M.S. Ramaiah Institute of Technology
(Autonomous Institute, Affiliated to VTU)
Department of Computer Science and Engineering

Course Name: Artificial Intelligence(AI)
Course Code: CS53
Credits: 3:0:0:1

Term: October 2021 – Feb 2022

Faculty:
Dr. Sangeetha V
Assistant Professor
Department of CSE

Unit I

Introduction: (Chapter 1 of Text Book 1)

- What is AI?
- Foundation of Artificial Intelligence
- History of Artificial Intelligence

Intelligent Agents: (Chapter 2 of Text Book 1)

- Agents and Environments
- Rationality
- The Nature of Environments
- The Structure of Agents

Problem-solving by search: (Chapter 3 of Text Book 1)

- Problem-Solving Agents
- Example Problems
- Searching for Solution
- Uniformed Search Strategies
- Informed Search Strategies
- Heuristic Functions

What is Artificial Intelligence?

Artificial intelligence (AI) is the study of ideas that enable computers to be intelligent.

The goals of the field of artificial intelligence are to make computers more useful and to understand the principles that make intelligence possible.

- Systems that think like humans ; Systems that think rationally
- Systems that act like humans ; Systems that act rationally

What is Artificial Intelligence? (contd..)

Thinking Humanly	Thinking Rationally
"The exciting new effort to make computers think . . . *machines with minds*, in the full and literal sense." (Haugeland, 1985) "[The automation of] activities that we associate with human thinking, activities such as decision-making, problem solving, learning . . ." (Bellman, 1978)	"The study of mental faculties through the use of computational models." (Charniak and McDermott, 1985) "The study of the computations that make it possible to perceive, reason, and act." (Winston, 1992)
Acting Humanly	**Acting Rationally**
"The art of creating machines that perform functions that require intelligence when performed by people." (Kurzweil, 1990) "The study of how to make computers do things at which, at the moment, people are better." (Rich and Knight, 1991)	"Computational Intelligence is the study of the design of intelligent agents." (Poole *et al.*, 1998) "AI . . . is concerned with intelligent behavior in artifacts." (Nilsson, 1998)

Some definitions of artificial intelligence, organized into four categories.

Eight definitions of AI, laid out along two dimensions.

Top - thought processes and reasoning,
Bottom - address behavior.

Left measure success - Human performance
Right ideal performance measure, called rationality

What is Artificial Intelligence? (contd..)

- Acting humanly: The Turing Test approach
- Thinking humanly: The cognitive modeling approach
- Thinking rationally: The “laws of thought” approach
- Acting rationally: The rational agent approach

What is Artificial Intelligence? (contd..)

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

What is Artificial Intelligence? (contd..)

- Acting humanly: The Turing Test approach

The computer would need to possess the following capabilities:

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

What is Artificial Intelligence? (contd..)

- Acting humanly: The Turing Test approach

The computer would need to possess the following capabilities:

- Computer vision to perceive objects,
- Robotics to manipulate objects and move about.

What is Artificial Intelligence? (contd..)

- Thinking humanly: The cognitive modeling approach
- If we are going to say that a given program thinks like a human, we must have some way of determining how humans think. We need to get inside the actual workings of human minds.
- There are three ways to do this:
  - through introspection—trying to catch our own thoughts as they go by;
  - through psychological experiments—observing a person in action; and
  - through brain imaging—observing the brain in action.

If the program's input–output behavior matches corresponding human behavior, that is evidence that some of the program's mechanisms could also be operating in humans.
Example : GPS in 1959, Allen Newell and Herbert Simon

What is Artificial Intelligence? (contd..)

- Thinking humanly: The cognitive modeling approach

What is Artificial Intelligence? (contd..)

- Thinking rationally: The “laws of thought” approach

What is Artificial Intelligence? (contd..)

- Thinking rationally: The “laws of thought” approach

Syllogisms

- Socrates has given the thinking process in terms of syllogisms which are very popular in philosophy
- Syllogisms provided patterns for argument structures that always yielded correct conclusions when given correct premises
- Example, “Socrates is a man; all men are mortal; therefore, Socrates is mortal.”

What is Artificial Intelligence? (contd..)

- Thinking rationally: The “laws of thought” approach

- These laws of thought were LOGIC supposed to govern the operation of the mind; their study initiated the field called logic
- There are two main obstacles to this approach.
  - First, it is not easy to take informal knowledge and state it in the formal terms required by logical notation, particularly when the knowledge is less than 100% certain.
  - Second, there is a big difference between solving a problem “in principle” and solving it in practice.

What is Artificial Intelligence? (contd..)

- Acting rationally: The rational agent approach

  An agent is just something that acts (Latin agree- to do).

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

What is Artificial Intelligence? (contd..)

- Acting rationally: The rational agent approach

- All computer programs do something, but computer agents are expected to do more: operate autonomously, perceive their environment, persist over a prolonged time period, adapt to change, and create and pursue goals.

What is Artificial Intelligence? (contd..)

- Acting rationally: The rational agent approach
- A rational agent is one that acts so as to achieve the best outcome or, when there is uncertainty, the best expected outcome.

Foundations of AI

Foundations of AI

• Philosophy	Logic, methods of reasoning, mind as physical system foundations of learning, language, rationality
• Mathematics	Formal representation and proof algorithms, computation, (un)decidability, (in)tractability, probability
• Economics	utility, decision theory
• Neuroscience	physical substrate for mental activity
• Psychology	phenomena of perception and motor control, experimental techniques
• Computer engineering	building fast computers
• Control theory	design systems that maximize an objective function over time
• Linguistics	knowledge representation, grammar

Foundations of AI-Philosophy

- Where does knowledge come from?
- How does knowledge lead to action?

- Aristotle (384–322 B.C.): was the first to formulate a precise set of laws governing the rational part of the mind.
- Thomas Hobbes (1588–1679) proposed that reasoning was like numerical computation, that "we add and subtract in our silent thoughts."
- Rene Descartes(1596-1650): Developed dualistic theory of mind and matter. Descartes attempted to demonstrate the existence of god and the distinction between the human soul and body.
- The empiricism movement, starting with Francis Bacon's (1561– 1626).
- The confirmation theory of Carnap and Carl Hempel (1905–1997) attempted to analyze the acquisition of knowedge from experience.
- Carnap's book The Logical Structure of the World (1928) defined an explicit computational procedure for extracting knowledge from elementary experiences. It was probably the first theory of mind as a computational process.

Foundations of AI- Mathematics

- What are the formal rules to draw valid conclusions?
- What can be computed?

- George Boole (1815–1864), who worked out the details of propositional, or Boolean, logic (Boole, 1847).
- In 1879, Gottlob Frege (1848–1925) extended Boole's logic to include objects and relations, creating the first order logic that is used today.
- The first nontrivial algorithm is thought to be Euclid's algorithm for computing greatest common divisors.
- Besides logic and computation, the third great contribution of mathematics to AI is the PROBABILITY theory of probability. The Italian Gerolamo Cardano (1501–1576) first framed the idea of probability, describing it in terms of the possible outcomes of gambling events.
- Thomas Bayes (1702–1761) proposed a rule for updating probabilities in the light of new evidence. Bayes' rule underlies most modern approaches to uncertain reasoning in AI systems.

