer it takes to solve it.

2.7 Search in Complex Environments

Agents rarely have complete control over their surroundings, and are more likely to have only partial control. This implies that they have control over the environment. Changes in the environment, in turn, will have a direct impact on the agents in the environment.

Environments are often described as non-deterministic. That is to say, activities taken in particular circumstances may fail. Furthermore, an agent executing the same task in two different contexts can provide radically different results.

An agent in an environment will have a pre-programmed set of talents that it may use to deal with various circumstances it may encounter. Effectoric capacities are the name given to these skills. The agent has a sensor that is plainly affected by the surroundings, as seen in the diagram on agents. The agent can use data from this sensor with previously collected data to determine which action to take.

Obviously, not every action can be carried out in every circumstance. An agent, for example, might be able to 'open door,' but only if the door is unlocked. A precondition is that the door must be unlocked before the action may be conducted.

The most difficult difficulty that agents face in an environment is determining which action to take at any given time in order to maximise their chances of achieving their goal, or at least working toward it. The qualities of an environment influence the complexity of a decision-making process.

There are several types of environments:

- Fully Observable vs Partially Observable
- Deterministic vs Stochastic
- Competitive vs Collaborative
- Single-agent vs Multi-agent
- Static vs Dynamic
- Discrete vs Continuous

Fully Observable vs Partially Observable

- A fully observable environment is one in which an agent sensor can perceive or access the complete state of an agent at any given time; otherwise, it is a partially observable environment.
- It's simple to preserve a completely observable environment because there's no need to keep track of the environment's past.
- When the agent has no sensors in all environments, it is said to be unobservable.

Deterministic vs Stochastic

- The environment is considered to be deterministic when a uniqueness in the agent's present state totally determines the agent's next state.
- The stochastic environment is random in nature, not unique, and the agent cannot entirely control it.

Competitive vs Collaborative

- When an agent competes with another agent to optimise the output, it is said to be in a competitive environment.

● Chess is a competitive game in which the agents compete against one another to win the game, which is the output.

● When numerous agents work together to generate the required result, the agent is said to be in a collaborative environment.

● When many self-driving cars are discovered on the road, they work together to avoid collisions and arrive at their planned destination.

Single-agent vs Multi-agent

● A single-agent environment is defined as one in which there is just one agent.

● A single-agent system is exemplified by a person who is left alone in a maze.

● A multi-agent environment is one in which there are multiple agents.

● Football is a multi-agent sport since each team has 11 players.

Dynamic vs Static

● Dynamic refers to an environment that changes constantly while the agent is engaged in some action.

● A roller coaster ride is dynamic since it is set in motion and the surroundings changes at all times.

● A static environment is one that is idle and does not modify its state.

● When an agent enters a vacant house, there is no change in the surroundings.

Discrete vs Continuous

● A discrete environment is one in which there are a finite number of actions that can be deliberated in the environment to produce the output.

- Chess is a discrete game since it has a limited number of moves. The amount of moves varies from game to game, but it is always finite.
- Continuous refers to an environment in which actions cannot be numbered, i.e. it is not discrete.
- Self-driving cars are an example of continuous environments as their actions are driving, parking, etc. which cannot be numbered.

2.8 Local Search and Optimization Problems

Local search algorithms

A local search algorithm completes its task by traversing on a single current node rather than multiple paths and following the neighbours of that node generally.

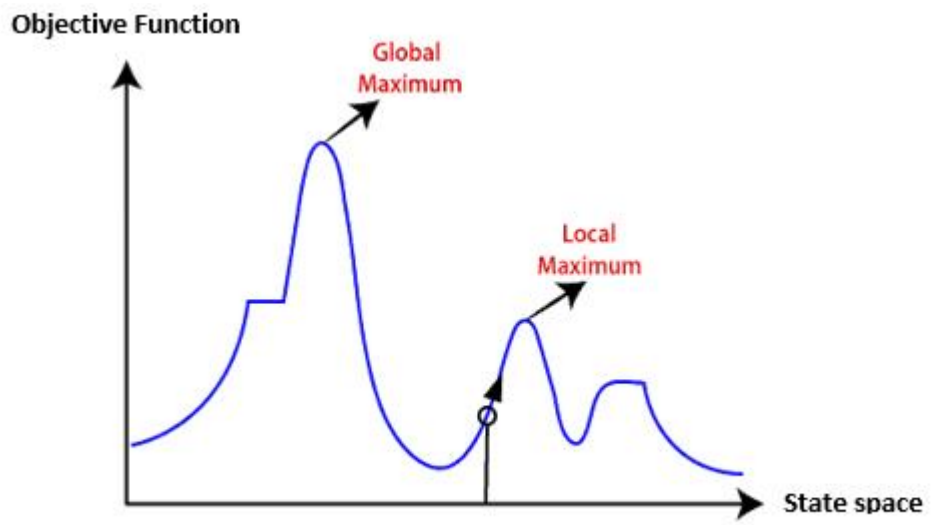
Although local search algorithms are not systematic, still they have the following two advantages:

- Local search algorithms use a very little or constant amount of memory as they operate only on a single path.
- Most often, they find a reasonable solution in large or infinite state spaces where the classical or systematic algorithms do not work.

Working of a Local search algorithm

Consider the below state-space landscape having both:

- Location: It is defined by the state.
- Elevation: It is defined by the value of the objective function or heuristic cost function.



A one-dimensional state-space landscape in which elevation corresponds to the objective function

The local search algorithm explores the above landscape by finding the following two points:

- Global Minimum: If the elevation corresponds to the cost, then the task is to find the lowest valley, which is known as Global Minimum.

- Global Maxima: If the elevation corresponds to an objective function, then it finds the highest peak which is called as Global Maxima. It is the highest point in the valley.

Some different types of local searches:

- Hill-climbing Search
- Simulated Annealing
- Local Beam Search

Note: Local search algorithms do not burden to remember all the nodes in the memory; it operates on complete state-formulation.

Hill Climbing Algorithm:

Hill climbing search is a local search problem. The purpose of the hill climbing search is to climb a hill and reach the topmost peak/ point of that hill. It is based on the heuristic search technique where the person who is climbing up on the hill estimates the direction which will lead him to the highest peak.

State-space Landscape of Hill climbing algorithm

To understand the concept of hill climbing algorithm, consider the below landscape representing the goal state/peak and the current state of the climber. The topographical regions shown in the figure can be defined as:

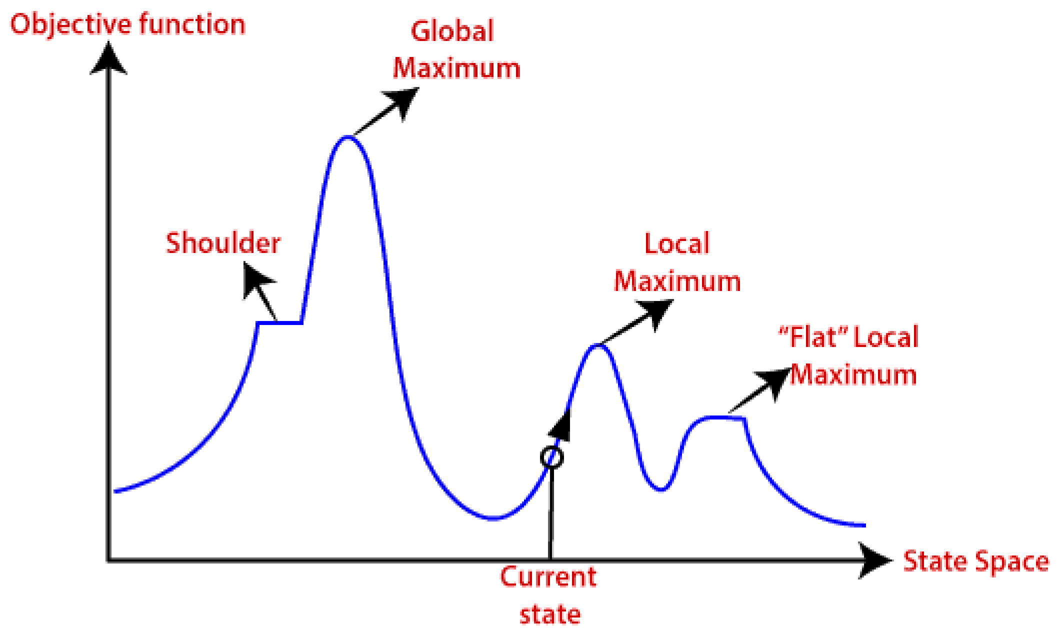
- Global Maximum: It is the highest point on the hill, which is the goal state.

- Local Maximum: It is the peak higher than all other peaks but lower than the global maximum.

- Flat local maximum: It is the flat area over the hill where it has no uphill or downhill. It is a saturated point of the hill.

- Shoulder: It is also a flat area where the summit is possible.

- Current state: It is the current position of the person.

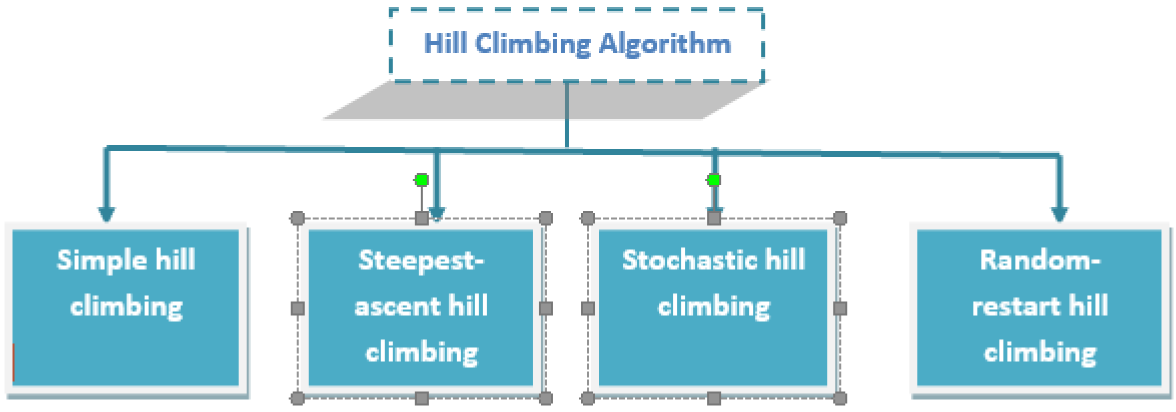


A one-dimensional state-space landscape in which elevation corresponds to the objective function

Types of Hill climbing search algorithm

There are following types of hill-climbing search:

- Simple hill climbing
- Steepest-ascent hill climbing
- Stochastic hill climbing
- Random-restart hill climbing



Simple hill climbing search

Simple hill climbing is the simplest technique to climb a hill. The task is to reach the highest peak of the mountain. Here, the movement of the climber depends on his move/steps. If he finds his next step better than the previous one, he continues to move else remain in the same state. This search focus only on his previous and next step.

Simple hill climbing Algorithm

1. Create a CURRENT node, NEIGHBOUR node, and a GOAL node.
2. If the CURRENT node=GOAL node, return GOAL and terminate the search.
3. Else CURRENT node<= NEIGHBOUR node, move ahead.
4. Loop until the goal is not reached or a point is not found.

Steepest-ascent hill climbing

Steepest-ascent hill climbing is different from simple hill climbing search. Unlike simple hill climbing search, It considers all the successive nodes, compares them, and choose the node which is closest to the solution. Steepest hill climbing search is similar to best-first search because it focuses on each node instead of one.

Note: Both simple, as well as steepest-ascent hill climbing search, fails when there is no closer node.

Steepest-ascent hill climbing algorithm

1. Create a CURRENT node and a GOAL node.
2. If the CURRENT node=GOAL node, return GOAL and terminate the search.
3. Loop until a better node is not found to reach the solution.
4. If there is any better successor node present, expand it.
5. When the GOAL is attained, return GOAL and terminate.

Stochastic hill climbing

Stochastic hill climbing does not focus on all the nodes. It selects one node at random and decides whether it should be expanded or search for a better one.

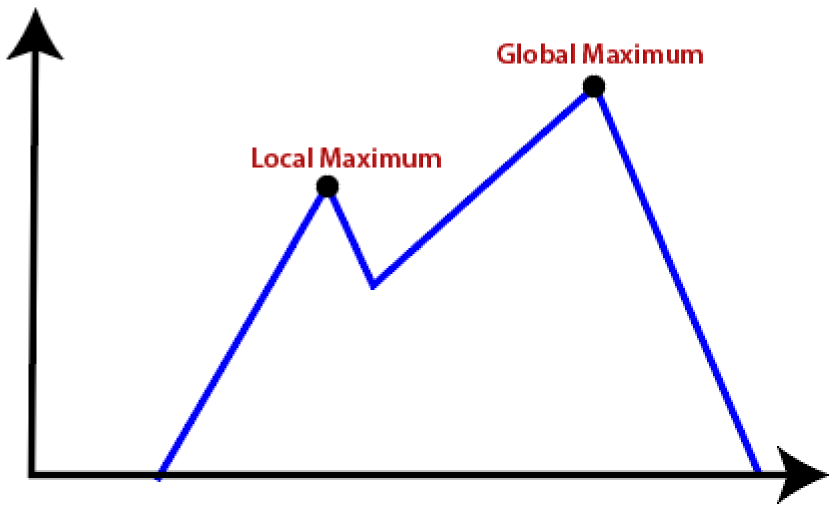
Random-restart hill climbing

Random-restart algorithm is based on try and try strategy. It iteratively searches the node and selects the best one at each step until the goal is not found. The success depends most commonly on the shape of the hill. If there are few plateaus, local maxima, and ridges, it becomes easy to reach the destination.

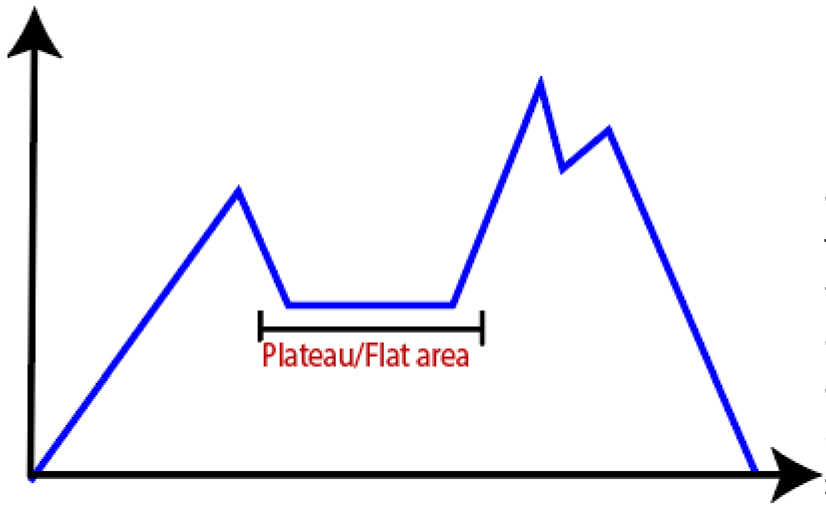
Limitations of Hill climbing algorithm

Hill climbing algorithm is a fast and furious approach. It finds the solution state rapidly because it is quite easy to improve a bad state. But, there are following limitations of this search:

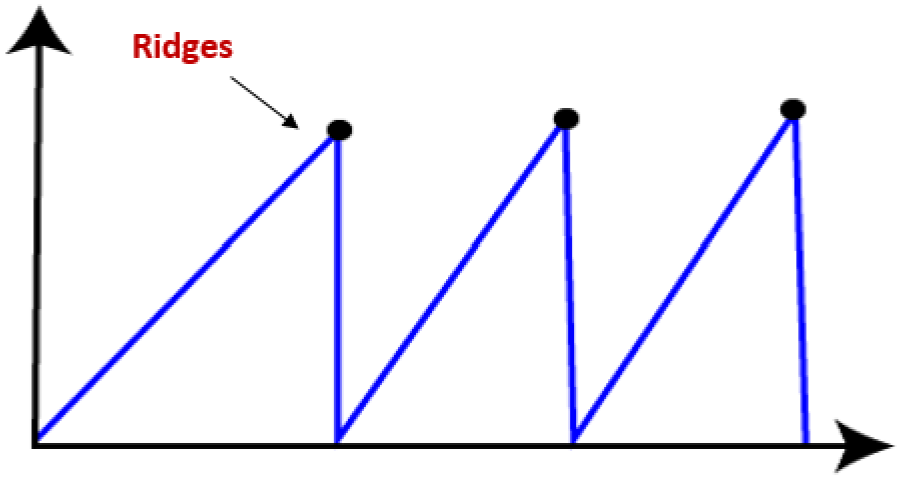
● Local Maxima: It is that peak of the mountain which is highest than all its neighbouring states but lower than the global maxima. It is not the goal peak because there is another peak higher than it.



- Plateau: It is a flat surface area where no uphill exists. It becomes difficult for the climber to decide that in which direction he should move to reach the goal point. Sometimes, the person gets lost in the flat area.



- Ridges: It is a challenging problem where the person finds two or more local maxima of the same height commonly. It becomes difficult for the person to navigate the right point and stuck to that point itself.



Simulated Annealing

Simulated annealing is similar to the hill climbing algorithm. It works on the current situation. It picks a random move instead of picking the best move. If the move leads to the improvement of the current situation, it is always accepted as a step towards the solution state, else it accepts the move having a probability less than 1.

This search technique was first used in 1980 to solve VLSI layout problems. It is also applied for factory scheduling and other large optimization tasks.

Local Beam Search

Local beam search is quite different from random-restart search. It keeps track of k states instead of just one. It selects k randomly generated states, and expand them at each step. If any state is a goal state, the search stops with success. Else it selects the best k successors from the complete list and repeats the same process.

In random-restart search where each search process runs independently, but in local beam search, the necessary information is shared between the parallel search processes.

Disadvantages of Local Beam search

- This search can suffer from a lack of diversity among the k states.
- It is an expensive version of hill climbing search.

2.9 Case Study: 4th Industrial Revolution Using AI, Big Data and Robotics

The fourth industrial revolution, often known as Industry 4.0, is getting a lot of attention, especially because of its potential influence on humanity. 4IR, according to Schwab, will alter how people live, work, and conduct business, as well as how we are governed. It is also thought that the industrial revolutions began in the 17th century, with Britain leading the way with the so-called "first" industrial revolution. The term "industrial revolution" refers to a period of economic upheaval that forced a shift in "people's livelihood from agrarian rural livelihood to city and town livelihood."

