

UNIT I

INTRODUCTION TO AI AND PRODUCTION SYSTEMS

CHAPTER - 1

What is Artificial Intelligence?

1. INTELLIGENCE

- ❖ The capacity to learn and solve problems.
- ❖ In particular,
 - the ability to solve novel problems (i.e solve new problems)
 - the ability to act rationally (i.e act based on reason)
 - the ability to act like humans

1.1 What is involved in intelligence?

- **Ability to interact with the real world**
 - to perceive, understand, and act
 - e.g., speech recognition and understanding and synthesis
 - e.g., image understanding
 - e.g., ability to take actions, have an effect
- **Reasoning and Planning**
 - modeling the external world, given input
 - solving new problems, planning, and making decisions
 - ability to deal with unexpected problems, uncertainties
- **Learning and Adaptation**
 - we are continuously learning and adapting
 - our internal models are always being “updated”
 - e.g., a baby learning to categorize and recognize animals

2. ARTIFICIAL INTELLIGENCE

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

The term AI is defined by each author in own ways which falls into 4 categories

1. The system that think like humans.
2. System that act like humans.
3. Systems that think rationally.
4. Systems that act rationally.

2.1 SOME DEFINITIONS OF AI

- **Building systems that think like humans**

“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, ... such as decision-making, problem solving, learning, ...” -- Bellman, 1978

- **Building systems that act like humans**

“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

- **Building systems that think rationally**

“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

- **Building systems that act rationally**

“A field of study that seeks to explain and emulate intelligent behavior in terms of computational processes” -- Schalkoff, 1990

“The branch of computer science that is concerned with the automation of intelligent behavior” -- Luger and Stubblefield, 1993

2.1.1. Acting Humanly: The Turing Test Approach

- ❖ Test proposed by Alan Turing in 1950
- ❖ The computer is asked questions by a human interrogator.

The computer passes the test if a human interrogator, after posing some written questions, cannot tell whether the written responses come from a person or not. Programming a computer to pass, the computer 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.

To pass the complete Turing Test, the computer will need

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

2.1.2 Thinking humanly: The cognitive modeling approach

We need to get inside actual working of the human mind:

- (a) Through introspection – trying to capture our own thoughts as they go by;
- (b) Through psychological experiments

Allen Newell and Herbert Simon, who developed GPS, the —General Problem Solver tried to trace the reasoning steps to traces of human subjects solving the same problems. The interdisciplinary field of cognitive science brings together computer models from AI and experimental techniques from psychology to try to construct precise and testable theories of the workings of the human mind

2.1.3 Thinking rationally : The “laws of thought approach”

The Greek philosopher Aristotle was one of the first to attempt to codify —right thinking that is irrefutable (ie. Impossible to deny) reasoning processes. His syllogism provided patterns for argument structures that always yielded correct conclusions when given correct premises—for example, Socrates is a man; all men are mortal; therefore Socrates is mortal. These laws of thought were supposed to govern the operation of the mind; their study initiated a field called logic.

2.1.4 Acting rationally : The rational agent approach

An agent is something that acts. Computer agents are not mere programs, but they are expected to have the following attributes also: (a) operating under autonomous control, (b) perceiving their environment, (c) persisting over a prolonged time period, (e) adapting to change. A rational agent is one that acts so as to achieve the best outcome.

3. HISTORY OF AI

- 1943: early beginnings
 - McCulloch & Pitts: Boolean circuit model of brain
- 1950: Turing
 - Turing's "Computing Machinery and Intelligence“
- 1956: birth of AI
 - Dartmouth meeting: "Artificial Intelligence“name adopted
- 1950s: initial promise
 - Early AI programs, including
 - Samuel's checkers program
 - Newell & Simon's Logic Theorist

- 1955-65: “great enthusiasm”
 - Newell and Simon: GPS, general problem solver
 - Gelertner: Geometry Theorem Prover
 - McCarthy: invention of LISP
- 1966—73: Reality dawns
 - Realization that many AI problems are intractable
 - Limitations of existing neural network methods identified
 - Neural network research almost disappears
- 1969—85: Adding domain knowledge
 - Development of knowledge-based systems
 - Success of rule-based expert systems,
 - E.g., DENDRAL, MYCIN
 - But were brittle and did not scale well in practice
- 1986-- Rise of machine learning
 - Neural networks return to popularity
 - Major advances in machine learning algorithms and applications
- 1990-- Role of uncertainty
 - Bayesian networks as a knowledge representation framework
- 1995--AI as Science
 - Integration of learning, reasoning, knowledge representation
 - AI methods used in vision, language, data mining, etc

3.1 AI Technique

AI technique is a method that exploits knowledge that should be represented in such a way that:

- **The knowledge captures generalizations.** In other words, it is not necessary to represent separately each individual situation. Instead, situations that share important properties are grouped together. If knowledge does not have this property, inordinate amounts of memory and updating will be required. So we usually call something without this property "data" rather than knowledge.