Foundations of AI- Economics

- How should we make decisions to maximize payoff?
- How should we do this when the payoff may be far in the future?

- The science of economics got its start in 1776, Smith was the first to treat it as a science, using the idea that economies can be thought of as consisting of individual agents maximizing their own economic well-being.
- Decision theory, which combines probability theory with utility theory, provides a formal and complete framework for decisions (economic or otherwise) made under uncertainty.
- Von Neumann and Morgenstern's development of game theory included the surprising result that, for some games, a rational agent should adopt policies that are (or least appear to be) randomized. Unlike decision theory, game theory does not offer an unambiguous prescription for selecting actions.

Foundations of AI- Neuroscience

- How do brains process information?

- Neuroscience is the study of the nervous system, particularly the brain.
- 335 B.C. Aristotle wrote, “Of all the animals, man has the largest brain in proportion to his size.”
- Nicolas Rashevsky (1936, 1938) was the first to apply mathematical models to the study of the nervous system.
- The measurement of intact brain activity began in 1929 with the invention by Hans Berger of the electroencephalograph (EEG).
- The recent development of functional magnetic resonance imaging (fMRI) (Ogawa et al., 1990; Cabeza and Nyberg, 2001) is giving neuroscientists unprecedentedly detailed images of brain activity, enabling measurements that correspond in interesting ways to ongoing cognitive processes.

Foundations of AI - Psychology

- How do humans and animals think and act?

➢ Behaviorism movement, led by John Watson(1878-1958). Behaviorists insisted on studying only objective measures of the percepts(stimulus) given to an animal and its resulting actions(or response). Behaviorism discovered a lot about rats and pigeons but had less success at understanding human.

➢ Cognitive psychology , views the brain as an information processing device.

Foundations of AI- Computer Engineering

- How can we build an efficient computer?

- The first operational computer was the electromechanical Heath Robinson,8 built in 1940 by Alan Turing's team for a single purpose: deciphering German messages.
- The first operational programmable computer was the Z-3, the invention of Konrad Zuse in Germany in 1941.
- The first electronic computer, the ABC, was assembled by John Atanasoff and his student Clifford Berry between 1940 and 1942 at Iowa State University.
- The first programmable machine was a loom, devised in 1805 by Joseph Marie Jacquard (1752–1834), that used punched cards to store instructions for the pattern to be woven.

Foundations of AI- Control theory and Cybernetics

- How can artifacts operate under their own control?

➢ Ktesibios of Alexandria (c. 250 B.C.) built the first self-controlling machine: a water clock with a regulator that maintained a constant flow rate. This invention changed the definition of what an artifact could do.

➢ Modern control theory, especially the branch known as stochastic optimal control, has as its goal the design of systems that maximize an objective function over time. This roughly OBJECTIVE FUNCTION matches our view of AI: designing systems that behave optimally.

Foundations of AI-Linguistics

- How does language relate to thought?

- In 1957, B. F. Skinner published Verbal Behavior. This was a comprehensive, detailed account of the behaviorist approach to language learning, written by the foremost expert in the field.
- Noam Chomsky, who had just published a book on his own theory, Syntactic Structures. Chomsky pointed out that the behaviorist theory did not address the notion of creativity in language.
- Modern linguistics and AI, then, were “born” at about the same time, and grew up together, intersecting in a hybrid field called computational linguistics or natural language processing.

History of Artificial Intelligence

- 1943 McCulloch & Pitts: Boolean circuit model of brain
- 1950 Turing's "Computing Machinery and Intelligence"
- 1956 Dartmouth meeting: "Artificial Intelligence" adopted
- 1952—69 Look, Ma, no hands!
- 1950s Early AI programs, including Samuel's checkers program, Newell & Simon's Logic Theorist, Gelernter's Geometry Engine
- 1965 Robinson's complete algorithm for logical reasoning
- 1966—73 AI discovers computational complexity
  Neural network research almost disappears
- 1969—79 Early development of knowledge-based systems
- 1980-- AI becomes an industry
- 1986-- Neural networks return to popularity
- 1987-- AI becomes a science
- 1995-- The emergence of intelligent agents

History of Artificial Intelligence

Stone age (1943-1956)

- Early work on neural networks and logic (Herbert Simon,1943)
- Birth of AI: Dartmouth workshop - summer 1956
- John McCarthy's name for the field: artificial intelligence

Herbert Simon in 1991

John McCarthy in AAAI-2002

Early enthusiasm, great expectations (1952-1969)

Dark ages (1969-1973)Why?

- AI did not scale up: combinatorial explanation.
- Failure of natural language translation approach based on simple grammars and word dictionary.
- Funding for natural language processing stopped.
- Failure of perceptron to learn such a simple function.
- Realization of the difficulty of the learning process.
- There is no Symbolic concept learning.

Renaissance (1969-1979)

Change of problem solving paradigm: From search-based problem solving to knowledge-based problem solving.

Industrial age (1980-present)

- Building expert systems with formal knowledge acquisition techniques.
- There exists successful commercial expert systems.
- Many AI companies.
- Exploration of different learning strategies.
- Reasoning, Genetic algorithms, Neural networks, etc.)

The Maturity and return of neural networks (1986-1995)

- Many successful applications of neural networks , machine learning, and data mining.
- Base claims on theorems and experiments rather than on intuition.
- Show relevance to real-world applications rather than toy examples.
- The reinvention of the back propagation learning algorithm for neural networks and other optimization process.

Intelligent agents (1995-present)

- The realization that the previously isolated subfields of AI (speech recognition, planning, robotics, computer vision, machine learning, knowledge representation, etc.) need to be reorganized when their results are to be tied together into a single agent design.
- A process of reintegration of different sub-areas of AI to build a “whole agent”:

Multi-agent systems;

Agents for different types of applications, web search engine agents.

- http://www.stottlerhenke.com/ai_general/history.htm

1997: The Deep Blue chess program beats the current world chess champion, Garry Kasparov, in a widely followed match.

First official Robo-Cup soccer match featuring table-top matches with 40 teams of interacting robots and over 5000 spectators.

Late 90's: Web crawlers and other AI-based information extraction programs become essential in widespread use of the world-wide-web. Demonstration of an Intelligent Room and Emotional Agents at MIT's AI Lab. Initiation of work on the Oxygen Architecture, which connects mobile and stationary computers in an adaptive network.

2000: Interactive robot pets (a.k.a. "smart toys") become commercially available, realizing the vision of the 18th cen. novelty toy makers.
Cynthia Breazeal at MIT publishes her dissertation on Sociable Machines, describing KISMET, a robot with a face that expresses emotions. The Nomad robot explores remote regions of Antarctica looking for meteorite samples.

http://www.aaai.org/AITopics/html/current.html for history in the making!

Unit I

Introduction: (Chapter 1 of Text Book 1)

- What is AI?
- Foundation of Artificial Intelligence
- History of Artificial Intelligence

Intelligent Agents: (Chapter 2 of Text Book 1)

- Agents and Environments
- Rationality
- The Nature of Environments
- The Structure of Agents

Problem-solving by search: (Chapter 3 of Text Book 1)

- Problem-Solving Agents
- Example Problems
- Searching for Solution
- Uniformed Search Strategies
- Informed Search Strategies
- Heuristic Functions

Agents

- An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators
- Perceive the environment through **sensors** (→Percepts)
- Act upon the environment through **actuators** (→Actions)

Agents

- Examples:
  Human agent
  Robotic agent
  Software agent
  - Receives keystrokes as sensory inputs and acts on the environment by displaying on the screen
  - Receives file contents as sensory inputs and acts on the environment by writing files
  - Receives network packets as sensory inputs and acts on the environment by sending network packets.