According to Blinov, economic activities were limited prior to the arrival of the first industrial revolution, which was championed by Britain, and as a result, many people were destitute. Human livelihood was based on what individuals got from small farms, which made life tough for the average person.

People were employing simple tools for production, which were mostly hand tools, and production was mostly confined to people's houses throughout this period.

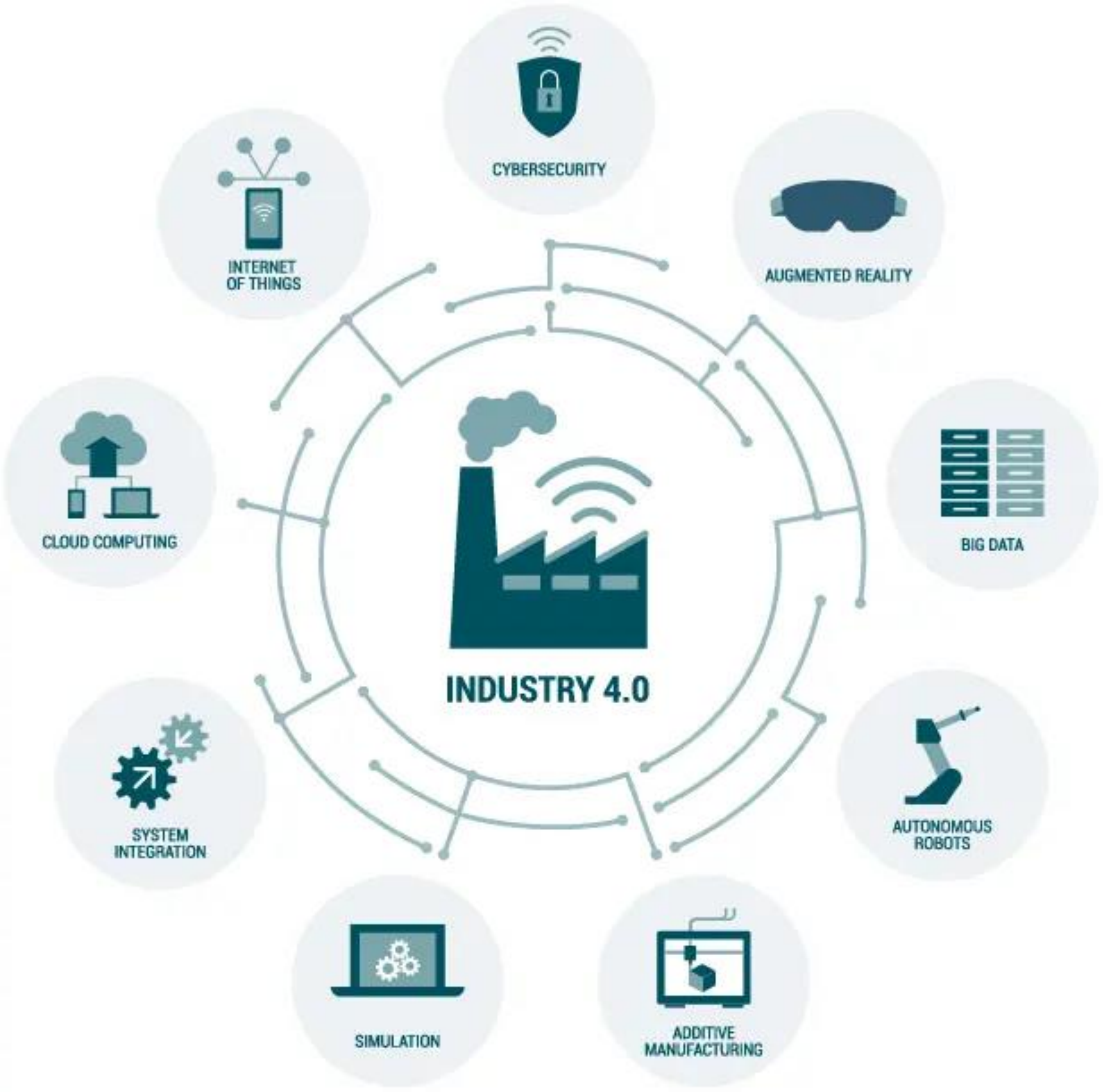
The steam engine replaced animal power, signalling the "transition from rural livelihood" to industrialization, when "special-purpose machinery" was now used, kicking off the first industrial revolution in Britain at the turn of the century.

Industry 4.0 is a new stage in the structure and control of the industrial value chain that is used interchangeably with the fourth industrial revolution.

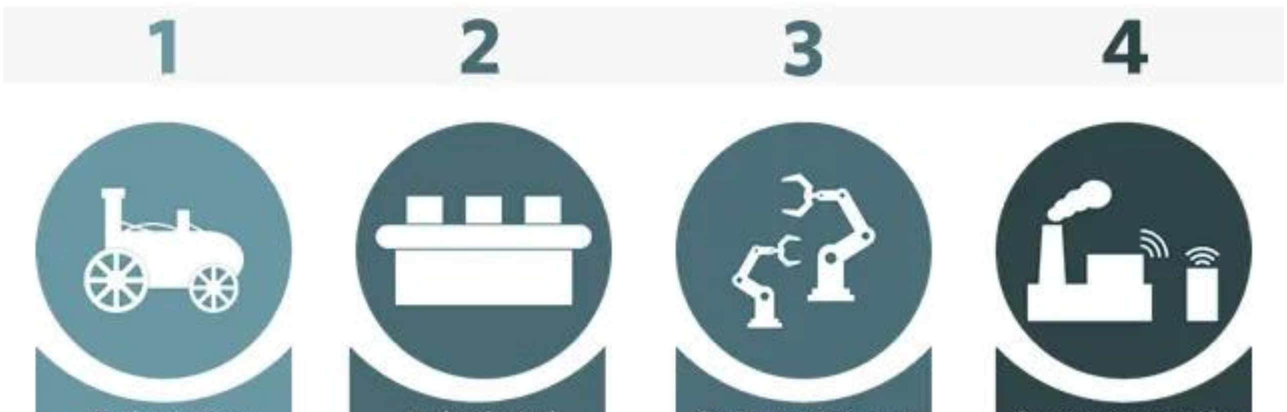
The intelligent networking of equipment and processes for industry using information and communication technologies is known as Industrie 4.0. (Plattform Industrie 4.0).

INDUSTRY 4.0 - the digital transformation

3rd platform, innovation accelerators, OT and manufacturing meet in transformation



FROM INDUSTRY 4.0 TO FOURTH INDUSTRIAL REVOLUTION



This allows products and manufacturing methods to be networked and 'communicate,' enabling new production methods, value creation, and real-time optimization. The capabilities required for smart factories are created through cyber-physical systems. These are the same features that we are familiar with from the Industrial Internet of Things, such as remote monitoring and track and trace, to name a few.

"A name for the current trend of automation and data exchange in manufacturing technologies, encompassing cyber-physical systems, the Internet of things, cloud computing and cognitive computing, and establishing the smart factory," according to the definition of Industry 4.0.

Industry 4.0 is a concept that has grown from a German endeavour to make the manufacturing industry more competitive ('Industrie 4.0') to a worldwide recognised term.

The terms "Industry 4.0" and "fourth industrial revolution" are frequently used interchangeably. It is distinguished by, among other things,

- There will be even more automation than there was during the third industrial revolution.

- Industrial IoT enables the bridging of the physical and digital worlds through cyber-physical systems.

- a trend away from a centralised industrial management system toward one in which smart products specify production steps

- Personalization and modification of products, as well as closed-loop data models and control systems

Through early stakeholder involvement and vertical and horizontal integration, the goal is to enable autonomous decision-making processes, real-time asset and process monitoring, and real-time connected value creation networks.

With reference architectures, standardisation, and even definitions in change, Industry 4.0 is a vision, policy, and concept in motion.

When compared to prior Industrial Revolutions, Industry 4.0 is unique. Industry 4.0, according to Schwab [6,] differs dramatically from the past three industrial revolutions due to its magnitude, scope, complexity, and transformation, which will be unlike any other revolution. According to Schwab, the revolution will be unlike anything humanity has ever known because it is not simply a continuation of the previous industrial revolution, but a new and separate revolution. The production of information, where people may come up with fresh information as well as generate new knowledge in the mining of information, is the first noteworthy difference between Industry 4.0 and previous revolutions. This is aided by the ability for a large number of individuals to be connected via mobile devices with high computing power, storage capacity, and unrestricted access to information.

In certain ways, the continuous collection and processing of data allows robots' intelligence to increase. Another feature that distinguishes Industry 4.0 from earlier industrial revolutions is that it will involve not only enormous technological advancements, but also a restructuring of current relationships in the manufacturing process.

Industry 4.0 also allows the manufacturing sector to join the information age by facilitating communication at all stages of the manufacturing process. Some academics predict that Industry 4.0 will result in new economic forms in numerous sectors of the economy, such as the sharing economy, in which services such as transportation, toys, and basketballs are shared.

Unit - 3

Adversarial Search and Games

3.1 Game Theory

Adversarial search is a sort of search that examines the problem that arises when we attempt to plan ahead of the world while other agents plan against us.

- In previous topics, we looked at search strategies that are purely associated with a single agent attempting to find a solution, which is often stated as a series of activities.
- However, in game play, there may be moments when more than one agent is looking for the same answer in the same search field.
- A multi-agent environment is one in which there are multiple agents, each of which is an opponent to the others and competes against them. Each agent must think about what another agent is doing and how that activity affects their own performance.
- Adversarial searches, also referred to as Games, are searches in which two or more players with conflicting goals try to find a solution in the same search space.
- The two fundamental variables that contribute in the modeling and solving of games in AI are a Search problem and a heuristic evaluation function.

Types of Games in AI

- **Perfect information:** A game with perfect information is one in which agents have complete visibility of the board. Agents can see each other's movements and have access to all game information. Examples include chess, checkers, go, and other games.
- **Imperfect information:** Tic-tac-toe, Battleship, blind, Bridge, and other games with incomplete information are known as such because the agents do not have all of the information and are unaware of what is going on.

- **Deterministic games:** Deterministic games have no element of chance and follow a strict pattern and set of rules. Examples include chess, checkers, go, tic-tac-toe, and other games.
- **Non-deterministic games:** Non-deterministic games are those with a number of unpredictable events and a chance or luck aspect. Dice or cards are used to introduce the element of chance or luck. These are unpredictably generated, and each action reaction is unique. Stochastic games are another name for these types of games.

Example: Backgammon, Monopoly, Poker, etc.

Key takeaway

Adversarial search is a type of search in which we look at the issue that develops when we try to plan ahead of the world while other agents plan against us.

Games are modeled as a Search problem and a heuristic evaluation function, which are the two primary variables that aid in the modeling and solving of games in AI.

Search for games

Searches in which two or more players with conflicting goals are trying to explore the same search space for the solution, are called adversarial searches, often known as Games.

Games are modeled as a Search problem and heuristic evaluation function, and these are the two main factors which help to model and solve games in AI.

Types of Games in AI:

	Deterministic	Chance Moves
Perfect information	Chess, Checkers, go, Othello	Backgammon, monopoly
Imperfect information	Battleships, blind, tic-tac-toe	Bridge, poker, scrabble, nuclear war

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

Example: Backgammon, Monopoly, Poker, etc.

Zero-Sum Game

- Zero-sum games are adversarial search which involves pure competition.
- In Zero-sum game each agent's gain or loss of utility is exactly balanced by the losses or gains of utility of another agent.
- One player of the game try to maximize one single value, while other player tries to minimize it.
- Each move by one player in the game is called as ply.
- Chess and tic-tac-toe are examples of a Zero-sum game.

Zero-sum game: Embedded thinking

The Zero-sum game involved embedded thinking in which one agent or player is trying to figure out:

- What to do.
- How to decide the move
- Needs to think about his opponent as well
- The opponent also thinks what to do

Each of the players is trying to find out the response of his opponent to their actions. This requires embedded thinking or backward reasoning to solve the game problems in AI.

Formalization of the problem:

A game can be defined as a type of search in AI which can be formalized of the following elements:

- Initial state: It specifies how the game is set up at the start.

- Player(s): It specifies which player has moved in the state space.
- Action(s): It returns the set of legal moves in state space.
- Result(s, a): It is the transition model, which specifies the result of moves in the state space.
- Terminal-Test(s): Terminal test is true if the game is over, else it is false at any case. The state where the game ends is called terminal states.
- Utility(s, p): A utility function gives the final numeric value for a game that ends in terminal states s for player p. It is also called payoff function. For Chess, the outcomes are a win, loss, or draw and its payoff values are +1, 0, ½ . And for tic-tac-toe, utility values are +1, -1, and 0.

Key takeaway

Adversarial search is a sort of search that examines the problem that arises when we attempt to plan ahead of the world while other agents plan against us.

3.2 Optimal Decisions in Games

Optimal decision in multiplayer games

A dependent strategy, which defines MAX's move in the initial state, then MAX's movements in the states resulting from every probable response by MIN (the opponent), and so on, is the optimal solution in adversarial search.

One move deep: If a game ends after one MAX and MIN move each, we say the tree is one move deep, with two half-moves, each referred to as a ply.

Minimax value: The node's minimax value is the utility (for MAX) of being in the corresponding state, assuming that both players play optimally from there until the game ends. The minimax value of a terminal state determines its utility.

Given a game tree, the optimal strategy can be computed using the minimax value of each node, i.e., MINIMAX (n).

MAX likes to get to the highest value state, whereas MIN prefers to go to the lowest value state.

$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if TERMINAL} - \text{TEST}(s) \\ \max_{a \in \text{Action}(s)} \text{MINIMAX (RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MAX} \\ \min_{a \in \text{Action}(s)} \text{MINIMAX (RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MIN} \end{cases}$$

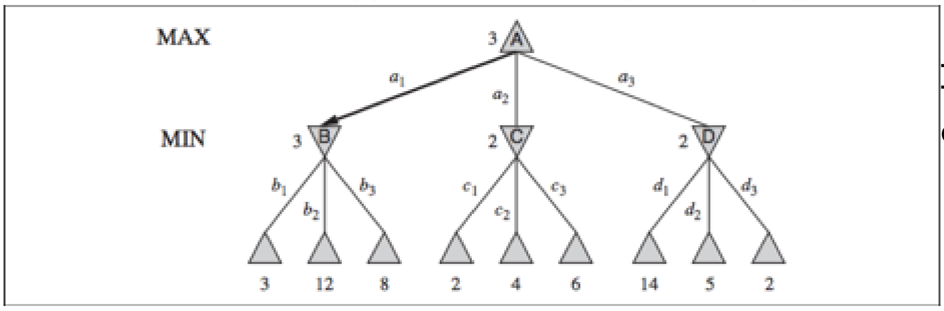


Fig 1: Two ply game tree

Multiplayer games

Each node's single value must be replaced by a vector of values. A vector VA, VB, VC> is associated with each node in a three-player game with players A, B, and C, for example.

From each actor's perspective, this vector indicates the utility of a terminal circumstance.

In nonterminal states, the backed-up value of a node n is always the utility vector of the successor state with the highest value for the player choosing at n.

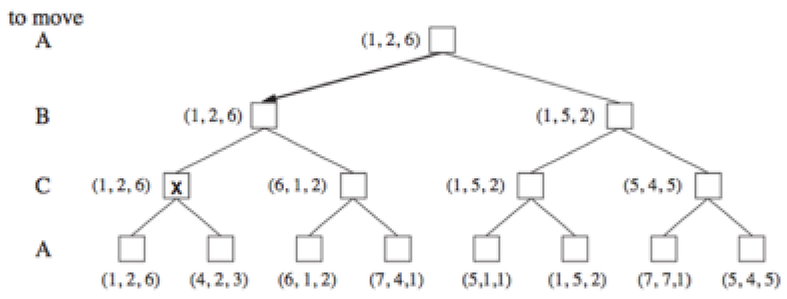


Fig 2: Three piles of a game tree with three players (A, B, C)

Key takeaway

A dependent strategy, which defines MAX's move in the initial state, then MAX's movements in the states resulting from every probable response by MIN (the opponent), and so on, is the optimal solution in adversarial search.

3.3 Heuristic Alpha–Beta Tree Search

- Alpha-beta pruning is a modified version of the minimax algorithm. It is an optimization technique for the minimax algorithm.

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

- Alpha-beta pruning can be applied at any depth of a tree, and sometimes it not only prune the tree leaves but also entire sub-tree.

- The two-parameter can be defined as:

    1. **Alpha:** The best (highest-value) choice we have found so far at any point along the path of Maximizer. The initial value of alpha is **-∞**.
    2. **Beta:** The best (lowest-value) choice we have found so far at any point along the path of Minimizer. The initial value of beta is **+∞**.

- The Alpha-beta pruning to a standard minimax algorithm returns the same move as the standard algorithm does, but it removes all the nodes which are not really affecting the final decision but making algorithm slow. Hence by pruning these nodes, it makes the algorithm fast.

