MODULE 1

# INTRODUCTION

* Artificial Intelligence is concerned with the design of intelligence in an artificial device. The term was coined by John McCarthy in 1956.
* Intelligence is the ability to acquire, understand and apply the knowledge to achieve goals in the world.
* AI is the study of the mental faculties through the use of computational models
* AI is the study of intellectual/mental processes as computational processes.
* AI program will demonstrate a high level of intelligence to a degree that equals or exceeds the intelligence required of a human in performing some tasks.
* AI is unique, sharing borders with Mathematics, Computer Science, Philosophy, Psychology, Biology, Cognitive Science and many others.
* Although there is no clear definition of AI or even Intelligence, it can be described as an attempt to build machines that like humans can think and act, able to learn and use knowledge to solve problems on their own.

## History of AI:

Important research that laid the groundwork for AI:

* In 1931, [Goedel](http://www.idsia.ch/~juergen/goedel.html) layed the foundation of Theoretical Computer Science**1920-30s**:

 He published the first universal formal language and showed that math itself is either flawed or allows for unprovable but true statements.

* In 1936, Turing reformulated Goedel’s result and church’s extension thereof.
* In 1956, John McCarthy coined the term "Artificial Intelligence" as the topic of the [**Dartmouth**](http://www-formal.stanford.edu/jmc/history/dartmouth.html)[**Conference**,](http://www-formal.stanford.edu/jmc/history/dartmouth.html) the first conference devoted to the subject.
* In 1957, The [**General Problem Solver (GPS)**](http://aitopics.org/publication/gps-program-simulates-human-thought) demonstrated by Newell, Shaw & Simon
* In 1958, John McCarthy (MIT) invented the Lisp language.
* In 1959, Arthur Samuel (IBM) wrote the first game-playing program, for checkers, to achieve sufficient skill to challenge a world champion.
* In 1963, Ivan Sutherland's MIT dissertation on Sketchpad introduced the idea of interactive graphics into computing.
* In 1966, Ross Quillian (PhD dissertation, Carnegie Inst. of Technology; now CMU)

demonstrated semantic nets

* In 1967, [Dendral program](http://aitopics.org/publication/heuristic-dendral-program-generating-explanatory-hypotheses-organic-chemistry) (Edward Feigenbaum, Joshua Lederberg, Bruce Buchanan, Georgia Sutherland at Stanford) demonstrated to interpret mass spectra on organic chemical compounds. First successful knowledge-based program for scientific reasoning.
* In 1967, Doug Engelbart invented the mouse at SRI
* In 1968, Marvin Minsky & Seymour Papert publish Perceptrons, demonstrating limits of simple neural nets.
* In 1972, Prolog developed by Alain Colmerauer.
* In Mid 80’s, Neural Networks become widely used with the Backpropagation algorithm (first described by Werbos in 1974).
* 1990, Major advances in all areas of AI, with significant demonstrations in machine learning, intelligent tutoring, case-based reasoning, multi-agent planning, scheduling, uncertain reasoning, data mining, natural language understanding and translation, vision, virtual reality, games, and other topics.
* In 1997, Deep Blue beats the World Chess Champion Kasparov
* In 2002[,iRobot,](http://www.irobot.com/) founded by researchers at the MIT Artificial Intelligence Lab, introduced **Roomba**, a vacuum cleaning robot. By 2006, two million had been sold.

## Foundations of Artificial Intelligence:

##### Philosophy

e.g., foundational issues (can a machine think?), issues of knowledge and believe, mutual knowledge

* 1. **Game Playing** Psychology and Cognitive Science

e.g., problem solving skills

* 1. **Neuro-Science**

e.g., brain architecture

* 1. **Computer Science and Engineering**

e.g., complexity theory, algorithms, logic and inference, programming languages, and system building.

* 1. **Mathematics and Physics**

e.g., statistical modeling, continuous mathematics,

6. **Statistical Physics, and Complex** Systems. Sub Areas of AI:

1. **Deep Blue** Chess program beat world champion Gary Kasparov
2. **Speech Recognition**

PEGASUS spoken language interface to American Airlines' EAASY SABRE reseration system, which allows users to obtain flight information and make reservations over the telephone. The 1990s has seen significant advances in speech recognition so that limited systems are now successful.

1. **Computer Vision**

Face recognition programs in use by banks, government, etc. The ALVINN system from CMU autonomously drove a van from Washington, D.C. to San Diego (all but 52 of 2,849 miles), averaging 63 mph day and night, and in all weather conditions. Handwriting recognition, electronics and manufacturing inspection, photo interpretation, baggage inspection, reverse engineering to automatically construct a 3D geometric model.