- **It can be understood by people who must provide it.** Although for many programs, the bulk of the data can be acquired automatically (for example, by taking readings from a variety of

instruments), in many AI domains, most of the knowledge a program has must ultimately be provided by people in terms they understand.

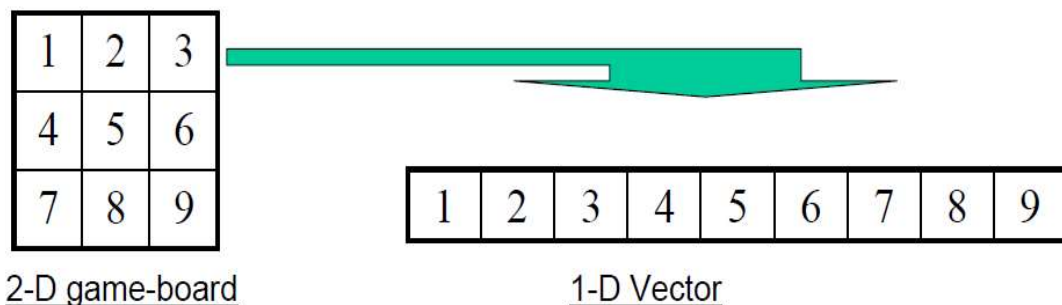
- **It can easily be modified to correct errors** and to reflect changes in the world and in our world view.
- **It can be used in a great many situations even if it is not totally accurate or complete.**
- **It can be used to help overcome its own sheer bulk by helping to narrow the range of possibilities that must usually be considered.**

Although AI techniques must be designed in keeping with these constraints imposed by AI problems, there is some degree of independence between problems and problem-solving techniques. It is possible to solve AI problems without using AI techniques (although, as we suggested above, those solutions are not likely to be very good).