Condition for Alpha-beta pruning:

The main condition which required for alpha-beta pruning is:

$\alpha >= \beta$

Pseudo-code for Alpha-beta Pruning:

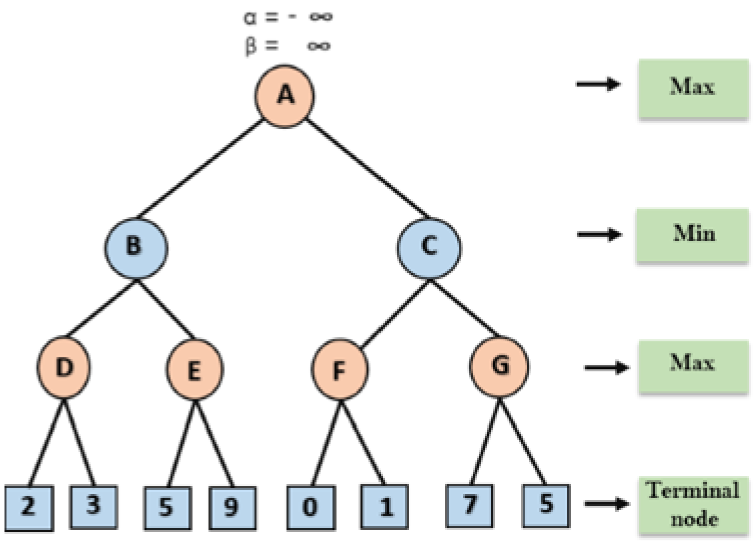
```
Function minimax(node, depth, alpha, beta, maximizingPlayer) is
    If depth ==0 or node is a terminal node then
        Return static evaluation of node
    If MaximizingPlayer then      // for Maximizer Player
        MaxEva= -infinity
        For each child of node do
            Eva= minimax(child, depth-1, alpha, beta, False)
            MaxEva= max(maxEva, eva)
            Alpha= max(alpha, maxEva)
            If beta<=alpha
                Break
        Return maxEva
    Else                     // for Minimizer player
        MinEva= +infinity
        For each child of node do
            Eva= minimax(child, depth-1, alpha, beta, true)
```

17. MinEva= min(minEva, eva)
18. Beta= min(beta, eva)
19. If beta<=alpha
20. Break
21. Return minEva

Working of Alpha-Beta Pruning:

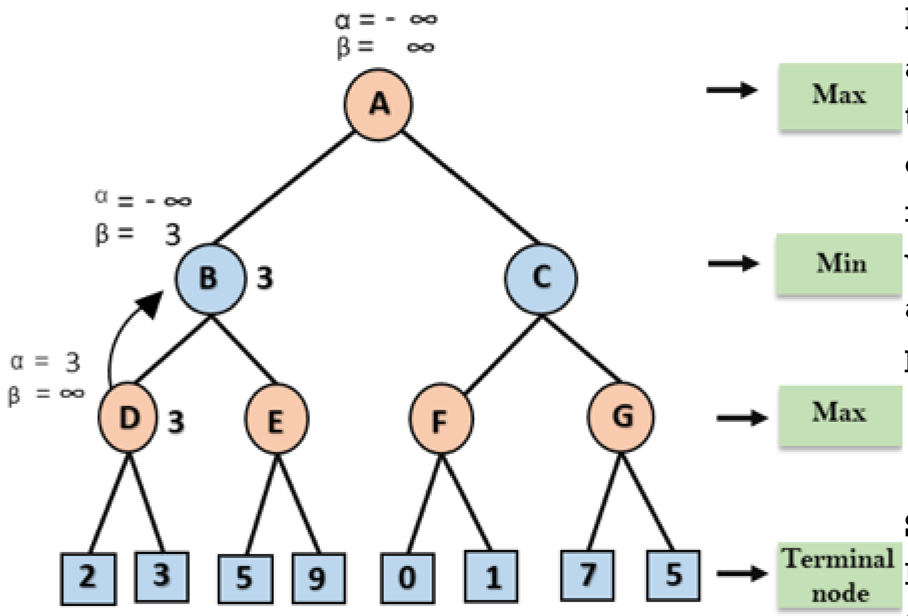
Let's take an example of two-player search tree to understand the working of Alpha-beta pruning

Step 1: At the first step the, Max player will start first move from node A where α= -∞ and β= +∞, these value of alpha and beta passed down to node B where again α= -∞ and β= +∞, and Node B passes the same value to its child D.



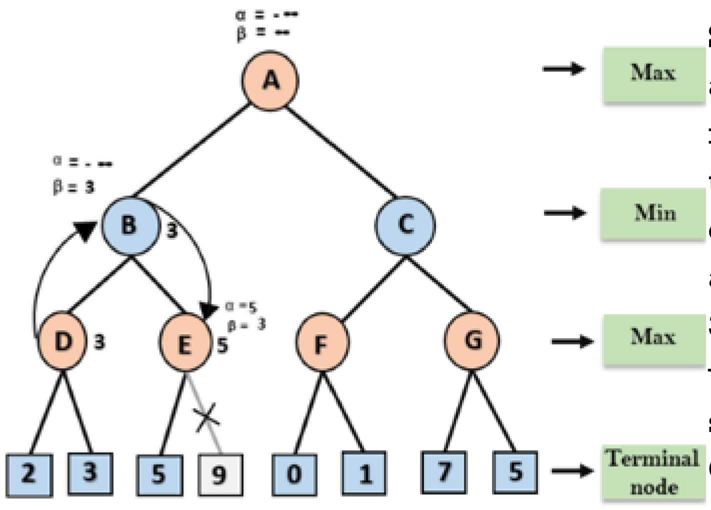
Step 2: At Node D, the value of α will be calculated as its turn for Max. The value of α is compared with firstly 2 and then 3, and the max (2, 3) = 3 will be the value of α at node D and node value will also 3.

Step 3: Now algorithm backtrack to node B, where the value of β will change as this is a turn of Min, Now β= +∞, will compare with the available subsequent nodes value, i.e. min (∞, 3) = 3, hence at node B now α= -∞, and β= 3.



In the next step, algorithm traverse the next successor of Node B which is node E, and the values of α= -∞, and β= 3 will also be passed.

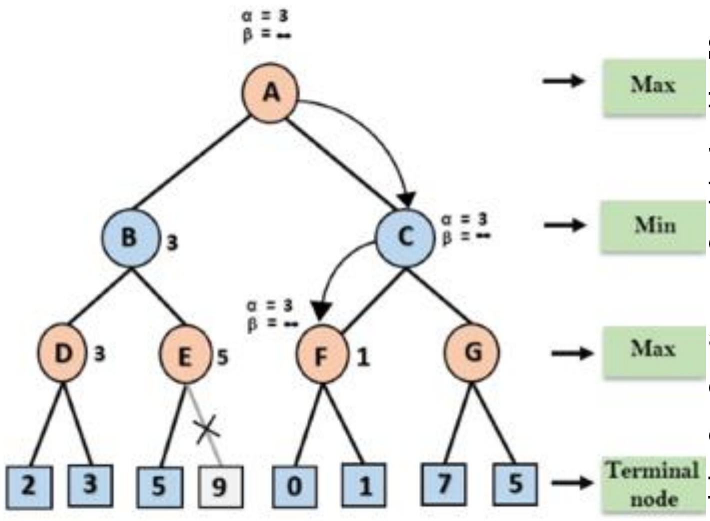
Step 4: At node E, Max will take its turn, and the value of alpha will change. The current value of alpha will be compared with 5, so max (-∞, 5) = 5, hence at node E α= 5 and β= 3, where α>=β, so the right successor of E will be pruned, and algorithm will not traverse it, and the value at node E will be 5.



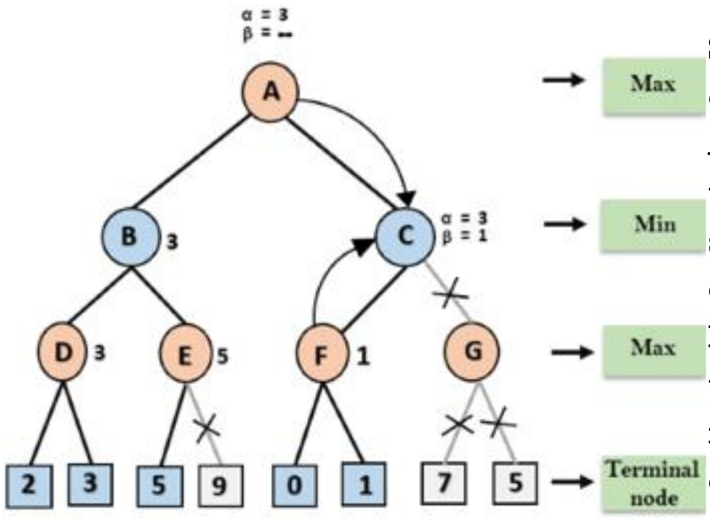
Step 5: At next step, algorithm again backtrack the tree, from node B to node A. At node A, the value of alpha will be changed the maximum available value is 3 as max (-∞, 3)= 3, and β= +∞, these two values now passes to right successor of A which is Node C.

At node C, α=3 and β= +∞, and the same values will be passed on to node F.

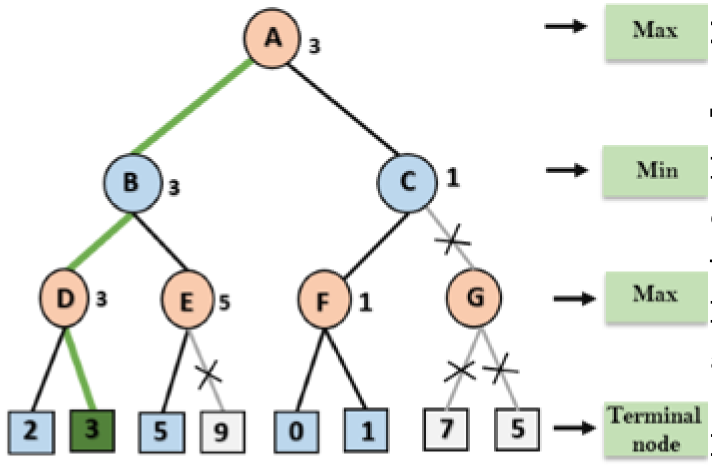
Step 6: At node F, again the value of α will be compared with left child which is 0, and max(3,0)= 3, and then compared with right child which is 1, and max(3,1)= 3 still α remains 3, but the node value of F will become 1.



Step 7: Node F returns the node value 1 to node C, at C α= 3 and β= +∞, here the value of beta will be changed, it will compare with 1 so min (∞, 1) = 1. Now at C, α=3 and β= 1, and again it satisfies the condition α>=β, so the next child of C which is G will be pruned, and the algorithm will not compute the entire sub-tree G.



Step 8: C now returns the value of 1 to A here the best value for A is max (3, 1) = 3. Following is the final game tree which is the showing the nodes which are computed and nodes which has never computed. Hence the optimal value for the maximizer is 3 for this example.



Move Ordering in Alpha-Beta pruning:

The effectiveness of alpha-beta pruning is highly dependent on the order in which each node is examined. Move order is an important aspect of alpha-beta pruning.

It can be of two types:

- **Worst ordering:** In some cases, alpha-beta pruning algorithm does not prune any of the leaves of the tree, and works exactly as minimax algorithm. In this case, it also consumes more time because of alpha-beta factors, such a move of pruning is called worst ordering. In this case, the best move occurs on the right side of the tree. The time complexity for such an order is O(b^m).

- **Ideal ordering:** The ideal ordering for alpha-beta pruning occurs when lots of pruning happens in the tree, and best moves occur at the left side of the tree. We apply DFS hence it first search left of the tree and go deep twice as minimax algorithm in the same amount of time. Complexity in ideal ordering is O($b^{m/2}$).

Rules to find good ordering:

Following are some rules to find good ordering in alpha-beta pruning:

- Occur the best move from the shallowest node.
- Order the nodes in the tree such that the best nodes are checked first.
- Use domain knowledge while finding the best move. Ex: for Chess, try order: captures first, then threats, then forward moves, backward moves.
- We can bookkeep the states, as there is a possibility that states may repeat.

Key points about alpha-beta pruning:

● The Max player will only update the value of alpha.

● The Min player will only update the value of beta.

● While backtracking the tree, the node values will be passed to upper nodes instead of values of alpha and beta.

● We will only pass the alpha, beta values to the child nodes.

3.4 Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) is an artificial intelligence (AI) search strategy (AI). It's a probabilistic and heuristic-driven search method that blends traditional tree search implementations with reinforcement learning machine learning principles.

There's always the risk that the current best action isn't the most optimal action in tree search. In such instances, the MCTS algorithm comes in handy since it continues to assess other options occasionally during the learning phase by executing them instead of the currently perceived optimal approach. The "exploration-exploitation trade-off" is what it's called. It takes advantage of the actions and tactics that have been determined to be the most effective up to this point, but it must also continue to investigate the local space of alternative decisions to see if they may replace the present best.

Exploration aids in the exploration and discovery of the tree's unexplored areas, which may lead to the discovery of a more ideal path. To put it another way, inquiry broadens the tree's width rather than its depth. Exploration can help ensure that MCTS isn't missing out on any possibly better options. However, in cases involving a high number of steps or repeats, it quickly becomes inefficient. To avoid this, exploitation is used to balance things out.

Exploitation remains focused on a single path with the highest estimated value. This is a greedy technique, as it will increase the depth of the tree rather than its breadth. In basic terms, when applied to trees, the UCB formula helps to balance the exploration-exploitation trade-off by regularly examining relatively unknown nodes of the tree and discovering possibly more optimum paths than the one it is currently exploiting.

For this characteristic, MCTS becomes particularly useful in making optimal decisions in Artificial Intelligence (AI) problems.

Monte Carlo Tree Search (MCTS) algorithm:

The building blocks of the search tree in MCTS are nodes. These nodes are created using the results of several simulations. Monte Carlo Tree Search is divided into four steps: selection, expansion, simulation, and back propagation. Each of these steps is described in greater depth below:

Selection - The MCTS method uses a specific technique to explore the current tree from the root node. To choose the nodes with the highest estimated value, the technique employs an evaluation function. To traverse the tree, MCTS uses the Upper Confidence Bound (UCB) formula applied to trees as the approach in the selection process. It strikes a balance between exploration and exploitation. A node is chosen during tree traversal based on some parameters that return the highest value. The parameters are defined by the formula below, which is commonly used for this purpose.

Where;

Si = value of a node i

Xi = empirical mean of a node i

C = a constant

t = total number of simulations

The child node that returns the biggest value from the above equation will be selected when traversing a tree during the selection phase. When a child node that is also a leaf node is detected during traversal, the MCTS enters the expansion stage.

Expansion - A new child node is added to the tree to the node that was optimally attained during the selection process in this process.

Simulation - A simulation is carried out in this process by selecting moves or strategies until a desired result or condition is obtained.

Backpropagation - The remaining tree must be modified after finding the value of the newly inserted node. As a result, the backpropagation process is carried out, in which the new node propagates back to the root node. The amount of simulations kept in each node is increased during the procedure. The number of wins is also increased if the new node's simulation results in a win.

The actions outlined above can be visualised using the graphic below:

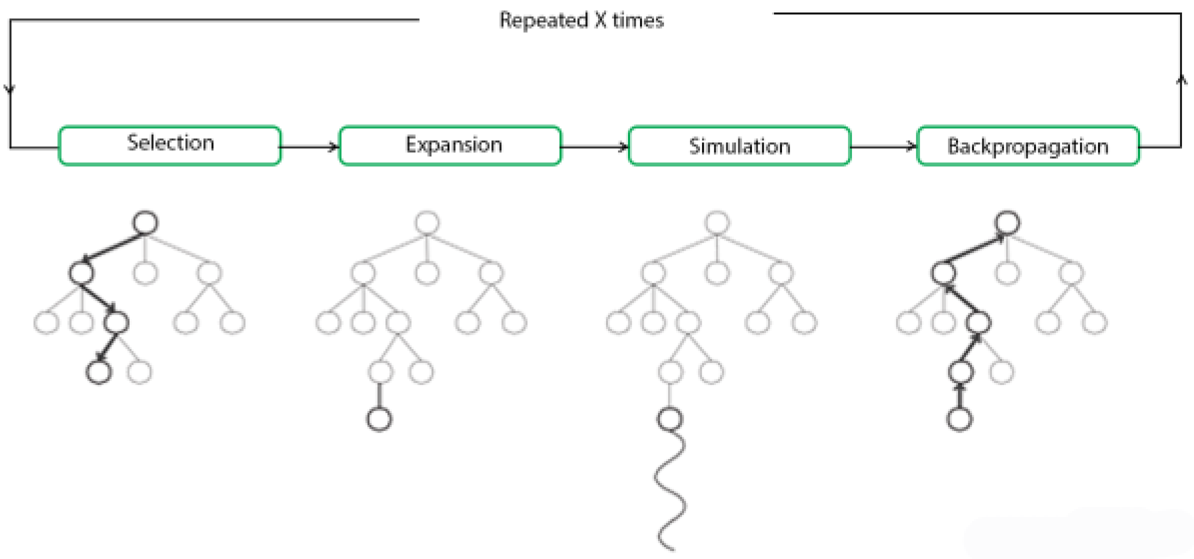


Fig 3: Outlined visualised graphics

These algorithms are especially beneficial in turn-based games like Tic Tac Toe, Connect 4, Checkers, Chess, Go, and others where there is no element of chance in the game rules. Artificial Intelligence Programs, such as AlphaGo, have lately employed this to compete against the world's best Go players. However, its use isn't confined to video games. It can be applied to any situation with state-action pairings and simulations to predict outcomes.

Key takeaway

Monte Carlo Tree Search (MCTS) is an artificial intelligence (AI) search strategy (AI). It's a probabilistic and heuristic-driven search method that blends traditional tree search implementations with reinforcement learning machine learning principles.

3.5 Stochastic Games

Stochastic games are simulations of dynamic interactions in which the environment changes in response to the activities of the participants. "In a stochastic game, the play progresses by steps from position to position, according to transition probabilities determined jointly by the two players," Shapley writes.

A group of participants participates in a stochastic game. At each stage of the game, the action takes place in a specific state (or position, in Shapley's terminology), and each player selects an action from a list of options. The stage payoff that each player receives is determined by the collection of actions that the players choose, as well as the current state, as well as a probability distribution for the subsequent state that the game will visit.

Stochastic games apply von Neumann's model of strategic-form games to dynamic settings in which the environment varies in response to the players' decisions. They also use the Markov decision problem model, which was created by numerous RAND Corporation researchers in 1949–1952, to competitive scenarios with multiple decision makers.

The complexity of stochastic games arises from the fact that the players' decisions have two, sometimes opposing, consequences. First, the players' actions, in combination with the present state, determine the immediate payment that each player receives. Second, the present state and the players' activities influence the next state selection, which influences future reward possibilities. Each player, in particular, must balance these factors when deciding on his actions, which might be a difficult decision.

Although this dichotomy is also present in one-player sequential decision problems, the presence of additional players who maximize their own goals adds complexity to the analysis of the situation.

Two player Games

For modelling and analysis of discrete systems functioning in an uncertain (adversarial) environment, stochastic two-player games on directed graphs are commonly employed. As vertices, a system's and its environment's possible configurations are represented, and transitions correspond to the system's, its environment's, or "nature's" actions. An infinite path in the graph corresponds to a run of the system. As a result, a system and its environment can be viewed as two hostile players, with one player (the system) aiming to maximise the probability of "good" runs and the other player (the environment) aiming to maximise the chance of "bad" runs.

Although there may be an equilibrium value for this probability in many circumstances, there may not be optimal solutions for both parties.

We go over some of the fundamental notions and algorithmic concerns that have been studied in this field, as well as some long-standing outstanding difficulties. Then we'll go through a few recent findings.