1. **Expert Systems**

Application-specific systems that rely on obtaining the knowledge of human experts in an area and programming that knowledge into a system.

* 1. **Diagnostic Systems:** MYCIN system for diagnosing bacterial infections of the blood and suggesting treatments. Intellipath pathology diagnosis system (AMA approved). Pathfinder medical diagnosis system, which suggests tests and makes diagnoses. Whirlpool customer assistance center.
	2. System Configuration

DEC's XCON system for custom hardware configuration. Radiotherapy treatment planning.

* 1. Financial Decision Making

Credit card companies, mortgage companies, banks, and the U.S. government employ AI systems to detect fraud and expedite financial transactions. For example, AMEX credit check.

* 1. Classification Systems

Put information into one of a fixed set of categories using several sources of information. E.g., financial decision making systems. NASA developed a system for classifying very faint areas in astronomical images into either stars or galaxies with very high accuracy by learning from human experts' classifications.

1. Mathematical Theorem Proving

Use inference methods to prove new theorems.

1. Natural Language Understanding

[AltaVista's translation](http://babelfish.altavista.digital.com/cgi-bin/translate) of web pages. Translation of Catepillar Truck manuals into 20 languages.

1. Scheduling and Planning

Automatic scheduling for manufacturing. DARPA's DART system used in Desert Storm and Desert Shield operations to plan logistics of people and supplies. American Airlines rerouting contingency planner. European space agency planning and scheduling of spacecraft assembly, integration and verification.

1. Artificial Neural Networks:
2. Machine Learning

**Application of AI**

AI algorithms have attracted close attention of researchers and have also been applied successfully to solve problems in engineering. Nevertheless, for large and complex problems, AI algorithms consume considerable computation time due to stochastic feature of the search approaches

1. Business; financial strategies
2. Engineering: check design, offer suggestions to create new product, expert systems for all engineering problems
3. Manufacturing: assembly, inspection and maintenance
4. Medicine: monitoring, diagnosing
5. Education: in teaching
6. Fraud detection
7. Object identification
8. Information retrieval
9. Space shuttle scheduling

## Building AI Systems:

1. **Perception**

Intelligent biological systems are physically embodied in the world and experience the world through their sensors (senses). For an autonomous vehicle, input might be images from a camera and range information from a rangefinder. For a medical diagnosis system, perception is the set of symptoms and test results that have been obtained and input to the system manually.

1. **Reasoning**

Inference, decision-making, classification from what is sensed and what the internal "model" is of the world. Might be a neural network, logical deduction system, Hidden Markov Model induction, heuristic searching a problem space, Bayes Network inference, genetic algorithms, etc.

Includes areas of knowledge representation, problem solving, decision theory, planning, game theory, machine learning, uncertainty reasoning, etc.

1. **Action**

Biological systems interact within their environment by actuation, speech, etc. All behavior is centered around actions in the world. Examples include controlling the steering of a Mars rover or autonomous vehicle, or suggesting tests and making diagnoses for a medical diagnosis system. Includes areas of robot actuation, natural language generation, and speech synthesis.

**The definitions of AI:**

|  |  |
| --- | --- |
| a) "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) | b) "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) |
| c) "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, 1 99 1 ) | d) "A field of study that seeks to explain and emulate intelligent behavior in terms of computational processes" (Schalkoff, 1 990)"The branch of computer science that is concerned with the automation of intelligent behavior" (Luger and Stubblefield, 1993) |

The definitions on the top, **(a)** and **(b)** are concerned with **reasoning**, whereas those on the bottom**, (c)** and **(d)** address **behavior.**The definitions on the left, **(a)** and **(c)** measure success in terms of human performance, and those on the right, **(b)** and **(d)** measure the ideal concept of intelligence called rationality

**Intelligent Systems:**

In order to design intelligent systems, it is important to categorize them into four categories (Luger and Stubberfield 1993), (Russell and Norvig, 2003)

1. Systems that think like humans
2. Systems that think rationally
3. Systems that behave like humans
4. Systems that behave rationally

|  |  |  |
| --- | --- | --- |
|  | **Human- Like** | **Rationally** |
| **Think:** | **Cognitive Science Approach*****“Machines that think like humans”*** | **Laws of thought Approach*****“ Machines that think Rationally”*** |
| **Act:** | **Turing Test Approach*****“Machines that behave like humans”*** | **Rational Agent Approach*****“Machines that behave Rationally”*** |

**Scientific Goal:** To determine which ideas about knowledge representation, learning, rule systems search, and so on, explain various sorts of real intelligence.

**Engineering Goal:** To solve real world problems using AI techniques such as Knowledge representation, learning, rule systems, search, and so on.