Tic-Tac-Toe

Solution 1

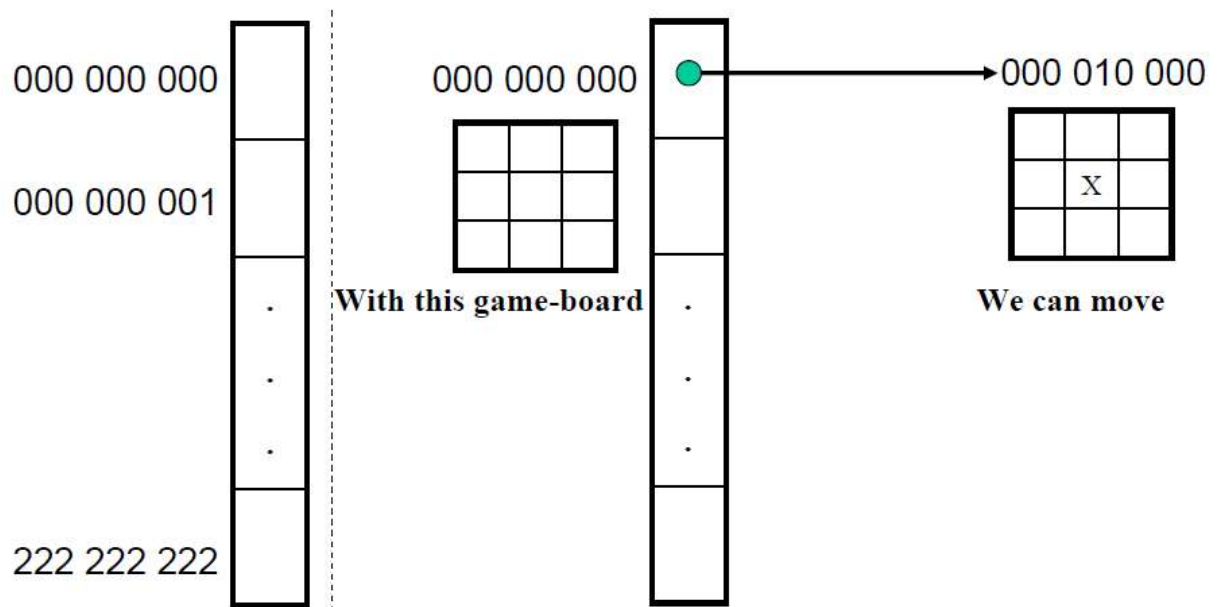
➤ **Data Structure:**



➤ **Elements of vector:**

- ✓ 0 : Empty
- ✓ 1 : X
- ✓ 2 : O

- The vector is a ternary number
- Store inside the program a move-table (lookup table):
- #Elements in the table: $19683 (=3^9)$
- Element = A vector which describes the most suitable move from the current game-board



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.

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.

Solution 2

Data Structure

- Use vector, called board, as Solution 1
- However, elements of the vector:
 - ✓ : Empty
 - ✓ : X
 - ✓ : O
- Turn of move: indexed by integer
 - ✓ 1,2,3, etc.

Function Library:

1. Make2:

- 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	2	3
4	5	6
7	8	9

- Let P represent for X or O
- can_win(P) :
 - ✓ P has filled already at least two cells on a straight line (horizontal, vertical, or diagonal)
- cannot_win(P) = NOT(can_win(P))

2. 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

3. Go(n): make a move.

```
IF odd(turn) THEN           // for X
    Board[n] = 3
ELSE                         // for O
    Board[n] = 5
turn = turn + 1
```

Algorithm:

1. Turn = 1: (X moves)

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

2. Turn = 2: (O moves)

```
IF board[5] is empty THEN
    Go(5)
ELSE
    Go(1)
```

3. Turn = 3: (X moves)

```
IF board[9] is empty THEN
    Go(9)
ELSE
    Go(3).
```

4. Turn = 4: (O moves)

```
IF Posswin(X) <> 0 THEN
    Go(Posswin(X))
    //Prevent the opponent to win
ELSE Go(Make2)
```