Application

Economic theory, evolutionary biology, and computer networks all use stochastic games. They are generalisations of repeated games that correspond to the unique case of a single state.

Key takeaway

Stochastic games are simulations of dynamic interactions in which the environment changes in response to the activities of the participants. "In a stochastic game, the play progresses by steps from position to position, according to transition probabilities determined jointly by the two players," Shapley writes.

3.6 Partially Observable Games

A partially observable stochastic game (POSG) is a tuple

{I, S, $\{b^0\}$, $\{A_i\}$, $\{O_i\}$,P, $\{R_i\}$i, where

- I is a finite set of agents (or controllers) indexed 1, . . . , n.
- S is a finite set of states.
- $b^0 \in \Delta(S)$ represents the initial state distribution
- Ai is a finite set of actions available to agent i and A~ = ×i∈IAi is the set of joint actions (i.e., action profiles), where ~a = ha1, . . . , ani denotes a joint action
- O_i is a finite set of observations for agent i and O~ = ×i∈IOi is the set of joint observations, where ~o = ho1, . . . , oni denotes a joint observation
- P is a set of Markovian state transition and observation probabilities, where P(s 0 , ~o|s, ~a) denotes the probability that taking joint action ~a in state s results in a transition to state s 0 and joint observation ~o
- R_i : S × A → < ~ is a reward function for agent i

A game unfolds across a finite or infinite number of stages, with the number of stages referred to as the game's horizon. We focus on finite-horizon POSGs in this study; some of the issues of solving the infinite-horizon situation are highlighted at the end. At each level, all agents choose an action at the same time and are rewarded and observed. Each agent's goal is to maximise the expected sum of prizes it will receive during the game. The reward functions determine whether agents compete or cooperate in their quest for a reward. A decentralised partially observable Markov decision process is one in which the agents share the same reward function.

3.7 Limitations of Game Search Algorithms

- The process of obtaining the goal is slower due to the large branching factor.
- The engine's performance and efficiency suffer as a result of evaluating and searching all conceivable nodes and branches.
- Both players have an excessive number of options from which to choose.
- It is impossible to investigate the complete tree if there is a time and space constraint.
- The biggest disadvantage of the minimax algorithm is that it becomes extremely slow while playing complex games like chess or go. This style of game contains a lot of branching, and the player has a lot of options to choose from.

The algorithm, however, can be enhanced by Alpha-Beta Pruning.

3.8 Constraint Satisfaction Problems (CSP)

A **constraint satisfaction problem** (CSP) consists of

- A set of variables,
- A domain for each variable, and
- A set of constraints.

The aim is to choose a value for each variable so that the resulting possible world satisfies the constraints; we want a model of the constraints.

A finite CSP has a finite set of variables and a finite domain for each variable. Many of the methods considered in this chapter only work for finite CSPs, although some are designed for infinite, even continuous, domains.

The multidimensional aspect of these problems, where each variable can be seen as a separate dimension, makes them difficult to solve but also provides structure that can be exploited.

Given a CSP, there are a number of tasks that can be performed:

- Determine whether or not there is a model.
- Find a model.
- Find all of the models or enumerate the models.
- Count the number of models.
- Find the best model, given a measure of how good models are.
- Determine whether some statement holds in all models.

This chapter mostly considers the problem of finding a model. Some of the methods can also determine if there is no solution. What may be more surprising is that some of the methods can find a model if one exists, but they cannot tell us that there is no model if none exists.

CSPs are very common, so it is worth trying to find relatively efficient ways to solve them. Determining whether there is a model for a CSP with finite domains is NP-hard and no known algorithms exist to solve such problems that do not use exponential time in the worst case. However, just because a problem is NP-hard does not mean that all instances are difficult to solve. Many instances have structure that can be exploited.

Key takeaway

Constraint satisfaction problems (CSPs) are mathematical problems in which the state of a group of objects must meet a set of constraints or limitations. Constraint satisfaction methods are used to solve CSPs, which express the entities in a problem as a homogeneous set of finite constraints over variables.

3.9 Constraint Propagation: Inference in CSPs

The constraints are used in a number of inference approaches to determine which variable/value combinations are consistent and which are not. Node, arc, path, and k-consistent are examples of these.

Constraint propagation: Using constraints to reduce the number of permissible values for one variable, which then reduces the number of legal values for another variable, and so on.

Local consistency: If we treat each variable as a node in a graph and each binary constraint as an arc, the act of enforcing local consistency in each area of the graph eliminates inconsistent values across the board.

Different types of local consistency exist:

Node consistency

- If all of the values in the variable's domain fulfil the variable's unary constraint, the variable is node-consistent (a node in the CSP network).
- If every variable in a network is node-consistent, we call it a node-consistent network.

Arc consistency

● If every value in a variable's domain fulfils the variable's binary constraints, the variable is arc-consistent in a CSP.

● If there is some value in the domain D_j that satisfies the binary constraint on the arc for every value in the current domain Di, then Xi is arc-consistent with regard to another variable X_j (X_i, X_j).

● If every variable in a network is arc-consistent with every other variable, it is said to be arc-consistent.

● Using the arcs, arc consistency tightens the domains (unary constraint) (binary constraints).

AC-3 algorithm

```
Function AC-3(csp) returns false if an inconsistency is found and true otherwise
Inputs: csp, a binary  CSP with components (X, D, C)
Local variables: queue, a queue of arcs, initially all the arcs in csp
While queue is not empty do
  $(X_i, X_j) \leftarrow \text{REMOVE} - \text{FIRST}(\text{queue})$
If REVISE(csp, $X_{ij}, X_j$) then
If  size of $D_i = 0$ then return false
For each $X_k$ in $X_i$, NEIGHBORS - $\{X_j\}$ do
Add $(X_k, X_i)$ to queue
Return true
Function REVISE (csp, $X_i, X_j$) returns true iff we revise the domain of $X_i$
Revised $\leftarrow$ false
For each x in $D_i$ do
```

If no value y in D_j allows (x,y) to satisfy the constraint between X_i and X_j **then**

Delete x from D_i

Revised $\leftarrow$ true

Return revised

This is arc consistency algorithm

- AC-3 keeps an arc queue that initially contains all of the arcs in the CSP.
- Then, from the queue, AC-3 selects an arbitrary arc (Xi, Xj) and makes Xi arc-consistent with respect to Xj.
- If Di remains unchanged, continue on to the next arc; if Di is revised, add any arcs (Xk, Xi) where Xk is a neighbour of Xi to the queue.
- If Di is reduced to zero, the entire CSP will collapse since there is no consistent answer.
- Otherwise, keep checking and removing values from variable domains until there are no more arcs in the queue.
- As a result, an arc-consistent CSP with smaller domains has the same solutions as the original.

The complexity of AC-3:

Assume a CSP with n variables, each with domain size at most d, and with c binary constraints (arcs). Checking consistency of an arc can be done in O(d2) time, total worst-case time is O(cd3)

Path consistency

● A two-variable set Xi, Xj is path-consistent with regard to a third variable Xm if there is an assignment to Xm that fulfils the constraints on Xi, Xm, and Xm, Xj for every assignment Xi = a, Xj = b consistent with the constraint on Xi, Xj.

● The binary restrictions are tightened by path consistency, which uses implicit constraints inferred from triples of variables.

K-consistency

● A CSP is k-consistent if a consistent value can always be assigned to any kth variable for any collection of k-1 variables and for any consistent assignment to those variables.

● 1-consistency = node consistency; 2-consistency = arc consistency; 3-consistency = path consistency.

● If a CSP is k-consistent and also (k – 1)-consistent, (k – 2)-consistent,... All the way down to 1-consistent, it is very k-consistent.

● We can ensure that if we take a CSP with n nodes and make it strongly n-consistent, we will discover a solution in time $O(n^2d)$. However, in the worst situation, the procedure for establishing n-consistency must take time exponential in n and demand space exponential in n.

Global constraints

A global constraint is one that applies to an infinite number of variables (but not necessarily all variables). Special-purpose algorithms that are more efficient than general-purpose approaches can manage global constraints.

1) inconsistency detection for Alldiff constraints

A straightforward formula is as follows: Remove any variable with a singleton domain from the constraint and delete its value from the domains of the remaining variables. Repeat for as long as singleton variables exist. An inconsistency has been discovered if an empty domain is formed at any stage or if there are more variables than domain values left.

Applying arc consistency to an analogous collection of binary constraints is sometimes more effective than using a simple consistency approach for a higher-order constraint.

2) inconsistency detection for resource constraint (the atmost constraint)

● By calculating the total of the current domains' minimum values, we may find inconsistencies.

● e.g., Atmost(10, P1, P2, P3, P4): a total of no more than 10 workers is assigned.

● The Atmost constraint cannot be met if each variable has the domain 3, 4, 5, 6.

● We can ensure consistency by eliminating any domain's maximum value if it differs from the minimum values of the other domains.

● For example, if the domains of each variable in the example are 2, 3, 4, 5, 6, the values 5 and 6 can be removed from each domain.

3) inconsistency detection for bounds consistent

Domains are represented by upper and lower limits and handled via bounds propagation for big resource-limited issues with integer values.

e.g - In an airline-scheduling problem, imagine there are two flights F1 and F2, each with a capacity of 165 and 385 passengers. The initial domains for passenger numbers on each aircraft are

D1 = [0, 165] and D2 = [0, 385].

Assume we also have the limitation that the two flights must each carry 420 passengers: F1 + F2 = 420. We reduce the domains to, by propagating bounds constraints.

D1 = [35, 165] and D2 = [255, 385].

If there is a value of Y that fulfils the constraint between X and Y for every variable Y, and for both the lower-bound and upper-bound values of X, a CSP is bound consistent.

Key takeaway

The constraints are used in a number of inference approaches to determine which variable/value combinations are consistent and which are not. Node, arc, path, and k-consistent are examples of these.

3.10 Backtracking Search for CSPs

CSPs are frequently solved via backtracking search, a type of depth-first search. Inference and search can be combined.

Commutativity: All CSPs are commutative. A issue is commutative if the sequence in which a series of activities is carried out has no bearing on the outcome.

Backtracking search: A depth-first search that assigns values to one variable at a time and then backtracks when there are no more legal values to assign to that variable.

The backtracking algorithm selects an unassigned variable each time and then attempts all of the values in that variable's domain in order to discover a solution. If an inconsistency is found, BACKTRACK fails, leading the previous call to try a different value.

There is no need to provide a domain-specific initial state, action function, transition model, or goal test to BACKTRACKING-SEARCH.

BACKTRACKING-SEARCH maintains only one representation of a state and modifies it rather than producing new ones.

```
Function BACTRACKING-SEARCH(csp) returns a solution, or failure
Return BACTRACK({}, csp)
Var←SELECT-UNASSIGNED-VARIABLE(csp)
For each value in ORDER-DOMAIN-VALUES(var,assignement,csp) do
If value is consistent with assignment then
Add{var=value} to assignment
Inferences ← INFERENCE(csp,var,value)
If inferences ≠ failure then
Add inferences to assignment
Result ← BACKTRACK (assignment, csp)
If result ≠ failure then
Return result
Remove {var=value} and inferences from assignment
Return failure
```

This is simple backtracking algorithm for CSP

Address the following questions to efficiently answer CSPs without domain-specific knowledge:

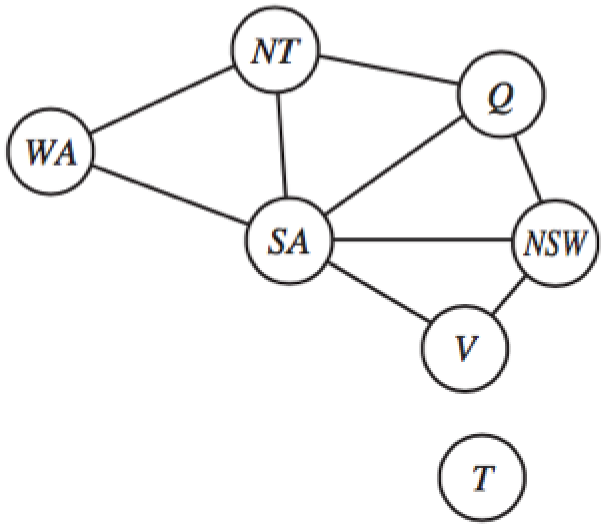
Which variable should be assigned next, according to the SELECT-UNASSIGNED-VARIABLE function?

What order should the values of the ORDER-DOMAIN-VALUES function be tested in?

2)INFERENCE FUNCTION: What inferences should be made at each stage of the search?

3)Can the search avoid repeating its failure when it comes across an assignment that violates a constraint?

1. Variable and value ordering



SELECT-UNASSIGNED-VARIABLE

Variable selection—fail-first

Minimum-remaining-values (MRV) heuristic: The concept of selecting the variable having the smallest number of "legal" values. The "most constrained variable" or "fail-first" heuristic selects a variable that is most likely to cause a failure soon, allowing the search tree to be pruned. If a variable X has no more permissible values, the MRV heuristic will select X, and failure will be noticed instantly, avoiding the need for fruitless searches through other variables.

For example, since there is only one potential value for SA after WA=red and NT=green, it makes logical to assign SA=blue next rather than Q.

Degree heuristic: By selecting the variable that is implicated in the greatest number of constraints on other unassigned variables, the degree heuristic seeks to lower the branching factor on future decisions.

SA, for example, has the highest degree of 5; the other variables have degrees of 2 or 3; and T has a degree of 0.

ORDER-DOMAIN-VALUES

Value selection—fail-last

If we're looking for all possible solutions to a problem (rather than simply the first one), the order doesn't matter.

Least-constraining-value heuristic: chooses the value that eliminates the fewest options for the constraint graph's surrounding variables. (Leave as much freedom as possible for subsequent variable assignments.)

For example, we've created a partial assignment with WA=red and NT=green, and Q is our next option. Because blue eliminates the final lawful value available for Q's neighbour, SA, who prefers red to blue, it is a negative decision.

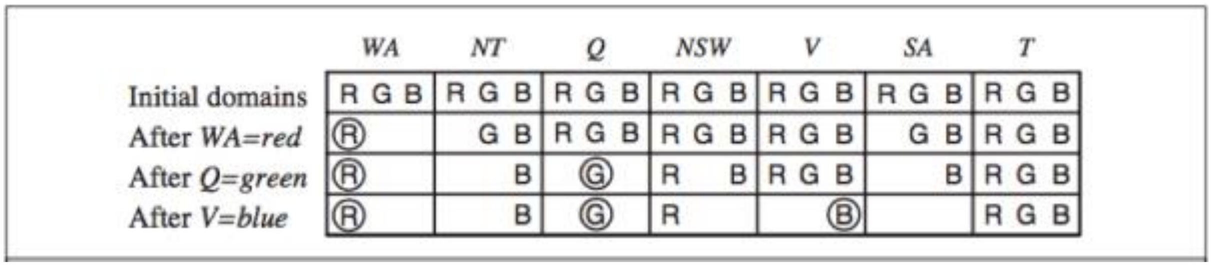
In a backtracking search, the minimum-remaining-values and degree heuristics are domain-independent approaches for determining which variable to choose next. The least-constraining-value heuristic aids in determining which value for a given variable to try first.

2. Interleaving search and inference

INFERENCE

Forward chaining - [One of the simplest forms of inference is forward checking.] When a variable X is assigned, the forward-checking method creates arc consistency for it: delete from Y's domain any value that is inconsistent with the value chosen for X for each unassigned variable Y that is associated to X by a constraint.

If we've previously done arc consistency as a preprocessing step, there's no reason to do forward checking.



	WA	*NT*	*Q*	*NSW*	*V*	*SA*	*T*
Initial domains	R G B	R G B	R G B	R G B	R G B	R G B	R G B
After *WA=red*	Ⓡ	G B	R G B	R G B	R G B	G B	R G B
After *Q=green*	Ⓡ	B	Ⓖ	R B	R G B	B	R G B
After *V=blue*	Ⓡ	B	Ⓖ	R	Ⓑ		R G B

Fig 4: Progress of a map coloring search with forward checking

Advantage: Combining the MRV heuristic with forward checking will make the search more effective for many issues.

Disadvantages: Forward checking only makes the current variable arc-consistent, not all the other variables.

MAC (Maintaining Arc Consistency) algorithm: [Detect this inconsistency more effectively than forward checking.] The INFERENCE function invokes AC-3 once a variable Xi is allocated a value, but instead of a queue of all arcs in the CSP, we only start with the arcs(Xj, Xi) for those Xj that are unassigned variables that are neighbours of Xi. The call to AC-3 then proceeds as usual, with the exception that if any variable's domain is reduced to the empty set, the call to AC-3 fails and we know to retrace immediately.

3. Intelligent backtracking

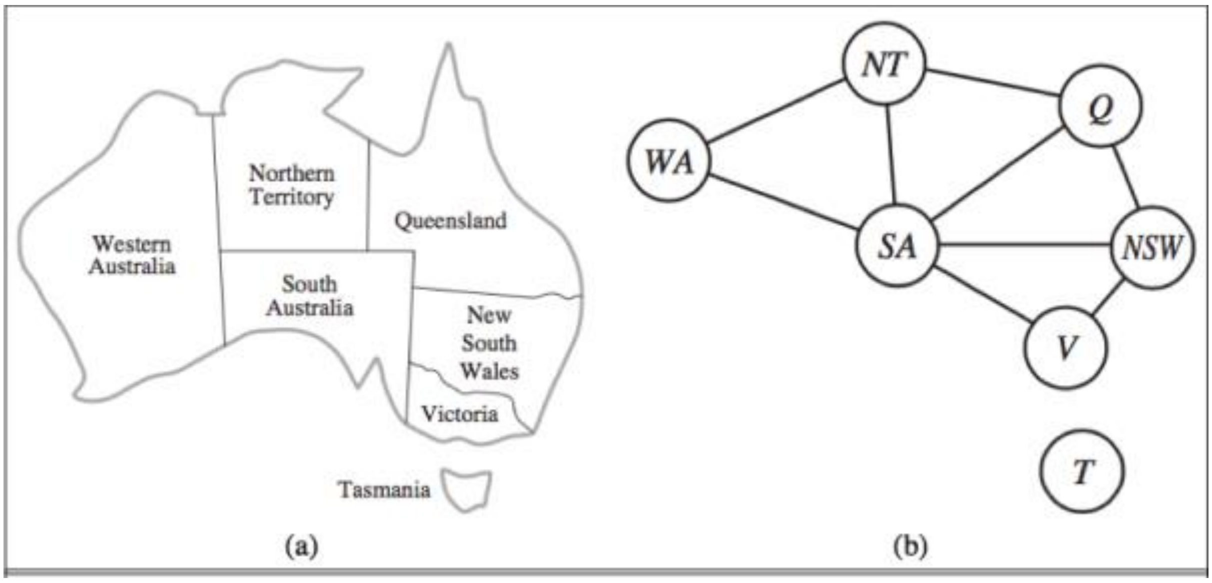


Fig 5: (a) Principal states and territories of Australia. Colouring this map can be viewed as CSP. (b) the map coloring problem represented as a constraint graph

Chronological backtracking: Figure shows the BACKGRACING-SEARCH. If a branch of the search fails, go back to the previous variable and try a different value for it. (The most recent point of decision is revisited.)