Traditionally, computer scientists and engineers have been more interested in the engineering goal, while psychologists, philosophers and cognitive scientists have been more interested in the scientific goal.

**Cognitive Science: Think Human-Like**

* 1. Requires a model for human cognition. Precise enough models allow simulation by computers.
	2. Focus is not just on behavior and I/O, but looks like reasoning process.
	3. Goal is not just to produce human-like behavior but to produce a sequence of steps of the reasoning process, similar to the steps followed by a human in solving the same task.

**Laws of thought: Think Rationally**

1. The study of mental faculties through the use of computational models; that it is, the study of computations that make it possible to perceive reason and act.
2. Focus is on inference mechanisms that are probably correct and guarantee an optimal solution.
3. Goal is to formalize the reasoning process as a system of logical rules and procedures of inference.
4. Develop systems of representation to allow inferences to be like “*Socrates is a man. All men are mortal. Therefore, Socrates is mortal”*

**Turing Test: Act Human-Like**

1. The art of creating machines that perform functions requiring intelligence when performed by people; that it is the study of, how to make computers do things which, at the moment, people do better.
2. Focus is on action, and not intelligent behavior centered around the representation of the world
3. Example: Turing Test
* 3 rooms contain: a person, a computer and an interrogator.
* The interrogator can communicate with the other 2 by teletype (to avoid the machine imitate the appearance of voice of the person)
* The interrogator tries to determine which the person is and which the machine is.
* The machine tries to fool the interrogator to believe that it is the human, and the person also tries to convince the interrogator that it is the human.
* If the machine succeeds in fooling the interrogator, then conclude that the machine is intelligent.

**Rational agent: Act Rationally**

1. Tries to explain and emulate intelligent behavior in terms of computational process; that it is concerned with the automation of the intelligence.
2. Focus is on systems that act sufficiently if not optimally in all situations.
3. Goal is to develop systems that are rational and sufficient

**The difference between strong AI and weak AI:**

**Strong AI** makes the bold claim that computers can be made to think on a level (at least) equal to humans.

**Weak AI** simply states that some "thinking-like" features can be added to computers to make them more useful tools... and this has already started to happen (witness expert systems, drive-by-wire cars and speech recognition software).

**AI Problems:**

AI problems (speech recognition, NLP, vision, automatic programming, knowledge representation, etc.) can be paired with techniques (NN, search, Bayesian nets, production systems, etc.).AI problems can be classified in two types:

1. Common-place tasks (Mundane Tasks)
2. Expert tasks

**Common-Place Tasks:**

1. *Recognizing* people, objects.
2. Communicating (through *natural language*).
3. *Navigating* around obstacles on the streets.

These tasks are done matter of factly and routinely by people and some other animals.

Expert tasks:

* 1. Medical diagnosis.
	2. Mathematical problem solving
	3. Playing games like chess

These tasks cannot be done by all people, and can only be performed by skilled specialists.

Clearly tasks of the first type are easy for humans to perform, and almost all are able to master them. The second range of tasks requires skill development and/or intelligence and only some specialists can perform them well. However, when we look at what computer systems have been able to achieve to date, we see that their achievements include performing sophisticated tasks like medical diagnosis, performing symbolic integration, proving theorems and playing chess.

**Intelligent Agent’s:**

***Agents and environments:***



**Fig 2.1:** Agents and Environments

**Agent:**

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

* + - * A *human agent* has eyes, ears, and other organs for sensors and hands, legs, mouth, and other body parts for actuators.
			* A *robotic agent* might have cameras and infrared range finders for sensors and various motors for actuators.
			* A *software agent* receives keystrokes, file contents, and network packets as sensory inputs and acts on the environment by displaying on the screen, writing files, and sending network packets.

**Percept:**

We use the term percept to refer to the agent's perceptual inputs at any given instant.

 **Percept Sequence:**

An agent's percept sequence is the complete history of everything the agent has ever perceived.

**Agent function:**

Mathematically speaking, we say that an agent's behavior is described by the agent function that maps any **given percept sequence to an action**.

**Agent program**

Internally, the agent function for an artificial agent will be implemented by an agent program. It is important to keep these two ideas distinct. The agent function is an abstract mathematical description; the agent program is a concrete implementation, running on the agent architecture.

To illustrate these ideas, we will use a very simple example-the vacuum-cleaner world shown in **Fig 2.1.5**. This particular world has just two locations: squares A and B. The vacuum agent perceives which square it is in and whether there is dirt in the square. It can choose to move left, move right, suck up the dirt, or do nothing. One very simple agent function is the following: if the current square is dirty, then suck, otherwise move to the other square. A partial tabulation of this agent function is shown in **Fig 2.1.6**.