5. Turn = 5: (X moves)

```
IF Posswin(X) <> 0 THEN
    Go(Posswin(X))
    //Win for X.
ELSE IF Posswin(O) <> 0 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 move.
2. Easier to understand the program's strategy.
3. Hard to generalize.
4. Checking for a possible win is quicker.
5. Human finds the row-scan approach easier, while computer finds the number-counting approach more efficient.

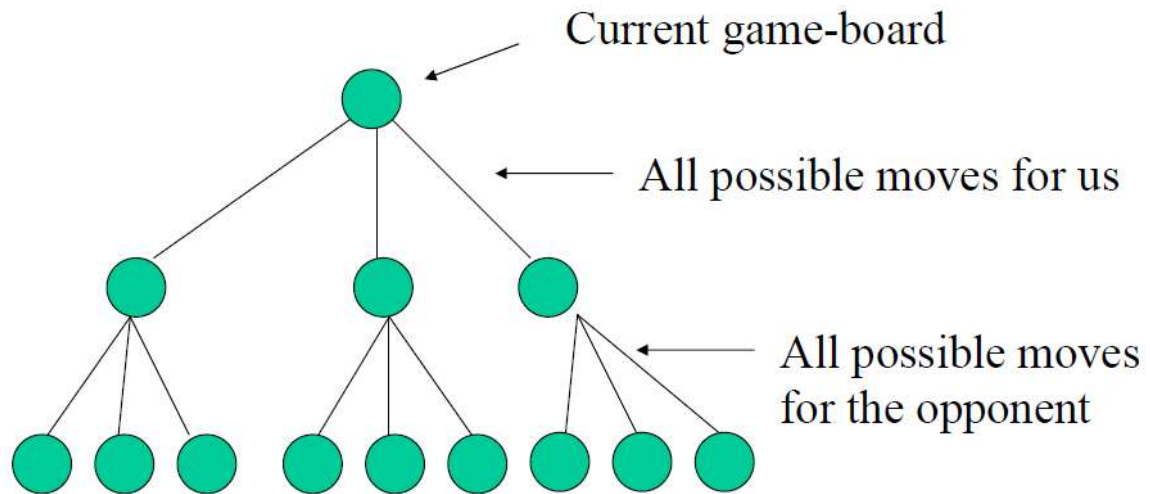
Solution 3

Data Structure

1. Game-board: Use vector as described for the above program
2. List:
 - ✓ Contains possible game-boards generated from the current game-board
 - ✓ Each the game-board is augmented a score indicating the possibility of victory of the current turn

Algorithm:

1. If it is a win, give it the highest rating.
2. Otherwise, consider all the moves the opponent could make next. Assume the opponent will make the move that is worst for us. Assign the rating of that move to the current node.
3. The best node is then the one with the highest rating.



Comments:

1. Require much more time to consider all possible moves.
2. Could be extended to handle more complicated games.

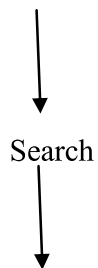
CHAPTER - 2

PROBLEMS, PROBLEM SPACES AND SEARCH

2. FORMULATING PROBLEMS

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

Formulate Goal, Formulate problem



Execute

2.1 WELL-DEFINED PROBLEMS AND SOLUTIONS

A problem can be defined formally by four components:

1. Initial state
2. Successor function
3. Goal test
4. Path cost

1. Initial State

The starting state which agent knows itself.

1. Successor Function

- A description of the possible actions available to the agent.
- State x , successor – FN (x) returns a set of $\langle \text{action}, \text{successor} \rangle$ ordered pairs, where each action is a legal action in a state x and each successor is a state that can be reached from x by applying that action.

2.1 State Space

The set of all possible states reachable from the initial state by any sequence of actions. The initial state and successor function defines the state space. The state space forms a graph in which nodes are state and axis between the nodes are action.

2.2 Path

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

2. Goal Test

Test to determine whether the given state is the goal state. If there is an explicit set of possible goal states, then we can whether any one of the goal state is reached or not.

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

3. Path cost