For example, consider the partial assignment Q=red, NSW=green, V=blue, T=red.

When we attempt the next variable, SA, we notice that every value breaks a constraint.

We go back to T and try a different hue, but it doesn't work.

Intelligent backtracking: Return to the variable that caused one of the possible values of the following variable (e.g. SA) to be impossible.

A collection of assignments for a variable that are in conflict with some value for that variable.

(For example, the conflict set for SA is Q=red, NSW=green, V=blue.)

Backtracks to the most recent assignment in the conflict set using the back jumping approach.

(e.g – Back jumping, for example, would leap over T and try a different value for V.)

Forward checking can supply the conflict set with no extra work.

Add X=x to Y's conflict set whenever forward checking based on an assignment X=x deletes a value from Y's domain.

If the last value in Y's domain is removed, the assignments in Y's conflict set are added to X's conflict set.

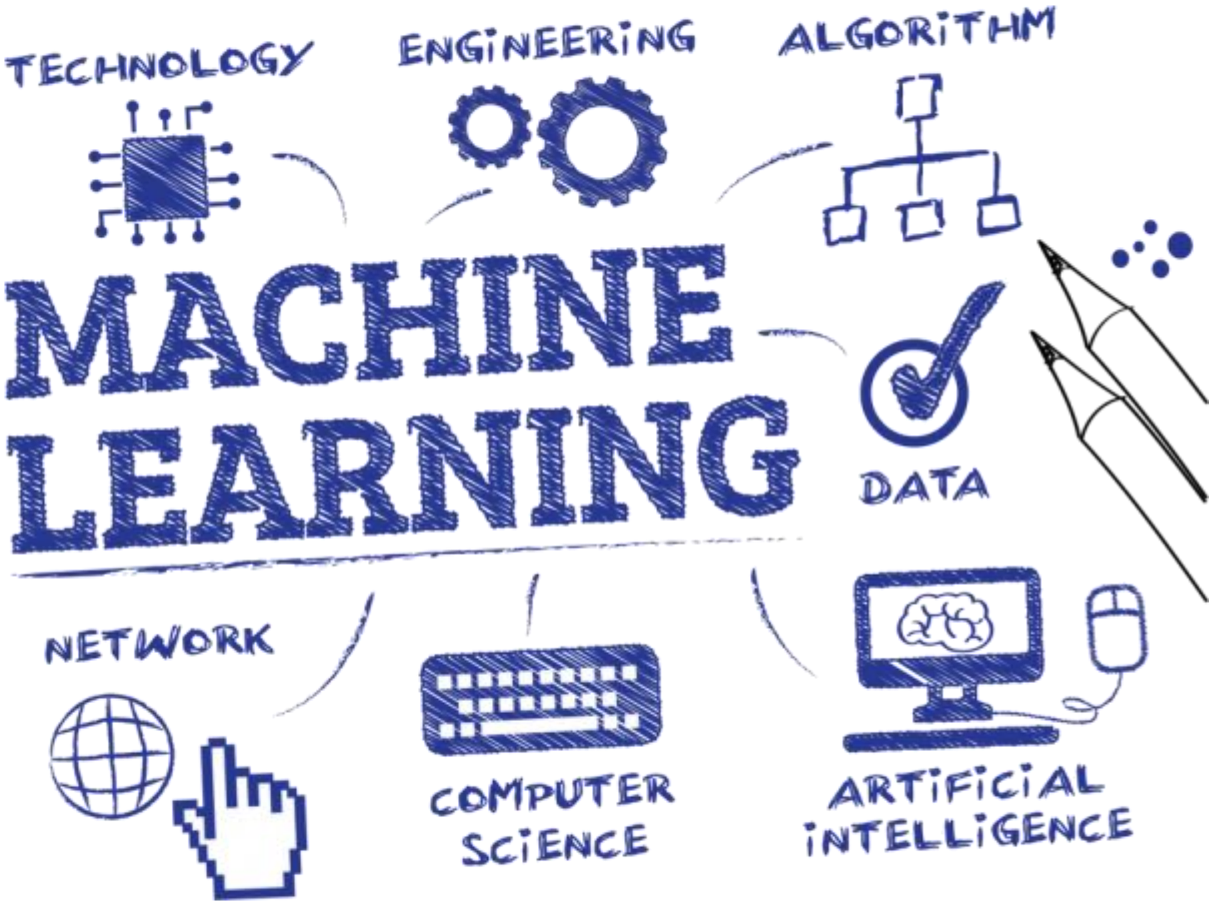
In reality, forward checking prunes every branch cut by backjumping. In a forward-checking search or a search that uses stronger consistency checking, basic backjumping is thus superfluous (such as MAC).

Key takeaway

CSPs are frequently solved via backtracking search, a type of depth-first search. Inference and search can be combined.

3.11 Case Study: Machine Learning at Google: The Amazing Use Case Of Becoming A Fully Sustainable Business

Google's purpose is to organise the world's information and make it accessible and helpful to everyone. They have also made major efforts from the beginning to do it in a way that does not deplete the world's natural resources.



Since 2007, the company has been carbon neutral, and 10 years later, they hope to have reached their next big goal: sourcing every watt of energy used for corporate operations from renewable sources.

During her visit to London to speak at the Economist Sustainability Summit 2018, Kate E Brandt, their lead for sustainability, spoke with me about some of the ways they have been tackling this ambitious challenge.

"We set a goal in 2012 to purchase 100 percent renewable energy for our operations – so it's a long-term commitment," she explained.

"We're still finalising our calculations, but all of our indicators point to our doing so in 2017 - so keep tuned!"

Of course, as the pioneers of machine learning and deep learning, Google has some serious technologies at their disposal to accomplish this. As you might think, it's been used to accomplish a number of goals across a wide range of use cases.

With data centres accounting for 2% of worldwide energy use, Google has made it a mission to improve efficiencies throughout its own network of 14 major hubs.

The problem is that the equipment is so complicated that there are literally billions of different server, chiller, cooling tower, heat exchanger, and control system configurations. Knowing which configurations will result in the best degree of Power Usage Effectiveness (PUE) — the metric used by Google to rate data centre energy efficiency – is impossible for humans to figure out. Even a team of Google data centre engineers with years of experience.

However, they went as far as they could, developing their own centres from the ground up to have complete control over the variables at play and customising components to be free of unnecessary, resource-sapping features found in off-the-shelf components.

To take things a step further, Google – particularly, one employee called Jim Gao – resorted to machine learning, the same technology that powers its image recognition and translation tools, which are used by millions of people around the world.

"So Jim took an online machine learning course and thought it was a pretty fascinating notion for optimising data centre cooling," Brandt adds.

"One thing he told me that makes it so effective as a tool is that if you think about ten devices, each with ten settings, that's ten billion possible combinations, which the human mind can't optimise."

"However, once he was able to train this algorithm to recognise trends across the many systems and how they impacted the cooling infrastructure, he saw a huge opportunity."

Machine learning is providing large volumes of data to complex algorithms meant to perform data processing tasks in a manner comparable to the human brain. As a result, computer systems have the ability to learn.

Jim and his team executed a pilot that resulted in a 40 percent reduction in the overall amount of energy consumed to cool the data centre.

"It really demonstrates how we can apply this technology to a complex system that is already highly optimised and get fantastic results." "It exemplifies what's so fascinating about machine learning's potential," Brandt adds.

"We see enormous chances to unlock new insights relating to sustainability with AI and machine learning, and I believe we are seeing some tremendous opportunities to improve the lives of people on the globe."

Unit - 4

Knowledge

4.1 Logical Agents, Knowledge-Based Agents

An intelligent agent needs knowledge of the real world in order to make effective decisions and reasoning.

- Knowledge-based agents are capable of retaining an internal state of knowledge, reasoning about it, updating it in response to observations, and acting on it. These agents can represent the world and operate intelligently using some type of formal representation.

Two major components make up knowledge-based agents:

- ○ Knowledge-base and
- ○ Inference system.

The following tasks must be performed by a knowledge-based agent:

- Agents should be able to represent states, actions, and other things.
- A representative new percept should be able to be incorporated.
- The internal representation of the world can be updated by an agent.
- The internal representation of the world can be deduced by an agent.
- An agent can deduce the best course of action.

The Architecture of knowledge-based agents

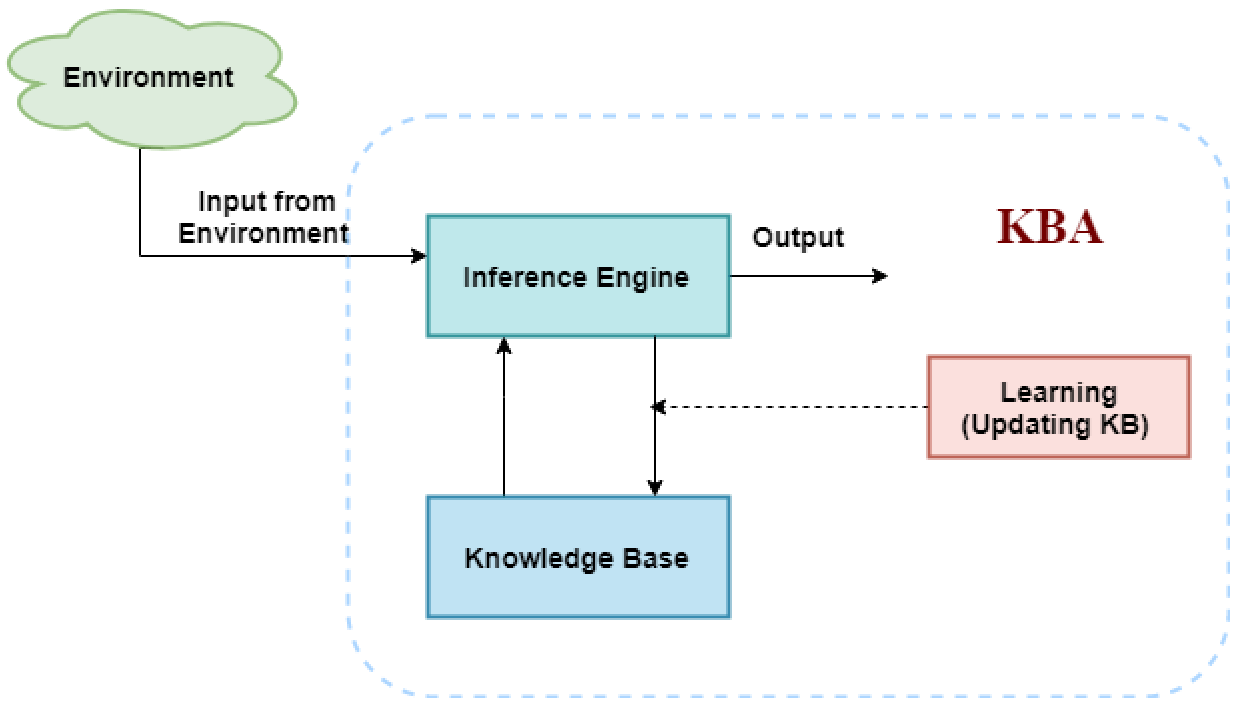


Fig 1: Architecture of knowledge-based agents

The graphic above shows a general design for a knowledge-based agent. The knowledge-based agent (KBA) collects input from the environment by monitoring it. The agent's inference engine processes the data and communicates with KB to make judgments based on the knowledge contained in KB. The learning component of KBA keeps the knowledge base current by learning new material.

Key takeaway

For efficient decision-making and reasoning, an intelligent agent needs knowledge about the real world.

These agents can use some type of formal representation to represent the world and act intelligently.

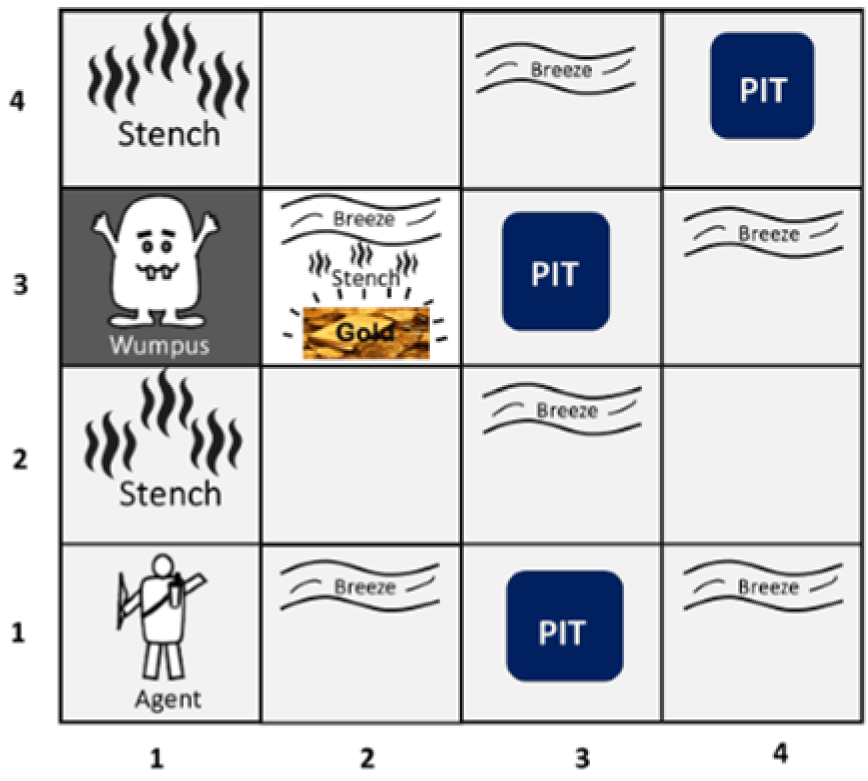
4.2 The Wumpus World

The Wumpus world is a basic world example that demonstrates the value of a knowledge-based agent and how knowledge representation is represented. It was inspired by Gregory Yob's 1973 video game Hunt the Wumpus.

The Wumpus world is a cave with four rooms and pathways connecting them. As a result, there are a total of 16 rooms that are interconnected. We now have a knowledge-based AI capable of progressing in this world. There is an area in the cave with a beast named Wumpus who eats everybody who enters. The agent can shoot the Wumpus, but he only has a single arrow. There are some Pits chambers in the Wumpus universe that are bottomless, and if an agent falls into one, he will be stuck there indefinitely.

The intriguing thing about this cave is that there is a chance of finding a gold heap in one of the rooms. So the agent's mission is to find the gold and get out of the cave without getting eaten by Wumpus or falling into Pits. If the agent returns with gold, he will be rewarded, but if he is devoured by Wumpus or falls into the pit, he will be penalised.

A sample diagram for portraying the Wumpus planet is shown below. It depicts some rooms with Pits, one room with Wumpus, and one agent in the world's (1, 1) square position.



There are some elements that can assist the agent in navigating the cave. The following are the components:

- The rooms adjacent to the Wumpus room are stinky, thus there is a stench there.

- The room next to PITs has a breeze, so if the agent gets close enough to PIT, he will feel it.

- If and only if the room contains gold, there will be glitter.

- If the agent is facing the Wumpus, the agent can kill it, and the Wumpus will cry horribly, which can be heard throughout the cave.

PEAS description of Wumpus world:

Performance Measures, Environment, Actuators, and Sensors (PEAS) are acronyms for Performance Measures, Environment, Actuators, and Sensors. The PEAS description aids in the classification of the agents.

The Wumpus World problem has the following PEAS description:

Performance measures:

- Agent gets the gold and return back safe = +1000 points
- Agent dies = -1000 points
- Each move of the agent = -1 point
- Agent uses the arrow = -10 points

Environment:

- There are 16(4x4) rooms in this cave.
- The rooms opposite to the Wumpus (not diagonally) stink.
- Rooms adjacent to the pit (but not diagonally) are cool.
- The gold-studded chamber gleams.
- Agent's first position is in Room[1, 1], facing the right side.
- Wumpus, gold, and the three pits can be found anywhere except in Room[1, 1].

Actuators:

Devices that enable the agent to carry out the operations listed below in the surroundings.

- Move forward
- Turn right
- Turn left
- Shoot

- Grab
- Release

Sensors:

Devices that assist the agent in detecting the following from their surroundings.

- Breeze
- Stench
- Glitter
- Scream (When the Wumpus is killed)
- Bump (when the agent hits a wall)

The Wumpus world Properties:

- **Partially observable:** Because the agent can only observe the immediate environment, such as a nearby room, the Wumpus universe is only partially visible.
- **Deterministic:** It is deterministic since the world's consequence and outcome are already known.
- **Sequential:** It is sequential because the order is critical.
- **Static:** Wumpus and Pits aren't moving, thus it's static.
- **Discrete:** The setting is distinct.

- **One agent:** We only have one agent, and Wumpus is not regarded as an agent, hence the environment is single agent.

Key takeaway

The Wumpus world is a cave with four rooms and pathways connecting them. As a result, there are a total of 16 rooms that are interconnected. We now have a knowledge-based AI capable of progressing in this world.

4.3 Logic

A logic is a formal language that allows sound inference and has well defined syntax and semantics. There are various logics that allow you to represent various types of things and allow for more or less efficient inference. Different varieties of logic exist, such as propositional logic, predicate logic, temporal logic, and description logic. However, expressing something in logic may not be natural, and conclusions may be ineffective.

Example of logic

Language of numerical constraints:

- A sentence:

$x + 3 \leq z$ x,

z - variable symbols (primitives in the language)

- An interpretation:

I: $x = 5, z = 2$

Variables mapped to specific real numbers

- Valuation (meaning) function V:

V $(x + 3 \leq z, I)$ is False for I: $x = 5, z = 2$

Is True for I: $x = 5, z = 10$

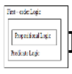


Fig 2: Logic

Types of logic

- Different types of logics possible
- Propositional logic
- First-order logic
- Temporal logic
- Numerical constraints logic
- Map-coloring logic

Key takeaway

A logic is a formal language, with precisely defined syntax and semantics, which supports sound inference.