 **Fig 2.1.5:** A vacuum-cleaner world with just two locations.

***Agent function***

|  |  |
| --- | --- |
| **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 |
| … |  |

**Fig 2.1.6**: Partial tabulation of a simple agent function for the example: vacuum-cleaner

world shown in the **Fig 2.1.5**

**Fig 2.1.6(i)**: The REFLEX-VACCUM-AGENT program is invoked for each new percept (location, status) and returns **an** action each time

**Strategies of Solving Tic-Tac-Toe**

**Game Playing Tic-Tac-Toe Game Playing:**

Function REFLEX-VACCUM-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

Tic-Tac-Toe is a simple and yet an interesting board game. Researchers have used various approaches to study the Tic-Tac-Toe game. For example, Fok and Ong and Grim et al. have used artificial neural network-based strategies to play it. Citrenbaum and Yakowitz discuss games like Go-Moku, Hex and Bridge-It which share some similarities with Tic-Tac-Toe



**A Formal Definition of the Game:**

The board used to play the Tic-Tac-Toe game consists of 9 cells laid out in the form of a 3x3 matrix (Fig. 1). The game is played by 2 players and either of them can start. Each of the two players is assigned a unique symbol (generally 0 and X). Each player alternately gets a turn to make a move. Making a move is compulsory and cannot be deferred. In each move a player places the symbol assigned to him/her in a hitherto blank cell.

Let a track be defined as any row, column or diagonal on the board. Since the board is a square matrix with 9 cells, all rows, columns and diagonals have exactly 3 cells. It can be easily observed that there are 3 rows, 3 columns and 2 diagonals, and hence a total of 8 tracks on the board (Fig. 1). The goal of the game is to fill all the three cells of any track on the board with the symbol assigned to one before the opponent does the same with the symbol assigned to him/her. At any point of the game, if there exists a track whose all three cells have been marked by the same symbol, then the player to whom that symbol has been assigned wins and the game terminates. If there exist no track whose cells have been marked by the same symbol when there is no more blank cell on the board then the game is drawn.

Let the priority of a cell be defined as the number of tracks passing through it. The priorities of the nine cells on the board according to this definition are tabulated in Table 1. Alternatively, let the priority of a track be defined as the sum of the priorities of its three cells. The priorities of the eight tracks on the board according to this definition are tabulated in Table 2. The prioritization of the cells and the tracks lays the foundation of the heuristics to be used in this study. These heuristics are somewhat similar to those proposed by Rich and Knight.



**Strategy 1:**

**Algorithm:**

1. View the vector as a ternary number. Convert it to a decimal number.
2. Use the computed number as an index into Move-Table and access the vector stored there.
3. Set the new board to that vector.

**Procedure:**

1. Elements of vector:

0: Empty

1: X

2: O

→ the vector is a ternary number

1. Store inside the program a move-table (lookup table):
	1. Elements in the table: 19683 (39)
	2. Element = A vector which describes the most suitable move from the



##### Comments:

1. A lot of space to store the Move-Table.
2. A lot of work to specify all the entries in the Move-Table.
3. Difficult to extend

##### Explanation of Strategy 2 of solving Tic-tac-toe problem Strategy 2:

Data Structure:

1. Use vector, called board, as Solution 1
2. However, elements of the vector: 2: Empty

3: X

5: O

1. Turn of move: indexed by integer 1,2,3, etc

##### Function Library:

1. Make2:
	1. Return a location on a game-board. IF (board[5] = 2)

RETURN 5; //the center cell. ELSE

RETURN any cell that is not at the board’s corner;

// (cell: 2,4,6,8)

* 1. Let P represent for X or O
	2. can\_win(P) :

P has filled already at least two cells on a straight line (horizontal, vertical, or diagonal)

* 1. cannot win(P) = NOT(can win(P))
1. Posswin(P):

IF (cannot win(P)) RETURN 0;

ELSE

RETURN index to the empty cell on the line of can\_win(P)

Let odd numbers are turns of X Let even numbers are turns of O

1. Go(n): make a move

##### Algorithm:

1. **Turn = 1**: (X moves)

Go(1) //make a move at the left-top cell

1. **Turn = 2**: (O moves)

IF board[5] is empty THEN

Go(5)

ELSE Go(1)

1. **Turn = 3:** (X moves)

IF board[9] is empty THEN Go(9)

ELSE Go(3).

1. **Turn = 4**: (O moves)

IF Posswin (X) <> 0 THEN Go (Posswin (X))

//Prevent the opponent to win ELSE Go (Make2)

1. **Turn = 5:** (X moves)

IF Posswin(X) <> 0 THEN Go(Posswin(X))

//Win for X.