A function that assigns a numeric cost to each path. The cost of a path can be described as the sum of the costs of the individual actions along that path.

Step cost of taking an action ‘ a ’ to go from one state ‘ x ’ to state ‘ y ’ is denoted by $C(x,a,y)$

C-Cost , x,y - states , Action , Step costs are non-negative

These 4 elements are gathered into a data structure that is given as input to problem solving algorithm. A solution quality is measured by path cost function. An optimal solution has lowest path cost among all solutions.

Total cost = Path cost + Search cost

Example: Route finding problem

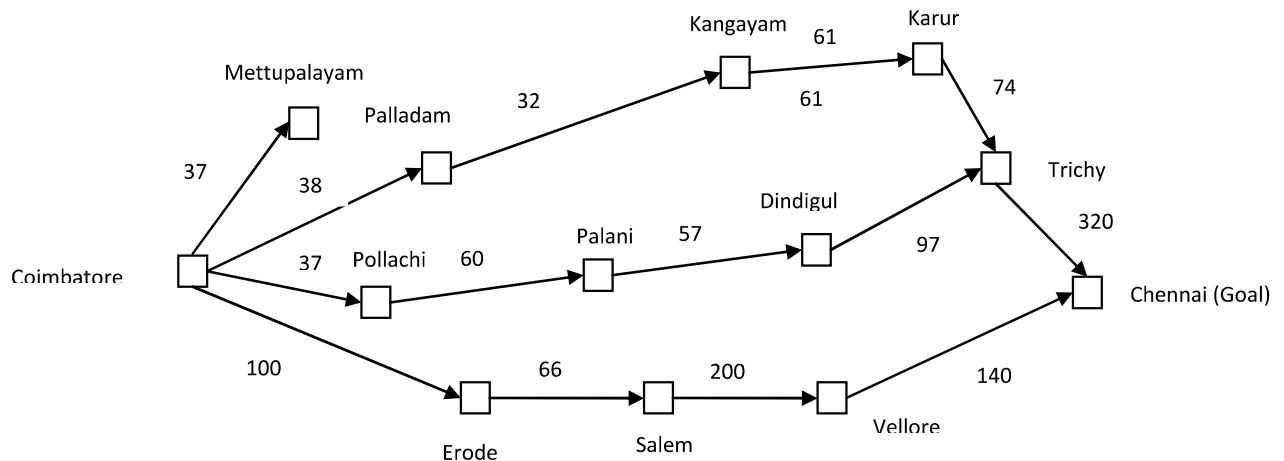


Fig: 1 Route Finding Problem

Initial State: In (Coimbatore)

Successor Function: {< Go (Pollachi), In (Pollachi)>

< Go (Erode), In (Erode)>

< Go (Palladam), In (Palladam)>

< Go (Mettupalayam), In (Mettupalayam)>}

Goal Test: In (Chennai)

Path Cost: {(In (Coimbatore)),}

{Go (Erode),} = 100 [kilometers]

{In (Erode)}

Path cost = 100 + 66 + 200 + 140 = 506

2.2 TOY PROBLEM

Example-1 : Vacuum World

Problem Formulation

- States

– 2 x 22 = 8 states

- Formula n^2n states
- Initial State
 - Any one of 8 states
- Successor Function
 - Legal states that result from three actions (Left, Right, Absorb)
- Goal Test
 - All squares are clean
- Path Cost
 - Number of steps (each step costs a value of 1)

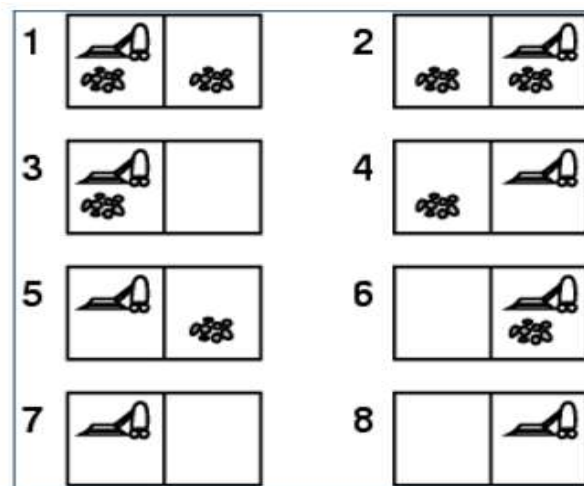


Fig 1.2 Vacuum World

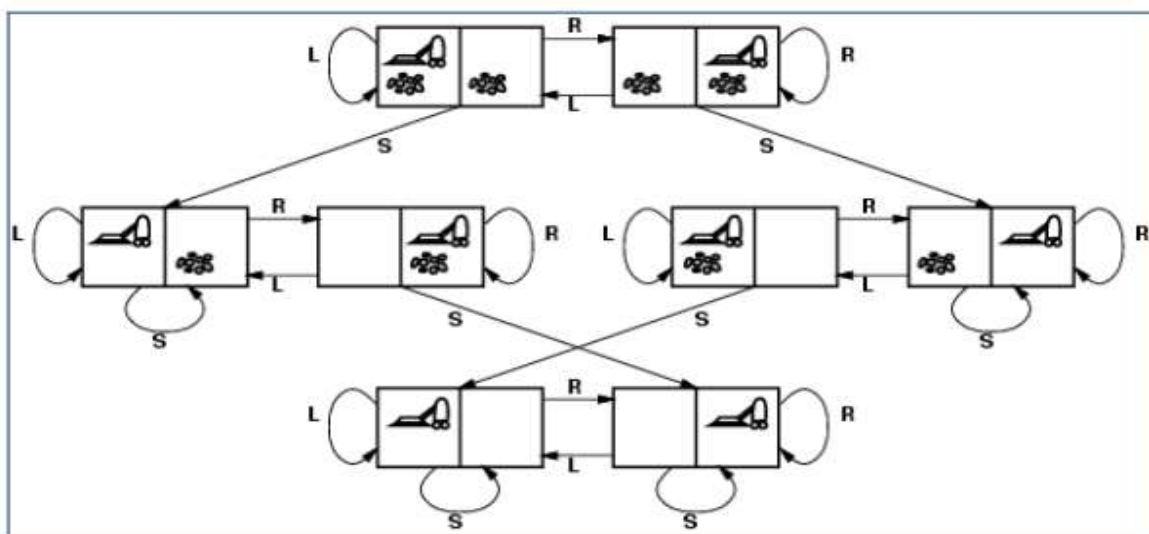


Fig: 1.3 State Space for the Vacuum World

State Space for the Vacuum World

Labels on Arcs denote L: Left, R: Right, S: Suck

Example 2: Playing chess

Initial State: Described as an 8 X 8 array where each positions contains a symbol standing for the appropriate piece in the official chess position.

Successor function: The legal states that results from set of rules.

They can be described easily by as a set of rules consisting of two parts: a left side that serves as a pattern to be matched against the current board position and a right side that describes the changes to be made to the board position to reflect the move. An example is shown in the following figure.

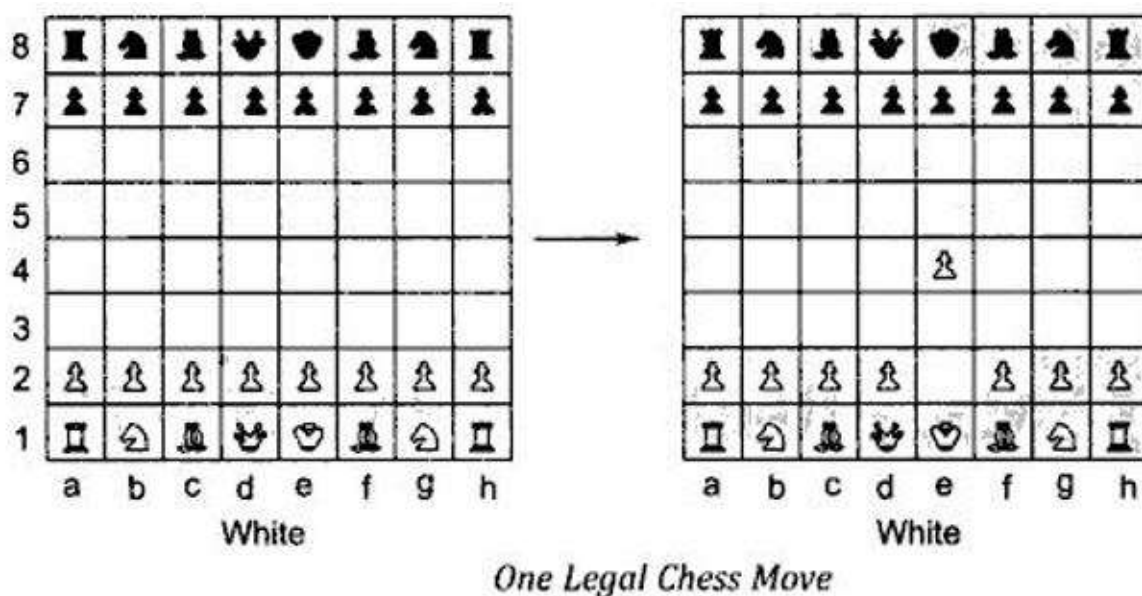


Fig 1.4: The legal states that results from set of rules

However if we write rules like the one above, we have to write a very large number of them since there has to be a separate set of rule for each of them roughly 10^{120} possible board positions.

Practical difficulties to implement large number of rules,

1. It will take too long to implement large number of rules and could not be done without mistakes.