The logic may be different types like propositional logic, predicate logic, temporal logic, description logic etc.

4.4 Propositional Logic: A Very Simple Logic

Propositional logic (PL) is the simplest form of logic where all the statements are made by propositions. A proposition is a declarative statement which is either true or false. It is a technique of knowledge representation in logical and mathematical form.

Example:

1. a) It is Sunday.
2. b) The Sun rises from West (False proposition)
3. c) 3+3= 7(False proposition)
4. d) 5 is a prime number.

Following are some basic facts about propositional logic:

● Propositional logic is also called Boolean logic as it works on 0 and 1.

● In propositional logic, we use symbolic variables to represent the logic, and we can use any symbol for a representing a proposition, such A, B, C, P, Q, R, etc.

● Propositions can be either true or false, but it cannot be both.

● Propositional logic consists of an object, relations or function, and logical connectives.

● These connectives are also called logical operators.

● The propositions and connectives are the basic elements of the propositional logic.

● Connectives can be said as a logical operator which connects two sentences.

● A proposition formula which is always true is called tautology, and it is also called a valid sentence.

● A proposition formula which is always false is called Contradiction.

● A proposition formula which has both true and false values is called

- Statements which are questions, commands, or opinions are not propositions such as "Where is Rohini", "How are you", "What is your name", are not propositions.

Syntax of propositional logic:

The syntax of propositional logic defines the allowable sentences for the knowledge representation. There are two types of Propositions:

1. Atomic Propositions
2. Compound propositions

1. Atomic Proposition: Atomic propositions are the simple propositions. It consists of a single proposition symbol. These are the sentences which must be either true or false.

Example:

1. a) 2+2 is 4, it is an atomic proposition as it is a true fact.
2. b) "The Sun is cold" is also a proposition as it is a false fact.

2. Compound proposition: Compound propositions are constructed by combining simpler or atomic propositions, using parenthesis and logical connectives.

Example:

1. a) "It is raining today, and street is wet."
2. b) "Ankit is a doctor, and his clinic is in Mumbai."

Logical Connectives:

Logical connectives are used to connect two simpler propositions or representing a sentence logically. We can create compound propositions with the help of logical connectives. There are mainly five connectives, which are given as follows:

1. Negation: A sentence such as ¬ P is called negation of P. A literal can be either Positive literal or negative literal.
2. Conjunction: A sentence which has ∧connective such as, P ∧ Q is called a conjunction.
   Example: Rohan is intelligent and hardworking. It can be written as, P= Rohan is intelligent, Q= Rohan is hardworking. → P∧ Q.
3. Disjunction: A sentence which has ∨ connective, such as P ∨ Q. Is called disjunction, where P and Q are the propositions.

Example: "Ritika is a doctor or Engineer",

Here P= Ritika is Doctor. Q= Ritika is Doctor, so we can write it as P ∨ Q.

4. Implication: A sentence such as P → Q, is called an implication. Implications are also known as if-then rules. It can be represented as

If it is raining, then the street is wet.

Let P= It is raining, and Q= Street is wet, so it is represented as P → Q

5. Biconditional: A sentence such as P⇔ Q is a Biconditional sentence, example If I am breathing, then I am alive

P= I am breathing, Q= I am alive, it can be represented as P ⇔ Q.

Following is the summarized table for Propositional Logic Connectives:

Connective symbols	Word	Technical term	Example
∧	AND	Conjunction	A ∧ B
∨	OR	Disjunction	A∨B
→	Implies	Implication	A→B
⇔	If and only if	Biconditional	A⇔B

$\neg$ Or $\sim$	Not	Negation	$\neg$ Aor $\sim B$

Truth Table:

In propositional logic, we need to know the truth values of propositions in all possible scenarios. We can combine all the possible combination with logical connectives, and the representation of these combinations in a tabular format is called Truth table. Following are the truth table for all logical connectives:

For negation

P	$\neg P$
True	False
False	True

For conjunction

P	Q	$P \wedge Q$
True	True	True
True	False	False
False	True	False
False	False	False

For disjunction

P	Q	$P \vee Q$

True	True	True
False	True	True
True	False	True
False	False	False

For implication

P	Q	$P \to Q$
True	True	True
True	False	False
False	True	True
False	False	True

For Biconditional

P	Q	P⇔Q
True	True	True
True	False	False
False	True	False
False	False	True

Truth table with three propositions:

We can build a proposition composing three propositions P, Q, and R. This truth table is made-up of 8n Tuples as we have taken three proposition symbols.

P	Q	R	$\neg R$	$P \vee Q$	$P \vee Q \to \neg R$

True	True	True	False	True	False
True	True	False	True	True	True
True	False	True	False	True	False
True	False	False	True	True	True
False	True	True	False	True	False
False	True	False	True	True	True
False	False	True	False	False	True
False	False	False	True	False	True

Precedence of connectives:

Just like arithmetic operators, there is a precedence order for propositional connectors or logical operators. This order should be followed while evaluating a propositional problem. Following is the list of the precedence order for operators:

Precedence	**Operators**
First Precedence	Parenthesis
Second Precedence	Negation
Third Precedence	Conjunction(AND)
Fourth Precedence	Disjunction(OR)
Fifth Precedence	Implication
Six Precedence	Biconditional

Note: For better understanding use parenthesis to make sure of the correct interpretations. Such as ¬R∨ Q, It can be interpreted as (¬R) ∨ Q.

Logical equivalence:

Logical equivalence is one of the features of propositional logic. Two propositions are said to be logically equivalent if and only if the columns in the truth table are identical to each other.

Let's take two propositions A and B, so for logical equivalence, we can write it as A⇔B. In below truth table we can see that column for ¬A∨ B and A→B, are identical hence A is Equivalent to B

A	B	$\neg A$	$\neg A \vee B$	$A \rightarrow B$
T	T	F	T	T
T	F	F	F	F
F	T	T	T	T
F	F	T	T	T

Properties of Operators:

Commutativity:

- P∧ Q= Q ∧ P, or
- P ∨ Q = Q ∨ P.

Associativity:

- (P ∧ Q) ∧ R= P ∧ (Q ∧ R),
- (P ∨ Q) ∨ R= P ∨ (Q ∨ R)

Identity element:

- P ∧ True = P,
- P ∨ True= True.

Distributive:

- P∧ (Q ∨ R) = (P ∧ Q) ∨ (P ∧ R).
- P ∨ (Q ∧ R) = (P ∨ Q) ∧ (P ∨ R).

DE Morgan's Law:

- ¬ (P ∧ Q) = (¬P) ∨ (¬Q)
- ¬ (P ∨ Q) = (¬ P) ∧ (¬Q).

Double-negation elimination:

- ¬ (¬P) = P.

Limitations of Propositional logic:

- We cannot represent relations like ALL, some, or none with propositional logic.

Example:

All the girls are intelligent.

Some apples are sweet.

- Propositional logic has limited expressive power.
- In propositional logic, we cannot describe statements in terms of their properties or logical relationships.

Key takeaway

Propositional logic (PL) is the simplest form of logic where all the statements are made by propositions. A proposition is a declarative statement which is either true or false. It is a technique of knowledge representation in logical and mathematical form.

4.5 Propositional Theorem Proving

Using logical principles of inference to produce logical implications is an alternative method to using logic to solve problems.

This can be less work than model-checking (creating a truth table) or even SAT solving in some circumstances.

Furthermore, logical inference rules are at the level of logical reasoning that humans deliberately strive for.

Entailment is a topic of interest in theorem-proving; for example, given a logical sentence A and a logical phrase B, we can ask if A entails B.

The main notion is that sentence A is what the agent already knows, and sentence B is what the agent can deduce from what it already knows.

The complete "brain" of an agent could be encoded in logic in sentence A.

The deduction theorem is a fundamental fact in logic that states:

If and only if A => B (A implies B) is a tautology, A entails B. (i.e. valid for all assignments of values to its variables).

Such that we can respond to the question "Does A imply B?" by demonstrating that the phrase A => B is a tautology

Remember that a phrase is unsatisfiable if it cannot be made true by assigning values to its variables.

So A => B is a tautology, then !(A=>B) is unsatisfiable, and !(A=>B) == !(!(A & !B)) == A & !B.

As a result, we can rewrite the deduction theorem as follows:

If and only if, A &!B is unsatisfactory, A entails B.

This means you can figure out entailment using a SAT solver!

Rules of Inference

Remember that proofs, or chains of accurate inferences, can be created using a variety of inference procedures.

Modus ponens, for example, is an inference rule that states:

A, A => B

---------   modus ponens

B

This rule states that if you're given a sentence A and a sentence A => B, you can infer B. For example, and-elimination is a set of rules that states:

A & B

-----   and-elimination

A

A & B

B

These two rules encode the (obvious!) fact that if the sentences A and B are both true, then A and B are both true.

Inference rules can also be used to express logical equivalences, such as:

A <==> B

(A=>B) & (B=>A)

There are many different inference rules, and selecting an acceptable set of rules for a theorem-prover is crucial.

- The application of a rule of inference can be thought of as an action taken on the state of the world, in which the rule of inference adds more facts to the knowledge base.

- If there are several inference rules to pick from, knowledge (i.e. heuristics) is required to assist in the decision-making process.

  - Alternatively, we could use backtracking, which entails picking a rule at random, applying it, and continuing until it "appears" that no proof is being achieved.

- Furthermore, how do we know that the inference rules we're employing are complete, i.e., how do we know that if a proof exists, our set of inference rules will locate it?

  - Is it enough to prove any entailment in propositional logic if the two and-elimination rules are our only rules of inference?

    - no!

    - for example, (P | P) -> P is clearly true, but and-elimination doesn’t apply to this sentence

4.6 Effective Propositional Model Checking

Given a fixed propositional vocabulary, the collection of viable models is small, hence entailment can be checked by enumerating models. Backtracking and local search methods are two efficient model-checking inference algorithms for propositional logic that can typically solve enormous problems fast.

Based on model checking, there are two families of methods addressing the SAT problem:

a. Based on backtracking

b. Based on local hill-climbing search

1. A complete backtracking algorithm

David-Putnam algorithm (DPLL):

```
Function DPLL-SATISFIABLE?(s) returns true or false

Inputs: s, a sentence in propositional logic

Clauses ← the set of clauses in the CNF representation of s

Symbols ← a list of  the proposition symbols in s

Return DPLL (clauses, symbols, {})

Function DPLL (clauses, symbols, model) returns true or false

If every clause in clauses is true in model then return true

If some clause in clauses in clauses is false in model then return false

P, value ← FIND-UNIT-CLAUSE (clauses, model)

If P is non-null then return DPLL (clauses, symbols – P, model ∪ {P=value} )
```

P ← FIRST(symbols): rest ←REST(symbols)

Return DPLL(clauses, rest, model ∪ {P=true}) or

DPLL(clauses, rest, model ∪ {P=true})

Above written is the DPLL algorithm for checking satisfiability of a sentence in propositional logic.

DPLL incorporates three enhancements over the TT-ENTAILS scheme: Early termination, pure symbol heuristic, and unit clause heuristic are all examples of heuristics.

Component analysis, variable and value ordering, intelligent backtracking, random restarts, and creative indexing are some of the tricks that SAT solvers use to scale up to huge problems.

Local search algorithms

Local search techniques can be easily applied to the SAT issue if the appropriate evaluation function is used. (An evaluation function that counts the amount of unsatisfied clauses can be used.)

These algorithms work by flipping the truth value of one symbol at a time in the space of full assignments.

Local minima are common in space, and escaping them necessitates various sorts of randomization.

WALKSAT and other local search methods can be utilised to identify solutions. These algorithms are sound, but they aren't complete.

WALKSAT: is one of the most basic and successful algorithms available.

Function WALKSAT (clauses, p, max, flips) returns a satisfying model or failure

Inputs: clauses, a set of clauses in propositional logic p, the probability of choosing to do a “random walk” move, typically around 0.5

Max, flips, number of flips allowed before giving up

Model ← a random assignment of true/false to the symbols in clauses

For i=1 **to** max flips **do**

If model satisfies clauses **then return** model

Clause ← a randomly selected clause from clauses that is false in model

With probability p flip the value in model of a randomly selected symbol from clause

Else flip whichever symbol in clause maximizes the number of satisfied clauses

Return failure

This is the WALKSAT algorithm for checking satisfiability by randomly flipping the value of variables

Every iteration, the algorithm selects an unsatisfied sentence and alternates between two methods for selecting a symbol to flip:

Either a. a "min-conflicts" step that reduces the number of unsatisfied clauses in the new state; or b. a "min-conflicts" step that reduces the number of unsatisfied clauses in the new state.

Alternatively, there is a "random walk" step that selects the symbol at random.

The input sentence is satisfied when the algorithm returns a model.

There are two possible reasons for the algorithm's failure:

Either a. The sentence is unsatisfiable;

Or b. We need to give the algorithm more time.

If we set max_flips=∞, p>0, the algorithm will:

Either a. If a model exists, eventually return it

Alternatively, b. If the statement is unsatisfactory, never end it.

As a result, WALKSAT is useful when we assume a solution but are unable to detect unsatisfiability.

The landscape of random SAT problems

When looking at satisfiability problems in CNF, an under constrained problem is one that has a small number of clauses confining the variables.

An issue that is over constrained has a large number of clauses compared to the number of variables and is likely to have no solutions.

The notation $CNF_k(m, n)$ denotes a k-CNF sentence with m clauses and n symbols. (with n variables and k literals per clause).

Given a set of random sentences, choose clauses consistently, independently, and without replacement from among those clauses with k distinct literals that are either positive or negative at random.

Hardness: problems right at the threshold > over constrained problems > under constrained problems

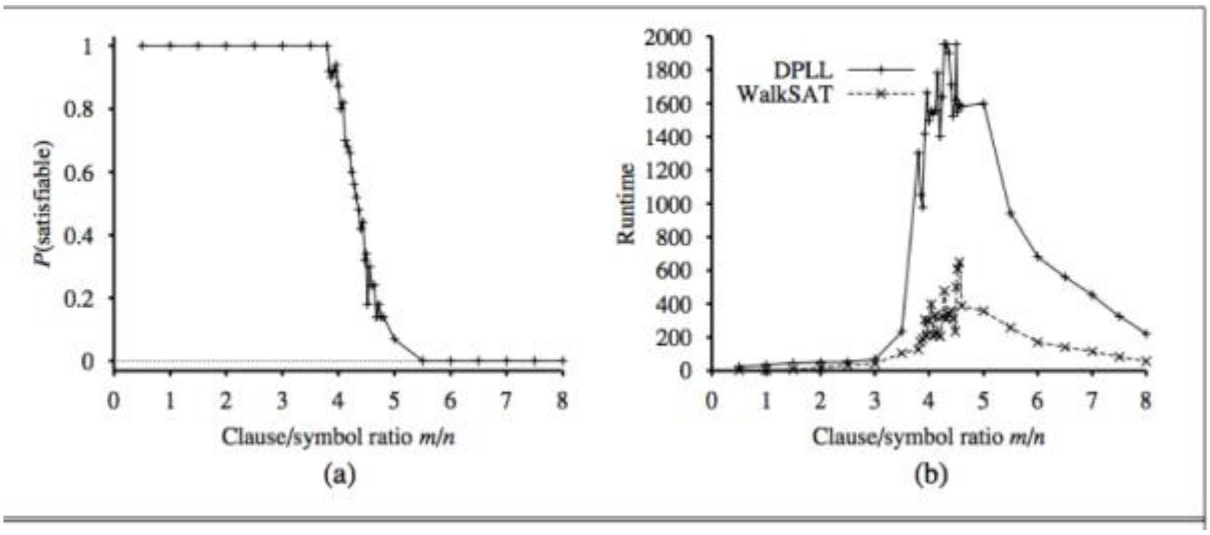


Fig 3: (a) graph showing the probability that random 3-CNF sentence with n = 50 symbols is satisfiable. (b) graph of the median run time on random 3-CNF sentences.

Satisfiability threshold conjecture: According to one hypothesis, there is a threshold ratio rk for any k3, such that when n approaches infinity, the likelihood that CNFk(n, rn) is satisfiable becomes 1 for all values or r below the threshold, and 0 for all values or r beyond the threshold.

4.7 Agents Based on Propositional Logic

1. The current state of the world

To avoid contradiction, we can link propositions to timestamps.

e.g. ¬ Stench, Stench

Fluent - Fluently refers to a changing aspect of the world. (For example, Ltx,y)

Atemporal variables - Symbols that represent eternal characteristics of the world do not require a time superscript.

Effect Axioms - define the result of an action at the following time step.

Frame problem - Because the effect axioms fail to define what remains unchanged as a result of an action, significant information is lost.

Solution: explicitly assert all the premises that remain the same by adding frame axioms.

Representation frame problem: In a universe with m distinct actions and n fluents, the proliferation of frame axioms is inefficient; the set of frame axioms will be O(mn).

Solution: Because the world is mobile, there is a solution (for humans each action typically changes no more than some number k of those fluents.) Instead of using a collection of axioms of size O(mk), define the transition model with a set of axioms of size O(mn).

Inferential frame problem: The difficulty of projecting the consequences of a t-step lan of action forward in time O(kt) rather than O(kt) (nt).

Solution: Change your focus from writing axioms about actions to writing axioms about fluents as a solution.

We'll have an axiom for each fluent F that describes the truth value of Ft+1 in terms of fluents at time t and possible actions at time t.

The truth value of F^{t+1} can be set in one of 2 ways:

Either a. At time t, the action causes F to be true at time t+1.

Or b. F was already true at time t, and the activity at time t did not change that.

A successor-state axiom is a type of axiom that has the following schema:

$F^{t+1} \Leftrightarrow$ Action causes $F^t \lor (F^t \land \neg \text{ Action Causes Not } F^t)$

Qualification problem: Specifying any exceptional exceptions that could result in the action failing.

2. A hybrid agent

Hybrid agent: mixes condition-action rules and problem-solving algorithms to determine various aspects of the state of the world.

As a current plan, the agent maintains and updates KB.

The atemporal axioms are found in the first KB. (Don't rely on t.)

Each time step adds a new percept sentence, as well as all the axioms that are dependent on t. (such as the successor-state axioms).