ELSE IF Posswin(O) <> THEN

Go(Posswin(O))

//Prevent the opponent to win ELSE IF board[7] is empty THEN Go(7)

ELSE Go(3).

##### Comments:

1. Not efficient in time, as it has to check several conditions before making each
2. Easier to understand the program’s strategy.
3. Hard to generalize

##### Introduction to Problem Solving, General problem solving

Problem solving is a process of generating solutions from observed data.

−a problem is characterized by a set of goals,

−a set of objects, and

−a set of operations.

These could be ill-defined and may evolve during problem solving.

##### Searching Solutions:

 To build a system to solve a problem:

* 1. Define the problem precisely
	2. Analyze the problem
	3. Isolate and represent the task knowledge that is necessary to solve the problem

Choose the best problem-solving techniques and apply it to the particular problem.

##### Defining the problem as State Space Search:

The state space representation forms the basis of most of the AI methods.

* Formulate a problem as a **state space search** by showing the legal problem states, the legal operators, and the initial and goal states.
* A **state** is defined by the specification of the values of all attributes of interest in the world
* An **operator** changes one state into the other; it has a precondition which is the value of certain attributes prior to the application of the operator, and a set of effects, which are the attributes altered by the operator
* The **initial state** is where you start
* The **goal state** is the partial description of the solution

##### Formal Description of the problem:

1. Define a state space that contains all the possible configurations of the relevant objects.
2. Specify one or more states within that space that describe possible situations from which the problem-solving process may start **(initial state)**
3. Specify one or more states that would be acceptable as solutions to the problem**. (goal states)**

Specify a set of rules that describe the actions **(operations)** available

##### State-Space Problem Formulation:

Example: A problem is defined by four items:

1. **initial state** e.g., "at Arad―
2. **actions or successor function:** *S(x)* = set of action–state pairs e.g., *S(Arad) =* {*<Arad*  *Zerind, Zerind>, …}*

##### goal test (or set of goal states)

e.g., *x* = "at Bucharest‖, *Checkmate(x)*

##### path cost (additive)

e.g., sum of distances, number of actions executed, etc.

*c(x,a,y)* is the step cost, assumed to be ≥ 0

A solution is a sequence of actions leading from the initial state to a goal state



##### Example: 8-queens problem



* 1. **Initial State**: Any arrangement of 0 to 8 queens on board.
	2. **Operators**: add a queen to any square.
	3. **Goal Test**: 8 queens on board, none attacked.
	4. **Path cost**: not applicable or Zero (because only the final state counts, search cost might be of interest).

State Spaces versus Search Trees:

* State Space
	+ Set of valid states for a problem
	+ Linked by operators
	+ e.g., 20 valid states (cities) in the Romanian travel problem
* Search Tree
* Root node = initial state
* Child nodes = states that can be visited from parent
* Note that the depth of the tree can be infinite
	+ E.g., via repeated states
* Partial search tree
	+ Portion of tree that has been expanded so far
* Fringe
	+ Leaves of partial search tree, candidates for expansion Search trees = data structure to search state-space

**Properties of Search Algorithms**

Which search algorithm one should use will generally depend on the problem domain. There are four important factors to consider:

1. ***Completeness*** – Is a solution guaranteed to be found if at least one solution exists?
2. ***Optimality*** – Is the solution found guaranteed to be the best (or lowest cost) solution if there exists more than one solution?
3. ***Time Complexity*** – The upper bound on the time required to find a solution, as a function of the complexity of the problem.
4. ***Space Complexity*** – The upper bound on the storage space (memory) required at any point during the search, as a function of the complexity of the problem.

**General problem solving, Water-jug problem, 8-puzzle problem General Problem Solver*:***

The General Problem Solver (GPS) was the first useful AI program, written by Simon, Shaw, and Newell in 1959. As the name implies, it was intended to solve nearly any problem***.***

Newell and Simon defined each problem as a space. At one end of the space is the starting point; on the other side is the goal. The problem-solving procedure itself is conceived as a set of operations to cross that space, to get from the starting point to the goal state, one step at a time.

The General Problem Solver, the program tests various actions (which Newell and Simon called operators) to see which will take it closer to the goal state. An operator is any activity that changes the state of the system. The General Problem Solver always chooses the operation that appears to bring it closer to its goal.

**Example: Water Jug Problem**

**Consider the following problem:**

A Water Jug Problem: You are given two jugs, a 4-gallon one and a 3-gallon one, a pump which has unlimited water which you can use to fill the jug, and the ground on which water may be poured. Neither jug has any measuring markings on it. How can you get exactly 2 gallons of water in the 4-gallon jug?