Agents

- Percept: the agent's perceptual inputs at any given instant
- Percept sequence: the complete history of everything the agent has perceived
- Agent function maps any given percept sequence to an action
- Agent program runs on the physical architecture to produce
- Agent = architecture + program

Agents -> Example : Vacuum-Cleaner World

- Percepts: location and contents, e.g., [A, dirty]
- Actions: Left, Right, Suck, NoOp

Percept sequence	Action
[A, Clean]	Right
[A, Dirty]	Suck
[B, Clean]	Left
[B, Dirty]	Suck
[A, Clean], [A, Clean]	Right
[A, Clean], [A, Dirty]	Suck
⋮	⋮
[A, Clean], [A, Clean], [A, Clean]	Right
[A, Clean], [A, Clean], [A, Dirty]	Suck
⋮	⋮

Partial tabulation of a simple agent function for the vacuum-cleaner world

Rationality

- An agent should "do the right thing", based on what it can perceive and the actions it can perform. The right action is the one that will cause the agent to be most successful
- Performance measure: An objective criterion for success of an agent's behavior
- Back to the vacuum-cleaner example
  - Amount of dirt cleaned within certain time
  - +1 credit for each clean square per unit time
- General rule: measure what one wants rather than how one thinks the agent should behave

Rationality- Rational Agent

What is rational at any given time depends on four things:

- The performance measure that defines the criterion of success.
- The agent's prior knowledge of the environment.
- The actions that the agent can perform.
- The agent's percept sequence to date.
- Definition:
  - For each possible percept sequence, a rational agent should select an action that is expected to maximize its performance measure, given the evidence provided by the percept sequence and whatever built-in knowledge the agent has.

Rationality –Vacuum Cleaner Example

- A simple agent that cleans a square if it is dirty and moves to the other square if not
- Is it rational?
- Assumption:
  - performance measure: 1 point for each clean square at each time step
  - The "geography" of the environment is known a priori
  - actions = {left, right, suck, no-op}
  - agent is able to perceive the location and dirt in that location

Unit I

Introduction: (Chapter 1 of Text Book 1)

- What is AI?
- Foundation of Artificial Intelligence
- History of Artificial Intelligence

Intelligent Agents: (Chapter 2 of Text Book 1)

- Agents and Environments
- Rationality
- The Nature of Environments
- The Structure of Agents

Problem-solving by search: (Chapter 3 of Text Book 1)

- Problem-Solving Agents
- Example Problems
- Searching for Solution
- Uniformed Search Strategies
- Informed Search Strategies
- Heuristic Functions

Rationality - Omniscience, Learning and Autonomy

- Omniscience is the state of possessing unlimited knowledge about all things possible.
- Omniscience is knowing everything
- Expert chess player exemplifies a kind of omniscience by determining his/her opponent's moves, but each possible move opens up certain countermoves.
- An omniscient agent knows the actual outcome of its actions and can act accordingly; but omniscience is impossible in reality.

Rationality - Omniscience, Learning and Autonomy

- Distinction between rationality and omniscience
  - expected performance vs. actual performance
- Agents can perform actions in order to modify future percepts so as to obtain useful information (information gathering, exploration)
- An agent can also learn from what it perceives
- An agent is autonomous if its behavior is determined by its own experience (with ability to learn and adapt)

The Nature of Environments

To build rational agents - Think about task environments are the "problems" to which rational agents are the "solutions."

- How to specify a task environment, illustrating the process with a number of examples.
- Task environments come in a variety of flavors.
- The flavor of the task environment directly affects the appropriate design for the agent program.

The Nature of Environments

- Specifying the task environment is always the first step in designing agent

- PEAS: **P**erformance, **E**nvironment, **A**ctuators, **S**ensors

P	**Performance** – how we measure the system's achievements
E	**Environment** – what the agent is interacting with
A	**Actuators** – what produces the outputs of the system
S	**Sensors** – what provides the inputs to the system

Taxi Driver Example

Agent Type	Performance Measure	Environment	Actuators	Sensors
Taxi driver	Safe, fast, legal, comfortable trip, maximize profits	Roads, other traffic, pedestrians, customers	Steering, accelerator, brake, signal, horn, display	Cameras, sonar, speedometer, GPS, odometer, accelerometer, engine sensors, keyboard

PEAS description of the task environment for an automated taxi.

Mushroom-Picking Robot

Performance Measure	Environment	Actuators	Sensors
Percentage of good mushrooms in correct bins	Conveyor belt with mushrooms, bins	Jointed arm and hand	camera, joint angle sensors

Medical Diagnosis System

Performance Measure	Environment	Actuators	Sensors
healthy patient, minimize costs, lawsuits	patient, hospital, staff	display questions, tests, diagnosis, treatments, referrals	keyboard entry of symptoms , findings, patient's answers

RAMAIAH

Agent Type	Performance Measure	Environment	Actuators	Sensors
Medical diagnosis system	Healthy patient, reduced costs	Patient, hospital, staff	Display of questions, tests, diagnoses, treatments, referrals	Keyboard entry of symptoms, findings, patient's answers
Satellite image analysis system	Correct image categorization	Downlink from orbiting satellite	Display of scene categorization	Color pixel arrays
Part-picking robot	Percentage of parts in correct bins	Conveyor belt with parts; bins	Jointed arm and hand	Camera, joint angle sensors
Refinery controller	Purity, yield, safety	Refinery, operators	Valves, pumps, heaters, displays	Temperature, pressure, chemical sensors
Interactive English tutor	Student's score on test	Set of students, testing agency	Display of exercises, suggestions, corrections	Keyboard entry

Examples of agent types and their PEAS descriptions.

Properties of Task Environments

- Fully observable vs partially observable
- Deterministic vs stochastic
- Episodic vs sequential
- Static vs dynamic
- Discrete vs continuous
- Single agent vs multiagent

Fully observable/partially observable/unobservable

Fully observable

- An agent can always see the entire state of an environment.
- **Example**: Chess

Partially Observable

- An agent can never see the entire state of an environment.
- **Example**: Card game

Fully observable/partially observable/unobservable

If an agent's sensors give it access to the complete state of the environment at each point in time, then we say that the task environment is fully observable.

Example: Automated taxi driver

An environment might be partially observable because of noisy and inaccurate sensors or because parts of the state are simply missing from the sensor data.

Example: Automated taxi driver- night driving

If the agent has no sensors at all then the environment is **unobservable**.

Example: Industrial Robots

RAMAIAH
Institute of Technology

Deterministic v/s stochastic

Deterministic

- An agent's current state and selected action can completely determine the next state of the environment.
- **Example:** Tic tac toe

Stochastic

- A stochastic environment is random in nature and cannot be determined completely by an agent.
- **Example**: Ludo (Any games that involve dice)

Deterministic v/s stochastic

If the next state of the environment is completely determined by the current state and the action executed by the agent, then we say the environment is deterministic; otherwise, it is stochastic.