The agent then employs logical inference by ASKING the KB questions (to work out which squares are safe and which have yet to be visited).

The agent program's main body creates a plan based on a diminishing priority of goals:

1. If there is a glint, devise a strategy for grabbing the gold, returning to the original place, and climbing out of the cave;

2. If there is no current plan, plan a route (using A* search) to the nearest safe square that has not yet been visited, ensuring that the path passes through only safe squares;

3. If there are no safe squares to investigate and you still have an arrow, try shooting at one of the available wumpus spots to create a safe square.

4. If this doesn't work, look for a square to explore that isn't shown to be dangerous.

5. If the job is difficult because there is no such square, return to the original spot and climb out of the cave.

```
Function HYBRID-WUMPUS_AGENT(percept) returns an action

Inputs: percept, a list, [stench, breeze, glitter, bump, scream]

Persisten: KB, a knowledge base, initially the atemporal “wumpus physics” t,  a counter, initially 0, indicating time plan, an action sequence, initially empty

TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))

TELL the KB the temporal “physics” sentences for time t

Safe $\leftarrow\left\{[x,y]:\mathrm{ASK}\left(\mathrm{KB},\ \mathrm{OK}_{x,y}^{t}\right)=\mathrm{true}\right\}$

If ASK(KB, Glitter$^{t}$)=true **then**

$\mathrm{plan} \leftarrow [\mathrm{Grab}] + \mathrm{PLAN}-\mathrm{ROUTE}(\mathrm{current},\{[1,1]\},\mathrm{safe}) + [\mathrm{Climb}]$

**If** plan is empty **then**

$\mathrm{unvisited}\leftarrow\left\{[x,y]:\mathrm{ASK}\left(\mathrm{KB},\mathrm{L}_{x,y}^{t}\right)=\mathrm{false\ for\ all\ t'}\leq t\right\}$
$\mathrm{plan}\leftarrow\mathrm{PLAN}-\mathrm{ROUTE}(\mathrm{current},\ \mathrm{unvisited}\ \cap \mathrm{safe},\mathrm{safe})$
```

```
If plan is empty and ASK(KB, $HaveArrow^t$) = true then
Possible_wumpus $\leftarrow \{[x,y] : ASK(KB, \neg W_{x,y}) = false\}$
Plan $\leftarrow$ PLAN – ROUTE(current, possible_wumpus, safe)
If plan is empty then  //no choice but to take a risk
not_unsafe $\leftarrow \{[x,y] : ASK(KB, \neg OK^t_{x,y}) = false\}$
Plan $\leftarrow$ PLAN-ROUTE(current, unvisited $\cap$ not-unsafe, safe)
If plan is empty then
Plan $\leftarrow$ PLAN-ROUTE(current, {[1,1]}, safe) +[Climb]
Action $\leftarrow$ POP(plan)
TELL(KB, MAKE-ACTION-SENTENCE(action, t))
$t \leftarrow t + 1$
Return action
Function PLAN_ROUTE(current, goals, allowed) returns  an action sequence
Inputs: currents, the agent's current position
Goals, a set of squares; try to plan a route to one of them
Allowed,  a set of squares that can form part of the route
Problem $\leftarrow$ ROUTE-PROBLEM (current, goals, allowed) (problem)
Return $A^*$ - GRAPH-SEARCH (problem)
```

This is the Hybrid agent program for the wumpus world.

Weakness: The computational expense goes up as time goes by.

3. Logical state estimation

To maintain a consistent update time, we must cache the inference result.

Belief state: Some representation of the set of all current world states that could exist. (used to replace the previous history of percepts and all of its repercussions.)

$WumpusAlive^1 \wedge L_{2,1}^1 \wedge B_{2,1} \wedge (P_{3,1} \vee P_{2,2})$

The proposition symbols linked with the current time step and the temporal symbols are used in a logical phrase.

Maintaining a logical statement that represents the set of probable states consistent with the observation history is required for logical state estimation. Each update step necessitates inference using the environment's transition model, which is comprised of successor-state axioms that define how each fluent changes.

State estimation: The process of updating one's belief state as new information becomes available.

Exact state estimation may necessitate logical formulations with exponentially increasing symbol counts.

One typical method for approximating state estimation is to characterise belief states as literal conjunctions (1-CNF formulas).

Given the belief state at t-1, the agent simply tries to prove Xt and Xt for each symbol Xt.

The new belief state is formed by the conjunction of verifiable literals, and the previous belief state is discarded.

(As time passes, this scheme may lose some information.)

Given the whole percept history, the set of feasible states represented by the 1-CNF belief state comprises all states that are in fact possible. The 1-CNF belief state serves as a conservative approximation or outer envelope.

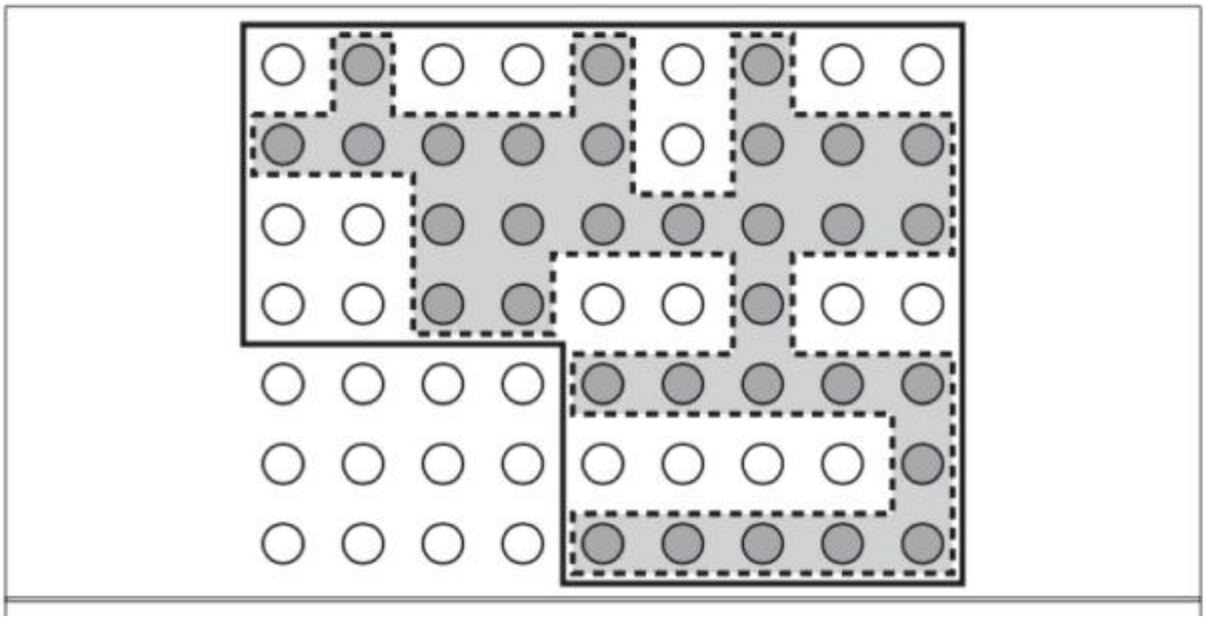


Fig 4: Depiction of 1-CNF belief state as a simply representable, conservative approximation to the exact belief state.

4. Making plans by propositional inference

Instead of using A* search, we can develop plans using logical inference.

Basic idea:

1. Make a sentence that incorporates the following elements:

a) Init0: a set of statements about the initial state; b) Init1: a set of assertions about the initial state; c) Init2:

b) Transition1,..., Transitiont: The successor-state axioms for all feasible actions at each time interval up to a maximum time interval t;

c) Have Goldt Climbed Outt: The claim that the goal has been reached at time t.

2. Use a SAT solver to complete the sentence. The aim can be achieved if the solver discovers a satisfying model; else, planning is impossible.

3. Assuming you've established a model, extract the variables that describe actions and set them to true.

They constitute a strategy for achieving the objectives when taken together.

SAT solution, or identifying feasible models detailing future action sequences that meet the objective, can be used to make decisions within a logical agent. This method only works in totally visible or sensor-free environments.

SATPLAN: is an acronym for "propositional planning." (Cannot be used in an environment that is just partially visible.)

SATPLAN looks for models in a sentence that includes the beginning state, aim, successor-state axioms, and action exclusion axioms.

(Because the agent has no idea how many steps it will take to attain the goal, the algorithm tries every number of steps t up to some maximum plausible plan length Tmax.)

```
Function SATPLAN(init, transition, goal, $T_{max}$) returns solution or failure

Inputs: init, transition, goal, constitute a description of the problem

$T_{max}$, an upper limit for plan length

For t=0 to $T_{max}$ do

Cnf $\leftarrow$ TRANSLATE-TO-SAT(init, transition, goat, t)

Model $\leftarrow$ SAT-SOLVER(cnf)

If model is null then

Return EXTRACT-SOLUTION(model)

Return failure
```

The SATPlan algorithm.

Precondition axioms: added to avoid creating plans containing illegal actions, saying that an action occurrence requires the preconditions to be satisfied.

Action exclusion axioms were included to prevent the formation of plans that had many simultaneous activities that interfered with one another.

Because it lacks the expressive power to deal concisely with time, space, and universal patterns of relationships among things, propositional logic does not scale to unbounded settings.

4.8 First-Order Logic

In the topic of Propositional logic, we have seen how to represent statements using propositional logic. But unfortunately, in propositional logic, we can only represent the facts, which are either true or false. PL is not sufficient to represent complex sentences or natural language statements. The propositional logic has very limited expressive power. Consider the following sentence, which we cannot represent using PL logic.

- "Some humans are intelligent", or
- "Sachin likes cricket."

To represent the above statements, PL logic is not sufficient, so we required some more powerful logic, such as first-order logic.

First-Order logic:

- First-order logic is another way of knowledge representation in artificial intelligence. It is an extension to propositional logic.

- FOL is sufficiently expressive to represent the natural language statements in a concise way.

- First-order logic is also known as Predicate logic or First-order predicate logic. First-order logic is a powerful language that develops information about the objects in a more easy way and can also express the relationship between those objects.

- First-order logic (like natural language) does not only assume that the world contains facts like propositional logic but also assumes the following things in the world:

    - Objects: A, B, people, numbers, colors, wars, theories, squares, pits, wumpus, ...
    - Relations: It can be unary relation such as: red, round, is adjacent, or n-any relation such as: the sister of, brother of, has color, comes between
    - Function: Father of, best friend, third inning of, end of, ...

- As a natural language, first-order logic also has two main parts:

    - Syntax
    - Semantics

Syntax of First-Order logic:

The syntax of FOL determines which collection of symbols is a logical expression in first-order logic. The basic syntactic elements of first-order logic are symbols. We write statements in short-hand notation in FOL.

Basic Elements of First-order logic:

Following are the basic elements of FOL syntax:

Constant	1, 2, A, John, Mumbai, cat,....
Variables	x, y, z, a, b,....

Predicates	Brother, Father, >,....
Function	Sqrt, LeftLegOf, ...
Connectives	∧, ∨, ¬, ⇒, ⇔
Equality	==
Quantifier	∀, ∃

Atomic sentences:

- Atomic sentences are the most basic sentences of first-order logic. These sentences are formed from a predicate symbol followed by a parenthesis with a sequence of terms.
- We can represent atomic sentences as Predicate (term1, term2, ..., term n).

Example: Ravi and Ajay are brothers: => Brothers(Ravi, Ajay).

Chinky is a cat: => cat (Chinky).

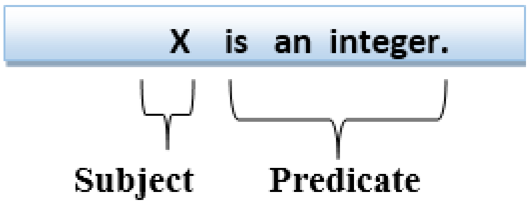
Complex Sentences:

- Complex sentences are made by combining atomic sentences using connectives.

First-order logic statements can be divided into two parts:

- Subject: Subject is the main part of the statement.
- Predicate: A predicate can be defined as a relation, which binds two atoms together in a statement.

Consider the statement: "x is an integer.", it consists of two parts, the first part x is the subject of the statement and second part "is an integer," is known as a predicate.



Quantifiers in First-order logic:

- A quantifier is a language element which generates quantification, and quantification specifies the quantity of specimen in the universe of discourse.

- These are the symbols that permit to determine or identify the range and scope of the variable in the logical expression. There are two types of quantifier:

1. Universal Quantifier, (for all, everyone, everything)
2. Existential quantifier, (for some, at least one).

Universal Quantifier:

Universal quantifier is a symbol of logical representation, which specifies that the statement within its range is true for everything or every instance of a particular thing.

The Universal quantifier is represented by a symbol ∀, which resembles an inverted A.

Note: In universal quantifier we use implication "→".

If x is a variable, then ∀x is read as:

- For all x

- For each x

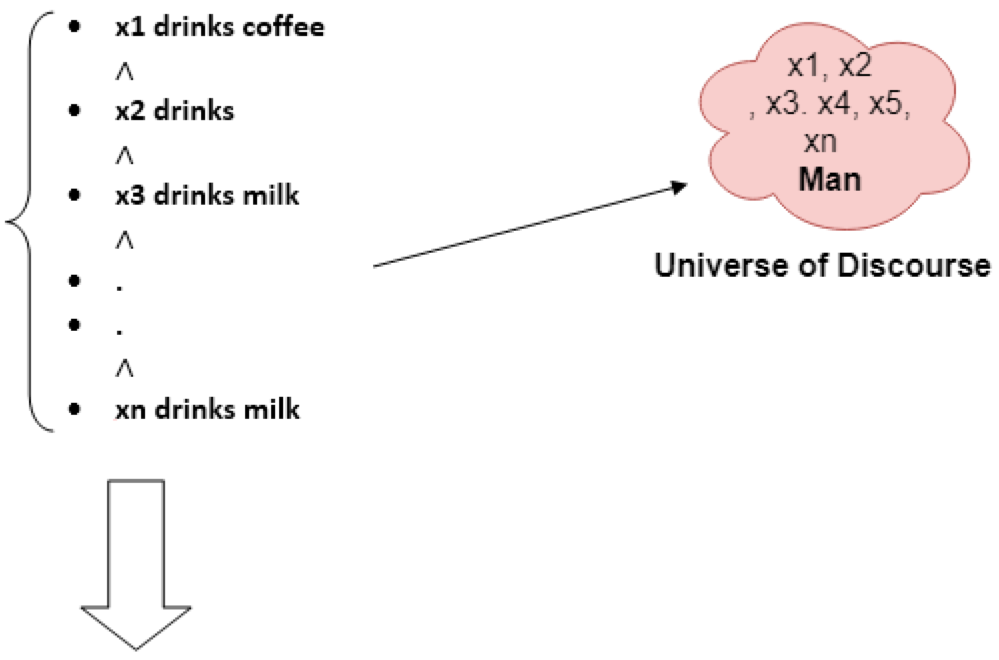
- For every x.

Example:

All man drink coffee.

Let a variable x which refers to a cat so all x can be represented in UOD as below:

∀x



So in shorthand notation, we can write it as :

man(x) → drink (x, coffee).

It will be read as: There are all x where x is a man who drink coffee.

Existential Quantifier:

Existential quantifiers are the type of quantifiers, which express that the statement within its scope is true for at least one instance of something.

It is denoted by the logical operator ∃, which resembles as inverted E. When it is used with a predicate variable then it is called as an existential quantifier.

Note: In Existential quantifier we always use AND or Conjunction symbol (∧).

If x is a variable, then existential quantifier will be ∃x or ∃(x). And it will be read as:

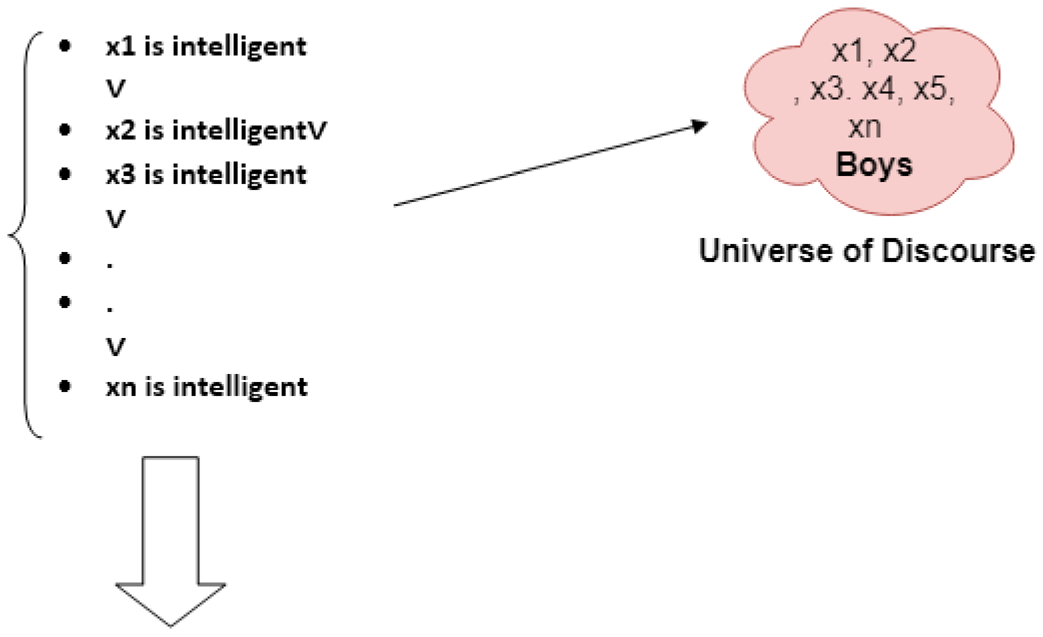
- There exists a 'x.'

- For some 'x.'
- For at least one 'x.'

Example:

Some boys are intelligent.

∃x:



- x1 is intelligent
  V
- x2 is intelligentV
- x3 is intelligent
  V
- .
- .
  V
- xn is intelligent

So in short-hand notation, we can write it as:

boys(x) ∧ intelligent(x)

It will be read as: There are some x where x is a boy who is intelligent.

Points to remember:

- The main connective for universal quantifier ∀ is implication →.
- The main connective for existential quantifier ∃ is and ∧.

Properties of Quantifiers:

- In universal quantifier, ∀x∀y is similar to ∀y∀x.

- In Existential quantifier, ∃x∃y is similar to ∃y∃x.
- ∃x∀y is not similar to ∀y∃x.