**State Representation and Initial State :**

We will represent a state of the problem as a tuple (x, y) where x represents the amount of water in the 4-gallon jug and y represents the amount of water in the 3-gallon jug. Note 0 ≤x≤ 4, and 0 ≤y ≤3. Our initial state: (0, 0)

Goal Predicate - state = (2, y) where 0≤ y≤ 3.

Operators -we must defi ne a set of operators that will take us from one state to another:

|  |  |  |  |
| --- | --- | --- | --- |
| 1. | Fill 4-gal jug | (x,y) | *→* (4,y) |
|  |  | x *<* 4 |  |
| 2. | Fill 3-gal jug | (x,y) | *→* (x,3) |
|  |  | y *<* 3 |  |
| 3. | Empty 4-gal jug on ground | (x,y) | *→* (0,y) |
|  |  | x *>* 0 |  |
| 4. | Empty 3-gal jug on ground | (x,y) | *→* (x,0) |
|  |  | y *>* 0 |  |
| 5. | Pour water from 3-gal jug | (x,y) | *→!* (4, y - (4 - x)) |
|  | to ll 4-gal jug | 0 *<* x+y 4 and y *>* 0 |
| 6. | Pour water from 4-gal jug | (x,y) | *→* (x - (3-y), 3) |
|  | to ll 3-gal-jug | 0 *<* x+y 3 and x *>* 0 |
| 7. | Pour all of water from 3-gal jug | (x,y) | *→* (x+y, 0) |
|  | into 4-gal jug | 0 *<* x+y 4 and y 0 |
| 8. | Pour all of water from 4-gal jug | (x,y) | *→* (0, x+y) |
|  | into 3-gal jug | 0 *<* x+y 3 and x 0 |

Through Graph Search, the following solution is found:

|  |  |
| --- | --- |
| Gals in 4-gal jug Gals in 3-gal jug | Rule Applied |
| 0 | 0 |  |  |
|  |  | 1. | Fill 4 |
| 4 | 0 |  |  |
| 6. Pour 4 into 3 to ll |
| 1 | 3 |  |  |
| 4. Empty 3 |
| 1 | 0 |  |  |
| 8. Pour all of 4 into 3 |
| 0 | 1 |  |  |
|  |  | 1. | Fill 4 |
| 4 | 1 |  |  |
| 6. Pour into 3 |
| 2 | 3 |  |  |

##### Second Solution:

#####

Control strategies

Control Strategies means how to decide which rule to apply next during the process of searching for a solution to a problem.

Requirement for a good Control Strategy

* 1. It should cause motion

In water jug problem, if we apply a simple control strategy of starting each time from the top of rule list and choose the first applicable one, then we will never move towards solution.

* 1. It should explore the solution space in a systematic manner

If we choose another control strategy, let us say, choose a rule randomly from the applicable rules then definitely it causes motion and eventually will lead to a solution. But one may arrive to same state several times. This is because control strategy is not systematic.

**Systematic Control Strategies (Blind searches):**

**Breadth First Search:**

Let us discuss these strategies using water jug problem. These may be applied to any search problem. Construct a tree with the initial state as its root.

Generate all the offspring of the root by applying each of the applicable rules to the initial state**.**



Now for each leaf node, generate all its successors by applying all the rules that are appropriate.

 **8 Puzzle Problem.**

The 8 puzzle consists of eight numbered, movable tiles set in a 3x3 frame. One cell of the frame is always empty thus making it possible to move an adjacent numbered tile into the empty cell. Such a puzzle is illustrated in following diagram.

The program is to change the initial configuration into the goal configuration. A solution to the problem is an appropriate sequence of moves, such as “move tiles 5 to the right, move tile 7 to the left, move tile 6 to the down, etc”.

 **Solution:**

To solve a problem using a production system, we must specify the global database the rules, and the control strategy. For the 8 puzzle problem that correspond to these three components. These elements are the problem states, moves and goal. In this problem each tile configuration is a state. The set of all configuration in the space of problem states or the problem space, there are only 3, 62,880 different configurations o the 8 tiles and blank space. Once the problem states have been conceptually identified, we must construct a computer representation, or description of them . this description is then used as the database of a production system. For the 8-puzzle, a straight forward description is a 3X3 array of matrix of numbers. The initial global database is this description of the initial problem state. Virtually any kind of data structure can be used to describe states.

A move transforms one problem state into another state. The 8-puzzle is conveniently interpreted as having the following for moves. Move empty space (blank) to the left, move blank up, move blank to the right and move blank down,. These moves are modeled by production rules that operate on the state descriptions in the appropriate manner.