Deterministic- Cross Word Puzzle

Stochastic- Poker

Episodic v/s sequential

Episodic

- Only the current percept is required for the action
- Every episode is independent of each other
- **Example**: part picking robot

Sequential

- An agent requires memory of past actions to determine the next best actions.
- The current decision could affect all future decisions.
- **Example**: Chess

Episodic v/s sequential

In an episodic task environment, the agent's experience is divided into atomic episodes. In each episode the agent receives a percept and then performs a single action.

In sequential environments, on the other hand, the current decision could affect all future decisions.

Episodic- Part Picking Robot

Sequential- Chess, Taxi driving

Static v/s dynamic

Static

- The environment does not change while an agent is acting
- **Example**: Crossword puzzles

Dynamic

- The environment may change over time
- **Example**: roller coaster ride

  Taxi driving

Static v/s dynamic

If the environment can change while an agent is deliberating, then we say the environment is dynamic for that agent; otherwise, it is static.

Static- Cross Word Puzzle

Dynamic- Taxi Driving

Discrete v/s continuous

Discrete

- The environment consists of a finite number of actions that can be deliberated in the environment to obtain the output.
- **Example**: chess

Continuous

- The environment in which the actions performed cannot be numbered ie.,is not discrete, is said to be continuous.
- **Example**: Self-driving cars

Discrete v/s continuous

The discrete/continuous distinction applies to the *state* of the environment, to the way *time* is handled, and to the *percepts* and *actions* of the agent.

Discrete- Cross Word Puzzle, Poker..

Continuous- Taxi Driving

Single agent v/s multi-agent

Single-agent

- If only one agent is involved in an environment, and operating by itself then such an environment is called single agent environment.
- **Example**: maze

Multi-agent

- if multiple agents are operating in an environment, then such an environment is called a multi-agent environment.
- **Example**: football

RAMAIAH
Institute of Technology

Single agent v/s multi-agent

An agent solving a crossword puzzle by itself is clearly in a single-agent environment, whereas an agent playing chess is in a two-agent environment.

Known

- In a known environment, the results for all actions are known to the agent.
- **Example**: Card game

Unknown

- In unknown environment, agent needs to learn how it works in order to perform an action.
- **Example**: A new video game

Examples

Task Environment	Oberservable	Deterministic	Episodic	Static	Discrete	Agents
Crossword puzzle	fully	deterministic	sequential	static	discrete	single
Chess with a clock	fully	strategic	sequential	semi	discrete	multi
Taxi driver	partially	stochastic	sequential	dynamic	conti.	multi
mushroom-picking	partially	stochastic	episodic	dynamic	conti.	single

- The environment type largely determines the agent design
- The real world is (of course) partially observable, stochastic, sequential, dynamic, continuous, multi-agent

The Structure of Agents

- Let us understand the different kinds of agent programs and how to convert them into learning agents that can improvise their agent function and generate better actions.

The Structure of Agents

- An intelligent agent is a combination of Agent Program and Architecture.
- Intelligent Agent = Agent Program + Architecture
- The agent program takes in the current percept of the environment from the sensors of the agent and returns an action to be performed by the actuators.
- Architecture is a computing device used to run the agent program.

The Structure of Agents

Agent Program vs. Agent Function

- Agent program takes the current percept as input
  - Nothing is available from the environment
- Agent function takes the entire percept history
  - To do this, remember all the percepts

Table-Driven Agent

- Designer needs to construct a table that contains the appropriate action for every possible percept sequence

- Drawbacks?
  - huge table
  - take a long time to construct such a table
  - no autonomy
  - Even with learning, need a long time to learn the table entries

Basic Agent Types/Agent programs

- Arranged in order of increasing generality:
  - Simple reflex agents
  - Model-based reflex agents
  - Goal-based agents
  - Utility-based agents; and
  - Learning agents

Basic Agent Types

Simple Reflex Agent

- They choose actions only based on the **current situation** ignoring the history of perceptions.
- Perform actions only on simple situation.
- They will work only if the environment is **fully observable**
- The agent function is based on the **condition-action rule**: "if condition, then action".
- This can only be done based on the pre-determined rules that are present in the knowledge base
- **Example**: if car-in-front-brakes

Simple Reflex Agent

Five Basic Agent Types

Simple reflex agents

Example: Medical diagnosis system

if the patient has reddish brown spots then start the treatment for measles.

RAMAIAH
Institute of Technology

Pseudo-Code

```
function SIMPLE-REFLEX-AGENT(percept) returns an action
   persistent: rules, a set of condition–action rules

   state ← INTERPRET-INPUT(percept)
   rule ← RULE-MATCH(state, rules)
   action ← rule.ACTION
   return action
```

INTERRUPT-INPUT – function generates an abstracted description of the current state from the percept.

RULE-MATCH – function returns the first rule in the set of rules that matches the given state description.

RULE-ACTION – the selected rule is executed as action of the given percept.

Pseudo-Code

```
function SIMPLE-REFLEX-AGENT(percept) returns an action
   persistent: rules, a set of condition–action rules

   state ← INTERPRET-INPUT(percept)
   rule ← RULE-MATCH(state, rules)
   action ← rule.ACTION
   return action
```

```
function REFLEX-VACUUM-AGENT([location,status]) returns an action

   if status = Dirty then return Suck
   else if location = A then return Right
   else if location = B then return Left
```

- Infinite loops are often unavoidable for simple reflex agent operating in partially observable environments
  - No location sensor

- Randomization will help
- A randomized simple relfex agent might outperform a deterministic simple reflex agent

- Better way: keep track of the part of the world it can't see now
  - Maintain internal states

Model-Based Reflex Agent

- A model-based reflex agent is an intelligent agent that uses **percept history** and internal memory to make decisions about the "**model**" of the world around it.
- The Model-based agent can work in a **partially observable environment**, and track the situation.
- **Model** – knowledge about "how the things happen in the world".
- **Internal State** – It is a representation of unobserved aspects of current state depending on percept history.
- **Updating the state requires the information about**
  - How the world evolves.
  - How the agent's actions affect the world.

Model-Based Reflex Agent

Pseudo-Code

```
function MODEL-BASED-REFLEX-AGENT(percept) returns an action
  persistent: state, the agent's current conception of the world state
              model, a description of how the next state depends on current state and action
              rules, a set of condition–action rules
              action, the most recent action, initially none

  state ← UPDATE-STATE(state, action, percept, model)
  rule ← RULE-MATCH(state, rules)
  action ← rule.ACTION
  return action
```

Goal-Based Agent

- They choose their actions in order to **achieve goals**.
- This allows the agent a way to choose among multiple possibilities, selecting the one which reaches a goal state.
- They usually require **search** and **planning**.
- **Example** : A GPS system finding a path to certain destination

Basic Agent Types

Goal-based reflex agents

- An agent knows the description of current state and also needs some sort of goal information that describes situations that are desirable.
- The action matches with the current state is selected depends on the goal state.
- The goal based agent is more flexible for more than one destination also.
- After identifying one destination, the new destination is specified, goal based agent is activated to come up with a new behavior.

Basic Agent Types

Goal-based reflex agents

- Search and Planning are the subfields of AI devoted to finding action sequences that achieve the agents goals.
- The goal-based agent appears less efficient,
- it is more flexible because the knowledge that supports its decisions is represented explicitly and can be modified.
- The goal-based agent‘s behavior can easily be changed to go to a different location.