Some Examples of FOL using quantifier:

1. All birds fly.

In this question the predicate is "fly(bird)."

And since there are all birds who fly so it will be represented as follows.

∀x bird(x) →fly(x).

2. Every man respects his parent.

In this question, the predicate is "respect(x, y)," where x=man, and y= parent.

Since there is every man so will use ∀, and it will be represented as follows:

∀x man(x) → respects (x, parent).

3. Some boys play cricket.

In this question, the predicate is "play(x, y)," where x= boys, and y= game. Since

There are some boys so we will use ∃, and it will be represented as:

∃x boys(x) → play(x, cricket).

4. Not all students like both Mathematics and Science.

In this question, the predicate is "like(x, y)," where x= student, and y= subject.

Since there are not all students, so we will use ∀ with negation, so following representation for this:

¬∀ (x) [student(x) → like(x, Mathematics) ∧ like(x, Science)].

5. Only one student failed in Mathematics.

In this question, the predicate is "failed(x, y)," where x= student, and y= subject. Since there is only one student who failed in Mathematics, so we will use following representation for this:

∃(x) [student(x) → failed (x, Mathematics) ∧∀ (y) [¬(x==y) ∧ student(y) → ¬failed (x, Mathematics)].

Free and Bound Variables:

The quantifiers interact with variables which appear in a suitable way. There are two types of variables in First-order logic which are given below:

Free Variable: A variable is said to be a free variable in a formula if it occurs outside the scope of the quantifier.

Example: ∀x ∃(y)[P (x, y, z)], where z is a free variable.

Bound Variable: A variable is said to be a bound variable in a formula if it occurs within the scope of the quantifier.

Example: ∀x [A (x) B(y)], here x and y are the bound variables.

4.9 Representation Revisited

Programming languages (such as C++, Java, and Lisp) are by far the most widely used formal languages.

This procedural method contrasts with propositional logic's declarative character, in which knowledge and inference are separated and inference is domain-independent.

They lack the expressiveness required to deal with incomplete data.

● Using disjunction and negation, propositional logic has enough expressive capability to deal with partial knowledge.

● Compositionality is a third characteristic of propositional logic that is useful in representation languages.

The meaning of a sentence in a compositional language is determined by the meaning of its constituent elements.

For example, ―S1,4 ^ S1,2‖ is related to the meanings of ―S1,4‖ and ―S1,2‖.

Propositional logic lacks the expressive power to succinctly explain a complex environment with numerous items.

For example, we were compelled to establish a distinct regulation for each square regarding breezes and pits, such as

$B_{1,1} \Leftrightarrow (P_{1,2} \text{ or } P_{2,1})$

In English, though, it seems simple enough to state unequivocally that squares near to pits are airy.

Natural languages are also non-compositional, in the sense that the meaning of a statement like "Then she saw it" might be influenced by the context created by numerous preceding and following sentences.

Finally, natural languages contain ambiguity, which can make thinking harder.

The following are the most obvious features of natural language syntax:

● Object-referential nouns and noun phrases (squares , pits, wumpuses).

● Verbs and verb phrases that allude to object relationships (is breezy, is adjacent to, shoots).

- Some of these relationships are functions, meaning they have only one value for each input. It's simple to start a list of objects, relations, and functions:

Objects: People, houses, numbers, theories, Ronald McDonald, colors, baseball games, wars, centuries….

Relation: These can be unary relations or properties like red, round, bogus, prime, multi-storied..........., or more general n-ary relations like brother of, bigger than, within, part of, has colour, happened after, owns, comes between, etc.

Functions: father of, closest friend, third inning of, more than, and the start of.

- The first-order logic language has a syntax and semantics based on objects and relations.
- First-order logic can also be used to convey facts about some or all of the universe's items.
- This allows generic laws or norms to be represented, such as the sentence "Squares adjacent to the Wumpus are stinky."

The ontological commitment made by each language—that is, what it assumes about the nature of reality—is the basic difference between propositional and first-order logic.

- Propositional logic, for example, assumes that there are facts in the world that either hold or do not hold. There are two possible states for each fact: true or untrue.
- Temporal logic assumes that facts are true at specific moments and that those times (which can be points or intervals) are in chronological sequence.

Higher-order logic considers the first-order logic's relations and functions to be objects in and of themselves. This enables claims to be made about all object relations. A talent for dealing with logical notation is required of an AI student.

Language	**Ontological Commitment (What exists in the world)**	**Epistemological Commitment (What an agent believes about facts)**
Propositional logic	Facts	True/false/unknown
First-order logic	Facts, objects, relations	True/false/unknown
Temporal logic	Facts, objects, relations, times	True/false/unknown
Probability theory	Facts	Degree of belief $\in [0,1]$
Fuzzy logic	Facts with degree of truth $\in [0,1]$	Known interval value

Table: Formal language and their ontological and epistemological commitments

4.10 Syntax and Semantics of First-Order Logic, Using First-Order Logic

- First-order logic is a type of knowledge representation used in artificial intelligence. It's a propositional logic variation.
- FOL is expressive enough to convey natural language statements in a concise manner.
- First-order logic is sometimes known as predicate logic or first-order predicate logic. First-order logic is a complex language for constructing information about objects and expressing relationships between them.
- First-order logic (as does natural language) assumes not just that the world contains facts, as propositional logic does, but also that the world has the following:

  - Objects: People, numbers, colors, conflicts, theories, squares, pits, wumpus,

- Relation: It can be a unary relation like red, round, or nearby, or an n-any relation like sister of, brother of, has color, or comes between.
- Functions: Father of, best friend, third inning of, last inning of......

- First-order logic contains two basic pieces as a natural language:

  - Syntax
  - Semantics

Syntax

Syntax has to do with what ‘things’ (symbols, notations) one is allowed to use in the language and in what way; there is/are a(n):

- Alphabet
- Language constructs
- Sentences to assert knowledge

Logical connectives (⇒, ∧, ∨, and ⇐⇒), negation (¬), and parentheses. These will be used to recursively build complex formulas, just as was done for propositional logic.

Constants symbols are strings that will be interpreted as representing objects, e.g., Bob might be a constant.

Variable symbols will be used as “place holders” for quantifying over objects.

Predicate symbols Each has an arity (number of arguments) associated with it, which can be zero or some other finite value. Predicates will be used to represent object characteristics and relationships.

Zero-arity Because predicate symbols are viewed as propositions in first-order logic, propositional logic is subsumed. These assertions can be thought of as world properties.

Predicates with a single parameter can be thought of as specifying object attributes. If Rich is a single-arity predicate, then Rich (Bob) is used to indicate that Bob is wealthy. Multi-arity predicates are used to describe relationships between items.

Formula $\rightarrow$ PrimitiveFormula

| (Formula Connective Formula)

| $\neg$ Sentence

| Qualifier Variable Formula

PrimitiveFormula $\rightarrow$ Predicate(Term,…,Term)

Term $\rightarrow$ Function(Term,…,Term)

| Constant

| Variable

Connectifier $\Rightarrow | \wedge | | \vee |$

Quantifier $\rightarrow$

Constant $\rightarrow$ any string that is not used as a variable predicate or function

Variable $\rightarrow$ any string that is not used as a constant predicate or function

Predicate $\rightarrow$ any string that is not used as a constant variable or function

Function $\rightarrow$ any string that is not used as a constant predicate or constant

Function symbols Each has a specific arity (number of input arguments) and is understood as a function that maps the stated number of input objects to objects. Allowing FatherOf to be a single-arity function symbol, the natural interpretation of FatherOf (Bob) is Bob's father.

Zero-arity function symbols are considered to be constants.

Universal and existential quantifier symbols will be used to quantify over objects. For example, ∀ x Alive(x) ⇒ Breathing(x) is a universally quantified statement that uses the variable x as a placeholder.

Semantics of First-Order Logic

As with all logics, the first step in determining the semantics of first-order logic is to define the models of first-order logic. Remember that one of the benefits of using first-order logic is that it allows us to freely discuss objects and their relationships. As a result, our models will comprise objects as well as information about their properties and interactions with other items.

First order model

A first-order model is a tuple hD, Ii, where D denotes a non-empty object domain and I denote an interpretation function. D is nothing more than a collection of items or elements that can be finite, infinite, or uncountable. The interpretation function I gives each of the available constant, function, and predicate symbols a meaning or interpretation as follows:

- If c is a constant symbol, then I(c) is an object in D. Thus, given a model, a constant can be viewed as naming an object in the domain.
- If f is a function symbol of arity n, then I(f) is a total function from Dn to D. That is the interpretation of f is a function that maps n domain objects to the domain D.
- If p is a predicate symbol of arity n > 0, then I(p) is a subset of Dn; that is, a predicate symbol is interpreted as a set of tuples from the domain. If a tuple O = (o_1, · · · , o_n) is in I(p) then we say that p is true for the object tuple O.
- If p is a predicate symbol of arity 0, i.e., a simple proposition, then I(p) is equal to either true or false.

Assume we have a single predicate TallerThan, a single function FatherOf, and a single constant Bob. The following could be a model M1 for these symbols:

D = {BOB, JON, NULL}

I(Bob) = BOB

I(TallerThan) = {hBOB, JONi}

Because I(FatherOf) is a function, we'll only show the value for each argument to give the FatherOf meaning.

I(FatherOf)(BOB) = JON

I(FatherOf)(JON) = NULL

I(FatherOf)(NULL) = NULL

M2 could also be interpreted as follows,

D = { BOB, JON }

I(Bob) = BOB

I(TallerThan) = { hBOB, JONi,hJON, BOBi }

I(FatherOf)(BOB) = BOB

I(FatherOf)(JON) = JON

It's vital to highlight the difference between Bob, which is a constant (a syntactic entity), and BOB, which is a domain object (a semantic entity). The second interpretation isn't exactly what we're looking for (the objects are dads of themselves, and TallerThan is inconsistent), but it's still a viable model. By imposing proper limitations on the symbols, the knowledge base can rule out such unexpected models from consideration.

Key takeaway

The syntax of FOL determines which collection of symbols is a logical expression in first-order logic.

The basic syntactic elements of first-order logic are symbols. We write statements in short-hand notation in FOL.

4.11 Knowledge Engineering in First-Order Logic

Inference is used in First-Order Logic to generate new facts or sentences from current ones. It's crucial to understand some basic FOL terms before diving into the FOL inference rule.

Substitution:

Substitution is a fundamental approach for modifying phrases and formulations. All first-order logic inference systems contain it. The substitution becomes more difficult when there are quantifiers in FOL. We refer to the replacement of a constant "a" for the variable "x" when we write F[a/x].

Equality:

Atomic sentences are generated in First-Order Logic not only through the employment of predicate and words, but also through the application of equality. We can do this by using equality symbols, which indicate that the two terms relate to the same thing.

FOL inference rules for quantifier:

Because inference rules in first-order logic are comparable to those in propositional logic, below are some basic inference rules in FOL:

- Universal Generalization

- Universal Instantiation
- Existential Instantiation
- Existential introduction

Universal Generalization:

- Universal generalization is a valid inference rule that states that if premise P(c) is true for any arbitrary element c in the universe of discourse, we can arrive at the conclusion x P. (x).
- It can be represented as: $\frac{\forall x p(x)}{p(c)}$
- If we want to prove that every element has a similar property, we can apply this rule.
- x must not be used as a free variable in this rule.

Example: Let's represent, P(c): "A byte contains 8 bits", so for ∀ x P(x) "All bytes contain 8 bits.", it will also be true.

Universal Instantiation:

- Universal instantiation, often known as universal elimination or UI, is a valid inference rule. It can be used numerous times to add more sentences.
- The new knowledge base is logically equivalent to the previous one.
- We can infer any phrase by replacing a ground word for the variable, according to the UI.
- According to the UI rule, any phrase P(c) can be inferred by substituting a ground term c (a constant inside domain x) for any object in the universe of discourse in x P(x).
- It can be represented as: $\frac{\forall \mathrm{xp(x)}}{\mathrm{p(c)}}$

Example: IF "Every person like ice-cream"=> ∀x P(x) so we can infer that

"John likes ice-cream" => P(c)

Existential Instantiation:

- Existential Elimination, also known as Existential Instantiation, is a valid first-order logic inference rule.
- It can only be used once to substitute for the existential sentence.
- Despite the fact that the new KB is not conceptually identical to the previous KB, it will suffice if the old KB was.
- This rule states that one can infer P(c) from the formula given in the form of ∃x P(x) for a new constant symbol c.
- The only constraint with this rule is that c must be a new word for which P(c) is true.
- It can be represented as: $\frac{\exists x p(c)}{p(x)}$

Existential introduction

- An existential generalization, also known as an existential introduction, is a valid inference rule in first-order logic.
- If some element c in the world of discourse has the characteristic P, we can infer that something else in the universe has the attribute P, according to this rule.
- It can be represented as: $\frac{p(c)}{\exists x p(x)}$

Example: Let's say that,

"Pritisha got good marks in English."

"Therefore, someone got good marks in English."

Propositional vs. First-Order Inference

Previously, inference in first order logic was checked via propositionalization, which is the act of turning the Knowledge Base included in first order logic into propositional logic and then utilizing any of the propositional logic inference mechanisms to check inference.

Inference rules for quantifiers:

To get sentences without quantifiers, several inference techniques can be applied to sentences containing quantifiers. We'll be able to make the conversion if we follow these requirements.

Universal Instantiation (UI):

The rule states that by substituting a ground word (a term without variables) for the variable, we can deduce any sentence. Let SUBST () stand for the outcome of the substitution on the sentence a. The rule is then written.

$$\frac{\forall v_{\alpha}}{\text{SUBST}\left(\left\{\frac{v}{g}\right\}, \alpha\right)}$$

For any v and g ground term combinations. For example, in the knowledge base, there is a statement that states that all greedy rulers are Evil.

∀x King(x)A Greedy(x) => Evil(x)

For the variable x, with the substitutions like {x/John}, {x/Richard} the following sentences can be inferred

King(John)A Greedy(John) => Evil(John)
King(Richard)A Greedy(Richard) => Evil(Richard)

As a result, the set of all potential instantiations can be used to substitute a globally quantified phrase.

Existential Instantiation (EI):

The existential statement states that there is some object that fulfills a requirement, and that the instantiation process is simply giving that object a name that does not already belong to another object. A Skolem constant is the moniker given to this new name. Existential Instantiation is a subset of a broader process known as "skolemization."

If the statement a, variable v, and constant symbol k do not occur anywhere else in the knowledge base,

$$\frac{\exists v_{\alpha}}{\mathrm{SUBST}(\{v/k\},\alpha)}$$

For example, from the sentence

$\exists x\ Crown(x) \wedge onHead(x, John)$

So we can infer the sentence

$Crown(C_1) \wedge OnHead(C_1, John)$

As long as C1 does not appear elsewhere in the knowledge base. Thus, an existentially quantified sentence can be replaced by one instantiation

Elimination of Universal and Existential quantifiers should give new knowledge base which can be shown to be inferentially equivalent to old in the sense that it is satisfiable exactly when the original knowledge base is satisfiable.

Key takeaway

In First-Order Logic, inference is used to derive new facts or sentences from existing ones.

In First-Order Logic, atomic sentences are formed not only via the use of predicate and words, but also via the application of equality.

There are some Inference rules that can be applied to sentences with quantifiers to obtain sentences without quantifiers.

4.12 Case Study: BBC To Launch AI - Enabled Interactive Radio Show for Amazon Echo And Google Home Chat bots

Our obsession with Alexa, Siri, and other voice-activated gadgets has inspired inventors across a wide range of industries to devise new ways to bring similar technology to their customers or design consumer experiences that take advantage of this potential. People appear to want spoken interfaces, and the BBC's Research & Development team has gone all out to provide the possibility for a two-way spoken discussion with their listeners.

Talking with Machines is the name of the BBC's initiative. "Explore the possibilities of these devices and platforms in terms of content, interface design, and software development processes," says the initiative. The BBC wants to be able to accommodate current technology while also laying the groundwork for future technology.

Their objectives are to create a device-independent platform that will support spoken interfaces and allow them to work with Siri, Alexa, and anyone else who joins them in the future, as well as to establish internal expertise within the BBC in the area of spoken interface technology.

Furthermore, the BBC research and development team is pondering and testing all of the interaction and content platforms that two-way communication on speaking devices might enable.

To build their own engine for speech to text and natural language processing, the BBC partnered with other internal teams working on similar topics.

Interactive Radio

The Talking with Machines programme stands out because it focuses on spoken interfaces, whereas other projects in the organisation are text-based. The team is taking notes and using what they've learned about generic conversational UI from others.

The construction of an original interactive audio drama pilot that employs the BBC's "storey engine" and was built expressly for voice devices is one of the BBC's first publicly displayed tests from this initiative. This engine simplifies the distribution of the same narrative across numerous platforms.

The BBC will release The Inspection Chamber, a comedy/science fiction narrative, later this year in conjunction with Rosina Sound. What distinguishes it as interactive? Listeners become a part of the storey as it streams through your Amazon Echo or Google Home and prompts you to add your own lines. The development team fully anticipates expanding into other voice-activated devices in the future.

According to the BBC's Research & Development blog, "with this pilot, you're actively playing a part in the storey, using your own voice—we wanted to make it feel like you're having a genuine, direct connection with the other characters in the piece."

Although there are parallels between this and a choose-your-own-adventure storey, this interactive storey goes far beyond. The Inspection Chamber evolved into its own one-of-a-kind experience, inspired by computer games such as The Stanley Parable and Papa Sangre, as well as authors such as Frank Kafka and Douglas Adams.

"Hello, my name is Dave," says a female voice at the start of the storey. I hope you've had a pleasant stay in the containment room." As the storey progresses, the listener is subjected to a scientific assessment and is asked questions. Although the overall narrative doesn’t change, the outcome of the 20-minute story will be different based on the listener’s answers to questions they are asked throughout the tale.

Listen to a sample now

The development team had to strike a balance between the storyline and technical constraints, such as Alexa's requirement that users speak every 90 seconds, vs every two minutes for Google Home. So, without being forced, the storey had to incorporate a natural way for the listener to respond. Furthermore, voice-activated devices can only recognise a limited set of words. A scientific exam in which the listener is given a this-or-that type of question to respond, with those possibilities being part of the lexicon of voice-activated gadgets, ended up being a rather logical fit for the scenario. The BBC tried out a few additional prototypes before settling on The Inspection Chamber because of its technology and storyline.