The rules each have preconditions that must be satisfied by a state description in order for them to be applicable to that state description. Thus the precondition for the rule associated with “move blank up” is derived from the requirement that the blank space must not already be in the top row.

The problem goal condition forms the basis for the termination condition of the production system. The control strategy repeatedly applies rules to state descriptions until a description of a goal state is produced. It also keeps track of rules that have been applied so that it can compose them into sequence representing the problem solution. A solution to the 8-puzzle problem is given in the following figure.

Example:- Depth – First – Search traversal and Breadth - First - Search traversal

for 8 – puzzle problem is shown in following diagrams.





**Exhaustive Searches, BFS and DFS**

Search is the systematic examination of states to find path from the start/root state to the goal state. Many traditional search algorithms are used in AI applications. For complex problems, the traditional algorithms are unable to find the solution within some practical time and space limits. Consequently, many special techniques are developed; using heuristic functions. The algorithms that use heuristic functions are called heuristic algorithms. Heuristic algorithms are not really intelligent; they appear to be intelligent because they achieve better performance.Heuristic algorithms are more efficient because they take advantage of feedback from the data to direct the search path.

**Uninformed search**

Also called blind, exhaustive or brute-force search, uses no information about the problem to guide the search and therefore may not be very efficient.

**Informed Search:**

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

**Uninformed Search Methods**:

**Breadth- First -Search:**

Consider the state space of a problem that takes the form of a tree. Now, if we search the goal along each breadth of the tree, starting from the root and continuing up to the largest depth, we call it ***breadth first search*.**

* Algorithm:
1. Create a variable called NODE-LIST and set it to initial state
2. Until a goal state is found or NODE-LIST is empty do
	1. Remove the first element from NODE-LIST and call it E. If NODE-LIST was empty, quit
	2. For each way that each rule can match the state described in E do:
		1. Apply the rule to generate a new state
		2. If the new state is a goal state, quit and return this state
		3. Otherwise, add the new state to the end of NODE-LIST

BFS illustrated:

**Step 1:** Initially fringe contains only one node corresponding to the source state A.



##### Figure 1

**FRINGE: A**

**Step 2:** A is removed from fringe. The node is expanded, and its children B and C are generated. They are placed at the back of fringe.

##### Figure 2

**FRINGE: B C**

**Step 3:** Node B is removed from fringe and is expanded. Its children D, E are generated and put at the back of fringe.

##### Figure 3

**FRINGE: C D E**

**Step 4:** Node C is removed from fringe and is expanded. Its children D and G are added to the back of fringe.



##### Figure 4

**FRINGE: D E D G**

**Step 5**: Node D is removed from fringe. Its children C and F are generated and added to the back of fringe.

##### Figure 5

**FRINGE: E D G C F**

**Step 6**: Node E is removed from fringe. It has no children.



#####  Figure 6

**FRINGE: D G C F**

**Step 7**: D is expanded; B and F are put in OPEN.

##### Figure 7

##### FRINGE: G C F B F

**Step 8**: G is selected for expansion. It is found to be a goal node. So the algorithm returns the path A C G by following the parent pointers of the node corresponding to G. The algorithm terminates.

Breadth first search is:

* + One of the simplest search strategies
	+ Complete. If there is a solution, BFS is guaranteed to find it.
	+ If there are multiple solutions, then a minimal solution will be found
	+ The algorithm is optimal (i.e., admissible) if all operators have the same cost. Otherwise, breadth first search finds a solution with the shortest path length.
	+ **Time complexity** : O(bd )
	+ **Space complexity** : O(bd )
	+ **Optimality** :Yes

b - branching factor(maximum no of successors of any node), d – Depth of the shallowest goal node

***Maximum length of any path (m) in search space***

**Advantages**: Finds the path of minimal length to the goal.

**Disadvantages:**

* Requires the generation and storage of a tree whose size is exponential the depth of the shallowest goal node.
* The breadth first search algorithm cannot be effectively used unless the search space is quite small.

**Depth- First- Search.**

We may sometimes search the goal along the largest depth of the tree, and move up only when further traversal along the depth is not possible. We then attempt to find alternative offspring of the parent of the node (state) last visited. If we visit the nodes of a tree using the above principles to search the goal, the traversal made is called depth first traversal and consequently the search strategy is called *depth first search*.