Goal-Based Agent

Utility-Based Agent

- A utility-based agent is an agent that acts based not only on what the goal is, but the **best way** to reach that goal.
- They choose actions based on a preference (utility) for each state.
- **Example** : A GPS system finding a shortest/fastest/safer to certain destination.

Basic Agent Types

Utility-based reflex agents

- An agent generates a goal state with high – quality behavior (utility) that is, if more than one sequence exists to reach the goal state then the sequence with more reliable, safer, quicker and cheaper than others to be selected.

- A utility function maps a state (or sequence of states) onto a real number, which describes the associated degree of happiness.

Basic Agent Types

Utility-based reflex agents

The utility function can be used for two different cases:

- First, when there are conflicting goals, only some of which can be achieved (for e.g., speed and safety), the utility function specifies the appropriate tradeoff.
- Second, when the agent aims for several goals, none of which can be achieved with certainty, then the success can be weighted up against the importance of the goals.

Utility-Based Agent

Utility Function

- Utility function maps a state or a sequence of states onto a real number □ degree of happiness

- Conflicting goals
  - Speed and safety
- Multiple goals

Learning Agent

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

Learning Agent

Unit I

Introduction: (Chapter 1 of Text Book 1)

- What is AI?
- Foundation of Artificial Intelligence
- History of Artificial Intelligence

Intelligent Agents: (Chapter 2 of Text Book 1)

- Agents and Environments
- Rationality
- The Nature of Environments
- The Structure of Agents

Problem-solving by search: (Chapter 3 of Text Book 1)

- Problem-Solving Agents
- Example Problems
- Searching for Solution
- Uniformed Search Strategies
- Informed Search Strategies
- Heuristic Functions

Problem Solving by Searching

Reflex agent is simple

- base their actions on
- a direct mapping from states to actions
- but cannot work well in environments
  - which this mapping would be too large to store
  - and would take too long to learn

Hence, goal-based agent is used

Problem-Solving Agents

Limiting the Objectives
Current State

Actions and States

Looking for a sequence of actions

Search Algorithm
takes a problem as input and returns a **solution**

Problem-solving agent

- A kind of goal-based agent
- It solves problem by
  - finding sequences of actions that lead to desirable states (goals)
- To solve a problem,
  - the first step is the ***goal formulation***, based on the current situation

Goal formulation

The goal is formulated

- as a set of world states, in which the goal is satisfied

Reaching from initial state goal state

- Actions are required

Actions are the operators

- causing transitions between world states
- **Actions** should be abstract enough at a certain degree, instead of very detailed
- E.g., <u>turn left</u> VS <u>turn left 30 degree</u>, etc.

Problem formulation

The process of deciding

- what actions and states to consider, given a goal

Example: **<u>Travelling agent</u>**

drive Arad Bucharest

- in-between states and actions defined
- States: Some places in Sibiu, Timisoara, Zerind.
- Actions: Turn left, Turn right, go straight, accelerate & brake, etc.

Search

Because there are many ways to achieve the same goal

- Those ways are together expressed as a tree
- Multiple options of unknown value at a point,
  - the agent can examine different possible sequences of actions, and choose the best
- This process of looking for the best sequence is called ***search***
- The best sequence is then a list of actions, called ***solution***

Search algorithm

Defined as

- taking a *problem* as input
- and returns a *solution* in the form of an action sequence

Once a solution is found

- the agent follows the solution
- and carries out the list of actions – execution phase

Design of an agent

- "Formulate, search, execute"

Design of an agent "Formulate, search, execute"

```
function SIMPLE-PROBLEM-SOLVING-AGENT(percept) returns an action
  persistent: seq, an action sequence, initially empty
              state, some description of the current world state
              goal, a goal, initially null
              problem, a problem formulation

  state ← UPDATE-STATE(state, percept)
  if seq is empty then
      goal ← FORMULATE-GOAL(state)
      problem ← FORMULATE-PROBLEM(state, goal)
      seq ← SEARCH(problem)
      if seq = failure then return a null action
  action ← FIRST(seq)
  seq ← REST(seq)
  return action
```

Figure 3.1 A simple problem-solving agent. It first formulates a goal and a problem, searches for a sequence of actions that would solve the problem, and then executes the actions one at a time. When this is complete, it formulates another goal and starts over.

Well-defined problems and solutions

A problem is defined by 5 components:

- Initial state
- Possible actions
- Transition model
- Goal Test
- Path cost

Well-defined problems and solutions

A problem is defined by 5 components:

- The initial state that the agent starts in.

  For example, the initial state for our agent in Romania might be described as In(Arad)

- A description of the possible actions available to the agent.

  Given a particular state s, ACTIONS(s) returns the set of actions that can be executed in s.

  In(Arad), the applicable actions are {Go(Sibiu), Go(Timisoara), Go(Zerind)}.

Well-defined problems and solutions

- TRANSITION MODEL

A description of what each action does;

Specified by a function RESULT(s, a) that returns the state that results from SUCCESSOR doing action a in state s.

RESULT(In(Arad),Go(Zerind)) = In(Zerind) .

Figure 3.2 A simplified road map of part of Romania.

Well-defined problems and solutions

- TRANSITION MODEL

Together, the initial state, actions, and transition model implicitly define the State Space of the problem

The state space forms a directed network or graph in which the nodes are states and the links between nodes are actions.

A path in the state space is a sequence of states connected by a sequence of actions.

Figure 3.2 A simplified road map of part of Romania.

Well-defined problems and solutions

- Goal Test

Determines whether a given state is a goal state.

The agent's goal in Romania is the singleton set {In(Bucharest)}.

Example: In chess, the goal is to reach a state called "checkmate," where the opponent's king is under attack and can't escape

Well-defined problems and solutions

- Path cost

The problem-solving agent chooses a cost function that reflects its own performance measure.

Path cost function assigns a numeric cost to each path.

Cost of a path can be described as the sum of the costs of the individual actions along the path.

The step cost of taking action a in state s to reach state s' is

denoted by c(s, a, s').

Well-defined problems and solutions

- Path cost

Example :

Agent – Arad to Bucharest, time is of the essence, so the cost of a path might be its length in kilometers.

Summarize Well-defined problems and solutions

Together a problem is defined by

- Initial state
- Actions
- Successor function
- Goal test
- Path cost function

The ***solution*** of a problem is then

- *a path from the initial state to a state satisfying the goal test*

Optimal solution

- the solution with lowest path cost among all solutions

Formulating problems

We proposed a formulation of the problem - Arad to Bucharest in terms of the initial state, actions, transition model, goal test, and path cost.

This formulation seems reasonable, but it is still a model—an abstract mathematical description

Abstract of State description

Formulating problems

State of the world includes so many things: the traveling companions, the current radio program, the scenery out of the window, the proximity of law enforcement officers, the distance to the next rest stop, the condition of the road, the weather, and so on

All these considerations are left out of our state descriptions because they are irrelevant to the problem of finding a route to Bucharest.

The process of removing detail from a representation is called **Abstraction.**

Example problems

Two main varieties of problem faced in AI games.

Toy problems

- those intended to illustrate or exercise various problem-solving methods
- E.g., puzzle, chess, etc.

Real-world problems

- tend to be more difficult and whose solutions people actually care about
- E.g., Design, planning, etc.