2. No program could easily handle all those rules and storing it possess serious difficulties.

In order to minimize such problems, we have to write rules describing the legal moves in as a general way as possible. The following is the way to describe the chess moves.

Current Position

While pawn at square (e, 2), AND Square (e, 3) is empty, AND Square (e , 4) is empty.

Changing Board Position

Move pawn from Square (e, 2) to Square (e , 4) .

Some of the problems that fall within the scope of AI and the kinds of techniques will be useful to solve these problems.

GoalTest

Any position in which the opponent does not have a legal move and his or her king is under attack.

Example: 3 Water Jug 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: (x, y) x= 0, 1, 2, 3, or 4 y= 0, 1, 2, 3

x represents quantity of water in 4-gallon jug and y represents quantity of water in 3-gallon jug.

•**Start state:** (0, 0).

•**Goal state:** (2, n) for any n. Attempting to end up in a goal state.(since the problem doesn't specify the quantity of water in 3-gallon jug)

- | | | |
|-----------|------------|----------------------------|
| 1. (x, y) | →(4, y) | Fill the 4-gallon jug |
| If x <4 | | |
| 2. (x, y) | →(x, 3) | Fill the 3-gallon jug |
| If y <3 | | |
| 3. (x, y) | →(x -d, y) | Pour some water out of the |
| If x >0 | | 4-gallon jug |
| 4. (x, y) | →(x, y -d) | Pour some water out of the |
| If y >0 | | 3-gallon jug |

5. (x, y) If x > 0	$\rightarrow(0, y)$	Empty the 4-gallon jug on the ground
6. (x, y) If y > 0	$\rightarrow(x, 0)$	Empty the 3-gallon jug on the ground
7. (x, y) If x + y $\geq$ 4, y > 0	$\rightarrow(4, y - (4 - x))$	Pour water from the 3-gallon jug into the 4-gallon jug until the 4-gallon jug is full
8. (x, y) If x + y $\geq$ 3, x > 0	$\rightarrow(x - (3 - y), 3)$	Pour water from the 4-gallon jug into the 3-gallon jug until the 3-gallon jug is full
9. (x, y) If x + y $\leq$ 4, y > 0	$\rightarrow(x + y, 0)$	Pour all the water from the 3-gallon jug into the 4-gallon jug
10. (x, y) If x + y $\leq$ 3, x > 0	$\rightarrow(0, x + y)$	Pour all the water from the 4-gallon jug into the 3-gallon jug
11. (0, 2)	$\rightarrow(2, 0)$	Pour the 2 gallons from the 3-gallon Jug into the 4-gallon jug
12. (2, y)	$\rightarrow(0, y)$	Empty the 2 gallons in the 4-gallon Jug on the ground

Production rules for the water jug problem

Trace of steps involved in solving the water jug problem First solution

Number of Steps	Rules applied	4-g jug	3-g jug
1	Initial state	0	0
2	R2 {Fill 3-g jug}	0	3
3	R7 {Pour all water from 3 to 4-g jug}	3	0
4	R2 {Fill 3-g jug}	3	3
5	R5 {Pour from 3 to 4-g jug until it is full}	4	2