* Algorithm:
1. Create a variable called NODE-LIST and set it to initial state
2. Until a goal state is found or NODE-LIST is empty do
	1. Remove the first element from NODE-LIST and call it E. If NODE-LIST was empty, quit
	2. For each way that each rule can match the state described in E do:
		1. Apply the rule to generate a new state
		2. If the new state is a goal state, quit and return this state

Otherwise, add the new state in front of NODE-LIST

##### DFS illustrated:



**A State Space Graph Step 1**: Initially fringe contains only the node for A.



##### Figure 1

**FRINGE: A**

**Step 2:** A is removed from fringe. A is expanded and its children B and C are put in front of fringe.



##### Figure 2

**FRINGE: B C**

**Step 3:** Node B is removed from fringe, and its children D and E are pushed in front of fringe.

##### Figure 3

**FRINGE: D E C**

**Step 4:** Node D is removed from fringe. C and F are pushed in front of fringe.

##### Figure 4

**FRINGE: C F E C**

**Step 5:** Node C is removed from fringe. Its child G is pushed in front of fringe.

##### Figure 5

**FRINGE: G F E C**

**Step 6:** Node G is expanded and found to be a goal node.



##### Figure 6

**FRINGE: *G* F E C**

The solution path A-B-D-C-G is returned and the algorithm terminates.

**Depth first searches:**

1. The algorithm takes exponential time.
2. If N is the maximum depth of a node in the search space, in the worst case the algorithm will

d

take time O(b ).

1. The space taken is linear in the depth of the search tree, O(bN).

Note that the time taken by the algorithm is related to the maximum depth of the search tree. If the search tree has infinite depth, the algorithm may not terminate. This can happen if the search space is infinite. It can also happen if the search space contains cycles. The latter case can be handled by checking for cycles in the algorithm. Thus **Depth First Search is not complete.**

**Exhaustive searches- Iterative Deeping DFS**

**Description:**

* It is a search strategy resulting when you combine BFS and DFS, thus combining the advantages of each strategy, taking the completeness and optimality of BFS and the modest memory requirements of DFS.
* IDS works by looking for the best search depth d, thus starting with depth limit 0 and make a BFS and if the search failed it increase the depth limit by 1 and try a BFS again with depth 1 and so on – first d = 0, then 1 then 2 and so on – until a depth d is reached where a goal is found.

Algorithm:

**procedure** IDDFS(root)

**for** depth **from** 0 **to** ∞

found ← DLS(root, depth)

**if** found ≠ null

**return** found

**procedure** DLS(node, depth) **if** depth = 0 **and** node is a goal **return** node

**else if** depth > 0

**foreach** child of node

found ← DLS(child, depth−1)

**if** found ≠ null **return** found **return** null

**Performance Measure:**

* Completeness: IDS is like BFS, is complete when the branching factor b is finite.
* Optimality: IDS is also like BFS optimal when the steps are of the same cost.
* Time Complexity:
* One may find that it is wasteful to generate nodes multiple times, but actually it is not that costly compared to BFS, that is because most of the generated nodes are always in the deepest level reached, consider that we are searching a binary tree and our depth limit reached 4, the nodes generated in last level = 24 = 16, the nodes generated in all nodes before last level = 20 + 21 + 22 + 23= 15
* Imagine this scenario, we are performing IDS and the depth limit reached depth d, now if you remember the way IDS expands nodes, you can see that nodes at depth d are generated once, nodes at depth d-1 are generated 2 times, nodes at depth d-2 are generated 3 times and so on, until you reach depth 1 which is generated d times, we can view the total number of generated nodes in the worst case as:

N(IDS) = (b)d + (d – 1)b2+ (d – 2)b3 + …. + (2)bd-1 + (1)bd = *O*(bd)

* If this search were to be done with BFS, the total number of generated nodes in the worst case will be like:

N(BFS) = b + b2 + b3 + b4 + …. bd + (bd+ 1 – b) = *O*(bd + 1)

* If we consider a realistic numbers, and use b = 10 and d = 5, then number of generated nodes in BFS and IDS will be like
	+ - N(IDS) = 50 + 400 + 3000 + 20000 + 100000 = 123450
		- N(BFS) = 10 + 100 + 1000 + 10000 + 100000 + 999990 = 1111100
		- BFS generates like 9 time nodes to those generated with IDS.
* **Space Complexity:**
	+ IDS is like DFS in its space complexity, taking *O*(bd) of memory.

Weblinks:

* 1. [*https://www.youtube.com/watch?v=7QcoJjSVT38*](https://www.youtube.com/watch?v=7QcoJjSVT38)
	2. <https://mhesham.wordpress.com/tag/iterative-deepening-depth-first-search>

**Conclusion:**

* We can conclude that IDS is a hybrid search strategy between BFS and DFS inheriting their advantages.
* IDS is faster than BFS and DFS.
* It is said that “IDS is the preferred uniformed search method when there is a large search space and the depth of the solution is not known”.