Example problems

Toy problems - Vacuum world

- States: The state is determined by both the agent location and the dirt locations.

  A larger environment with n locations has $n \cdot 2^n$ states.

  Thus, there are $2 \times 2^2 = 8$ possible world states.
- Initial state: Any state can be designated as the initial state.
- Actions: Left, Right, and Suck.

Example problems

Toy problems - Vacuum world

- Transition model: The actions have their expected effects, except that moving Left in the leftmost square, moving Right in the rightmost square, and Sucking in a clean square have no effect.

The complete state space is shown in Figure

Toy problems

- Number of states: 8
- Initial state: Any
- Number of actions: 3
  - left, right, suck

  Goal: clean up all dirt
  - Goal states: {7, 8}
- Path Cost:
  - Each step costs 1

Figure 3.3 The state space for the vacuum world. Links denote actions: L = *Left*, R = *Right*, S = *Suck*.

Example problems

Toy problems -8 Puzzle

The 8-puzzle belongs to the family of sliding-block puzzles, which are often used as test problems for new search algorithms in AI.

- It consists of 3 x 3 board with 8 numbered tiles and a blank space
- A tile adjacent to the blank space can slide into the blank space.
- The objective is to reach the specified goal state.

Start State

Goal State

Example problems

- Initial state: Any state can be designated as the initial state.
- Actions: Movements of the blank space Left, Right, Up, or Down.
- Transition model: Given a state and action, this returns the resulting state
  For example, if we apply Left to the start state in Figure, the resulting state has the 5 and the blank switched.
- Goal test: This checks whether the state matches the goal configuration
- Path cost: Each step costs 1, so the path cost is the number of steps in the path.

Start State Goal State

Example problems

The 8-puzzle has 9!/2 = 181, 440 reachable states and is easily solved.

2	1	6
4		8
7	5	3

Initial State

1	2	3
8		4
7	6	5

Goal State

Example problems

Toy problems - 8-queens

- n queens puzzle of placing 8 queens on an n×n chessboard
- 8 chess queens on an 8×8 chessboard such that none of them is able to capture any other using the standard chess queen's moves.
- Any queen is assumed to be able to attack any other.
- Thus, a solution requires that no two queens share the same row, column, or diagonal.

RAMAIAH
Institute of Technology

Example problems

Toy problems - 8-queens

There are two main kinds of formulation.

An incremental formulation involves operators that augment the state description, starting with an empty state; for the 8-queens problem, this means that each action adds a queen to the state.

A complete-state formulation starts with all 8 queens on the board and moves them around.

Example problems

Toy problems - 8-queens

As per the incremental formulation :

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

Example problems

Real-world problems

- Route-finding algorithms are used in a variety of applications.
- Others, such as routing video streams in computer networks, military operations planning, and airline travel-planning systems, involve much more complex specifications.

Example problems

Real-world problems

1. Airline travel problems
2. Touring problems
3. Traveling salesperson problem
4. A VLSI layout
5. Robot navigation
6. Automatic assembly sequencing

Example problems

1. Airline travel problems

- **States**: Each state obviously includes a location (e.g., an airport) and the current time. Furthermore, because the cost of an action (a flight segment) may depend on previous segments, their fare bases, and their status as domestic or international, the state must record extra information about these "historical" aspects.
- **Initial state**: This is specified by the user's query.
- **Actions**: Take any flight from the current location, in any seat class, leaving after the current time, leaving enough time for within-airport transfer if needed.
- **Transition model**: The state resulting from taking a flight will have the flight's destination as the current location and the flight's arrival time as the current time.
- **Goal test**: Are we at the final destination specified by the user?
- **Path cost**: This depends on monetary cost, waiting time, flight time, customs and immigration procedures, seat quality, time of day, type of airplane, frequent-flyer mileage awards, and so on.

2.Touring problems

Are closely related to route-finding problems

Problem - "Visit every city at least once, starting and ending in Bucharest".

Figure 3.2 A simplified road map of part of Romania.

In(Bucharest), Visited({Bucharest}),

In(Vaslui), Visited({Bucharest, Urziceni, Vaslui})

3. Traveling salesperson problem

Is a touring problem in which each city must be visited exactly once.

The aim is to find the shortest tour.

The problem is known to be NP-hard, but an enormous amount of effort has been expended to improve the capabilities of TSP algorithms.

Figure 3.2 A simplified road map of part of Romania.

Unit I

Introduction: (Chapter 1 of Text Book 1)

- What is AI?
- Foundation of Artificial Intelligence
- History of Artificial Intelligence

Intelligent Agents: (Chapter 2 of Text Book 1)

- Agents and Environments
- Rationality
- The Nature of Environments
- The Structure of Agents

Problem-solving by search: (Chapter 3 of Text Book 1)

- Problem-Solving Agents
- Example Problems
- Searching for Solution
- Uniformed Search Strategies
- Informed Search Strategies
- Heuristic Functions

Searching for solutions

Searching for solutions

All problems are transformed

- as a search tree generated by the initial state and successor function

Finding out a solution is done by

- Searching through the state space
- Represent state space with Tree and Graph

Search tree

Initial state

- The root of the search tree is a **search node**

Expanding

- applying successor function to the current state thereby generating a new set of states

Leaf nodes

- a node with no children in the tree.

The set of all leaf nodes available for expansion at any given point is called the **Frontier(Open list) .**

The process of expanding nodes on the frontier continues until either a solution is found or there are no more states to expand.

Figure shows the first few steps in growing the search tree for finding a route from Ara

to Bucharest.

Figure 3.2 A simplified road map of part of Romania.

Search algorithms share this basic structure.

They vary primarily according to 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
```

Search algorithms share this basic structure.

In(Arad) is a repeated state in the search tree, generated in this case by a **loopy path.**

Loops can cause certain algorithms to fail, making otherwise solvable problems unsolvable

Loopy paths are a special case of redundant paths

redundant paths are unavoidable.

The way to avoid exploring redundant paths is to remember where one has been.

Figure 3.2 A simplified road map of part of Romania.

Search algorithms share this basic structure.

To do this, we augment the TREE-SEARCH algorithm with a data structure called the explored set (also known as the closed list), which remembers every expanded node.

Newly generated nodes that match previously generated nodes—ones in the explored set or the frontier—can be discarded instead of being added to the frontier.

The new algorithm, called GRAPH-SEARCH, is shown in figure

RAMAIAH Institute of Technology

```
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 frontier using the initial state of problem
  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 the frontier
         only if not in the frontier or explored set
```

Figure 3.2 A simplified road map of part of Romania.

Figure 3.8 A sequence of search trees generated by a graph search on the Romania problem of Figure 3.2. At each stage, we have extended each path by one step. Notice that at the third stage, the northernmost city (Oradea) has become a dead end: both of its successors are already explored via other paths.

Rectangular grid Problem

Figure 3.2 A simplified road map of part of Romania.

Route finding on a rectangular grid is important example in computer games.

In grid, each state has four successors, so a search tree of depth d that includes repeated states has 4^d leaves; but there are only about $2d^2$ distinct states within d steps of any given state.

For d = 20, this means about a trillion nodes but only about 800 distinct states.

Graph search algorithm

The algorithm has another nice property:

The frontier separates the state-space graph into the explored region and the unexplored region, so that every path from the initial state to an unexplored state has to pass through a state in the frontier.

This property is illustrated in next Figure.

Figure 3.2 A simplified road map of part of Romania.

Figure 3.9 The separation property of GRAPH-SEARCH, illustrated on a rectangular-grid problem. The frontier (white nodes) always separates the explored region of the state space (black nodes) from the unexplored region (gray nodes). In (a), just the root has been expanded. In (b), one leaf node has been expanded. In (c), the remaining successors of the root have been expanded in clockwise order.

Infrastructure for search algorithms

- Search algorithms require a data structure to keep track of the search tree that is being constructed.
- For each node n of the tree, we have a structure that contains four components

  - n.STATE: the state in the state space to which the node corresponds;
  - n.PARENT: the node in the search tree that generated this node;
  - n.ACTION: the action that was applied to the parent to generate the node;
  - n.PATH-COST: the cost, traditionally denoted by $g(n)$, of the path from the initial state to the node, as indicated by the parent pointers.

The node data structure is depicted in Figure

Nodes are the data structures from which the search tree is constructed. Each has a parent, a state, and various bookkeeping fields. Arrows point from child to parent.

- n.STATE: the state in the state space to which the node corresponds;
- n.PARENT: the node in the search tree that generated this node;
- n.ACTION: the action that was applied to the parent to generate the node;
- n.PATH-COST: the cost, traditionally denoted by $g(n)$, of the path from the initial state to the node, as indicated by the parent pointers.

RAMAIAH
Institute of Technology

The function CHILD-NODE takes a parent node and an action and returns the resulting child node:

```
function CHILD-NODE(problem, parent, action) returns a node
  return a node with
      STATE = problem.RESULT(parent.STATE, action),
      PARENT = parent, ACTION = action,
      PATH-COST = parent.PATH-COST + problem.STEP-COST(parent.STATE, action)
```

Data structure -Queue

We need to put Nodes somewhere.

The frontier needs to be stored in such a way that the search algorithm can easily choose the next node to expand according to its preferred strategy.

The appropriate data structure for this is a queue.

The operations on a queue are as follows:

- EMPTY?(*queue*) returns true only if there are no more elements in the queue.
- POP(*queue*) removes the first element of the queue and returns it.
- INSERT(*element*, *queue*) inserts an element and returns the resulting queue.

Data structure -Queue

Three common Queue variants are

- First-in First-out (FIFO) : pops the oldest element of the queue
- Last-in First-out(LIFO) : pops the newest element of the queue
- Priority queue: pops the element of the queue with the highest priority according to some ordering function.

Measuring problem-solving performance

The evaluation of a search strategy in 4 ways:

- **Completeness:**
  - is the strategy guaranteed to find a solution when there is one?
- **Optimality**:
  - does the strategy find the highest-quality solution when there are several different solutions?
- **Time complexity**:
  - how long does it take to find a solution?
- **Space complexity**:
  - how much memory is needed to perform the search?

Measuring problem-solving performance

Graph is an explicit data structure that is input to the search program.

The typical measure is the size of the state space graph, $|V| + |E|$

where V is the set of vertices (nodes) of the graph

E is the set of edges (links).

In AI, the graph is often represented implicitly by the initial state, actions, and transition model

Measuring problem-solving performance

In AI, complexity is expressed with 3 quantities

- **b**, branching factor, maximum number of successors of any node
- **d**, the depth of the shallowest goal node.
- **m**, the maximum length of any path in the state space

Time and Space is measured in

- Time - number of nodes generated during the search
- Space -maximum number of nodes stored in memory

Measuring problem-solving performance

For effectiveness of a search algorithm

- we can just consider the total cost
- **The total cost** = path cost (g) of the solution found **+** search cost
  - search cost = time necessary to find the solution

Tradeoff:

- (long time, optimal solution with least g)
- vs. (shorter time, solution with slightly larger path cost g)

Unit I

Introduction: (Chapter 1 of Text Book 1)

- What is AI?
- Foundation of Artificial Intelligence
- History of Artificial Intelligence

Intelligent Agents: (Chapter 2 of Text Book 1)

- Agents and Environments
- Rationality
- The Nature of Environments
- The Structure of Agents

Problem-solving by search: (Chapter 3 of Text Book 1)

- Problem-Solving Agents
- Example Problems
- Searching for Solution
- Uniformed Search Strategies
- Informed Search Strategies
- Heuristic Functions

Search strategies

Uninformed search (Blind search)

- no information about the number of steps or the path cost from the current state to the goal
- search the state space **<u>blindly</u>**

Informed search, or heuristic search

- a cleverer strategy that searches toward the goal, based on the information from the current state so far

Uninformed search strategies

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

- Comparing uninformed search strategies

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.
- Breadth-first search implemented using FIFO queue data structure.

Figure 3.12 Breadth-first search on a simple binary tree. At each stage, the node to be expanded next is indicated by a marker.

Example:Breadth-first search

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

Breadth-first search

```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  frontier ← a FIFO queue with node as the only element
  explored ← an empty set
  loop do
     if EMPTY?(frontier) then return failure
     node ← POP(frontier)  /* chooses the shallowest node in frontier */
     add node.STATE to explored
     for each action in problem.ACTIONS(node.STATE) do
        child ← CHILD-NODE(problem, node, action)
        if child.STATE is not in explored or frontier then
           if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
           frontier ← INSERT(child, frontier)
```

Properties of breadth-first search

- 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.
- Time complexity: 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.

$$b + b^2 + b^3 + \cdots + b^d = O(b^d)$$

- Space complexity: Memory size to store every level of the tree is $O(b^d)$

Breadth-first search

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.

Disadvantage

- 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.

Uniform cost search

Breadth-first finds the shallowest goal state

- but not necessarily be the least-cost solution
- work only if all step costs are equal

Uniform cost search

- modifies breadth-first strategy by <u>always expanding the lowest-cost node</u>
- The lowest-cost node is measured by the path cost *g(n)*

RAMAIAH
Institute of Technology

Uniform cost search

- 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.
- A uniform-cost search algorithm is implemented by the priority queue.

Example: Uniform cost search

RAMAIAH
Institute of Technology

Uniform cost search

```
function UNIFORM-COST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  frontier ← a priority queue ordered by PATH-COST, with node as the only element
  explored ← an empty set
  loop do
     if EMPTY?(frontier) then return failure
     node ← POP(frontier)  /* chooses the lowest-cost node in frontier */
     if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
     add node.STATE to explored
     for each action in problem.ACTIONS(node.STATE) do
        child ← CHILD-NODE(problem, node, action)
        if child.STATE is not in explored or frontier then
           frontier ← INSERT(child, frontier)
        else if child.STATE is in frontier with higher PATH-COST then
           replace that frontier node with child
```

Uniform cost search

- Completeness: Is complete, as if there is a solution, UCS will find it.
- Optimality : Is optimal as it only selects a path with the lowest path cost.
- Time complexity:

  Let C* is Cost of the optimal solution,

  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.

  Worst-case time complexity of Uniform-cost search is $O(b^{1 + [C^*/\varepsilon]})$.
- Space complexity: Worst-case space complexity of Uniform-cost search is $O(b^{1 + [C^*/\varepsilon]})$.

Uniform cost search

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.

Depth-first search

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

Example:Depth-first search

RAMAIAH
Institute of Technology

Depth-first search

- Completeness: Is complete within finite state space as it will expand every node within a limited search tree.
- Optimal: Is non-optimal, as it may generate a large number of steps or high cost to reach to the goal node.
- Time Complexity: Time complexity of DFS will be equivalent to the node traversed by the algorithm. It is given by: $O(b^m)$

  Where, m= maximum depth of any node and this can be much larger than d
- 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).

Depth-first search

Advantage:

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

Disadvantage:

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

Depth-Limited Search Algorithm

- A depth-limited search algorithm is similar to DFS 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.

Example:Depth-Limited Search Algorithm

Depth-Limited Search Algorithm

```
function DEPTH-LIMITED-SEARCH(problem, limit) returns a solution, or failure/cutoff
  return RECURSIVE-DLS(MAKE-NODE(problem.INITIAL-STATE), problem, limit)

function RECURSIVE-DLS(node, problem, limit) returns a solution, or failure/cutoff
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  else if limit = 0 then return cutoff
  else
      cutoff_occurred? ← false
      for each action in problem.ACTIONS(node.STATE) do
          child ← CHILD-NODE(problem, node, action)
          result ← RECURSIVE-DLS(child, problem, limit − 1)
          if result = cutoff then cutoff_occurred? ← true
          else if result ≠ failure then return result
      if cutoff_occurred? then return cutoff else return failure
```

Properties of Depth-limited search

Completeness: Is complete if the solution is above the depth-limit.

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

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

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

Depth-limited search

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.

Iterative deepening depth-first search

- It 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 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.

Example: Iterative deepening depth-first search

1'st Iteration--⟶ A d=0

2'nd Iteration----> A -> B -> C d=1

3'rd Iteration------>A, B, D, E, C, F, G d=2

Iterative deepening depth-first search

```
function ITERATIVE-DEEPENING-SEARCH(problem) returns a solution, or failure
    for depth = 0 to ∞ do
        result ← DEPTH-LIMITED-SEARCH(problem, depth)
        if result ≠ cutoff then return result
```

Iterative deepening search (Analysis)

Advantages:

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

Disadvantages:

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

Properties of iterative deepening search

- Completeness: Is complete is if the branching factor is finite.
- Optimal: Is optimal if path cost is a non- decreasing function of the depth of the node.
- 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 is $O(bd)$.

Bidirectional search

- 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 divides 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.

Example Bidirectional search

Bidirectional search

A schematic view of a bidirectional search that is about to succeed when a branch from the start node meets a branch from the goal node.

Bidirectional search

Completeness: Is complete if we use BFS in both searches.

Optimal: Is Optimal.

Time Complexity: Time complexity of bidirectional search using BFS is $O(b^{d/2})$.

Space Complexity: Space complexity of bidirectional search is $O(b^{d/2})$.

Bidirectional search

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.

RAMAIAH
Institute of Technology

Comparing search strategies

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes^a	Yes^{a,b}	No	No	Yes^a	Yes^{a,d}
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes^c	Yes	No	No	Yes^c	Yes^{c,d}

Figure 3.21 Evaluation of tree-search strategies. b is the branching factor; d is the depth of the shallowest solution; m is the maximum depth of the search tree; l is the depth limit. Superscript caveats are as follows: ^{a} complete if b is finite; ^{b} complete if step costs $\geq \epsilon$ for positive ϵ; ^{c} optimal if step costs are all identical; ^{d} if both directions use breadth-first search.

Informed search strategies

The informed search algorithm is more useful for large search space.

Informed search algorithm uses the idea of heuristic, so it is also called Heuristic search.

Informed search strategies

- **Heuristics function:** Heuristic is a function which is used in Informed Search, and it finds the most promising path.
- It takes the current state of the agent as its input and produces the estimation of how close agent is from the goal.
- It is represented by h(n), and it calculates the cost of an optimal path between the pair of states.
- The value of the heuristic function is always positive.

Informed search strategies

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

Greedy best-first search

- It is the combination of DFS and BFS. It uses the heuristic function
- With the help of best-first search, at each step, we can choose the most promising node, we expand the node which is closest to the goal node and the closest cost is estimated by heuristic function, i.e.
- The greedy best first algorithm is implemented by the priority queue.
- Thus, it evaluates nodes by using just the heuristic function

$f(n) = h(n)$

$h(n)$= estimated cost from node n to the goal.

we use the straightline distance heuristic, which we will call h^{SLD}

Example :Greedy best-first search

Final solution path will be **S-⟶ B--⟶F-⟶ G = 2+1+3 =8**

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

Properties of Greedy best-first search

- Time Complexity: The worst case time complexity of Greedy best first search is $O(b^m)$.
- Space Complexity: The worst case space complexity of Greedy best first search is $O(b^m)$. Where, m is the maximum depth of the search space.
- Complete: Greedy best-first search is also incomplete, even if the given state space is finite.
- Optimal: Greedy best first search algorithm is not optimal.

Greedy best-first search

Advantages

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

Disadvantages

- It can behave as an unguided depth-first search in the worst case scenario.
- It can get stuck in a loop as DFS.
- This algorithm is not optimal.

A* search

- A* search is the most commonly known form of best-first search.
- It uses heuristic function h(n), and cost to reach the node n from the start state g(n).
- A* search algorithm finds the shortest path through the search space using the heuristic function.
- In A* search algorithm, we use search heuristic as well as the cost to reach the node.

Examples: A*

Heuristic Table

A 24	F 145	K 108	P 65
B 28	G 123	L 51	Q 11
C 70	H 95	N 58	
D 95	I 77	O 10	
E 118	J 57	M 0	

Properties of A* search

Complete: A* algorithm is complete as long as Branching factor is finite. Cost at every action is fixed.

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

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

Time Complexity: O(b^d), where b is the branching factor.

Space Complexity: O(b^d)

A* search

Advantages

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

Disadvantages

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

Heuristic Functions

- The 8-puzzle was one of the earliest heuristic search problems
- If we want to find the shortest solutions , we need a heuristic function that never overestimates the number of steps to the goal.

2 commonly used methods

- h1 = the number of misplaced tiles.
- h1 is an admissible heuristic because it is clear that any tile that is out of place must be moved at least once
- h2 = the sum of the distances of the tiles from their goal positions.
- Called also the city block distance or Manhattan distance.
- h2 is also admissible because all any move can do is move one tile one step closer to the goal.

Heuristic Functions

The 8-puzzle was one of the earliest heuristic search problems.

A typical instance of the 8-puzzle. The solution is 26 steps long.

Heuristic Functions

The effect of heuristic accuracy on performance

- Branching factor $b*$
- Total number of nodes generated by a particular problem is N
- Solution depth is d
- $b*$ is the branching factor that a uniform tree of depth d would have to have in order to contain N + 1 nodes.
- Thus, $N + 1 = 1 + b^* + (b^*)^2 + \cdots + (b^*)^d$.

g=0, h=3, f=g+h=3
g=1, h=4, f=5
g=1, h=2, f=3
g=1, h=4, f=5
1
g=2, h=1, f=3
g=2, h=3, f=5
g=2, h=3, f=5
2
g=3, h=2, f=5
g=3, h=0, f=3
3

RAMAIAH
Institute of Technology

THANK YOU