6	R3 {Empty 4-gallon jug}	0	2
7	R7 {Pour all water from 3 to 4-g jug}	2	0

Goal State

Second Solution

Number of Steps	Rules applied	4-g jug	3-g jug
1	Initial state	0	0
2	R1 {Fill 4-gallon jug}	4	0
3	R6 {Pour from 4 to 3-g jug until it is full}	1	3
4	R4 {Empty 3-gallon jug}	1	0
5	R8 {Pour all water from 4 to 3-gallon jug}	0	1
6	R1 {Fill 4-gallon jug}	4	1
7	R6 {Pour from 4 to 3-g jug until it is full}	2	3
8	R4 {Empty 3-gallon jug}	2	0

Goal State

Example - 5 8-puzzle Problem

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

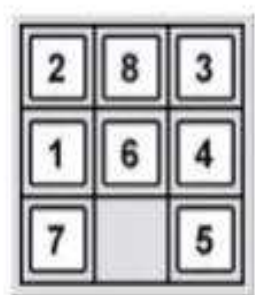
States: A state description specifies the location of each of the eight tiles and the blank in one of the nine squares.

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

Successor function: This generates the legal states that result from trying the four actions (blank moves Left, Right, Up, or Down).

Goal test: This checks whether the state matches the goal configuration (Other goal configurations are possible.)

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



Initial State



Goal State

Fig 1.5 8 Puzzle Problem

Exampe-6 8-queens problem

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

States: Any arrangement of 0 to 8 queens on the board is a state.

Initial state: No queens on the board.

Successor function: Add a queen to any empty square.

Goal test: 8 queens are on the board, none attacked.

Path cost: Zero (search cost only exists)

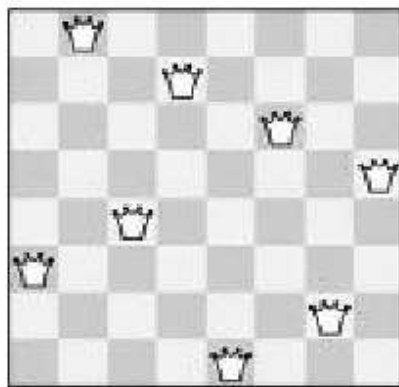


Fig 1.6 Solution to the 8 queens problem

2.3 PRODUCTION SYSTEMS

Production system is a mechanism that describes and performs the search process.

A production system consists of four basic components:

1. A set of rules of the form $C_i \rightarrow A_i$ where C_i is the condition part and A_i is the action part. The condition determines when a given rule is applied, and the action determines what happens when it is applied.
(i.e A set of rules, each consist of left side (a pattern) that determines the applicability of the rule and a right side that describes the operations to be performed if the rule is applied)
2. One or more knowledge databases that contain whatever information is relevant for the given problem. Some parts of the database may be permanent, while others may temporary and only exist during the solution of the current problem. The information in the databases may be structured in any appropriate manner.
3. A control strategy that determines the order in which the rules are applied to the database, and provides a way of resolving any conflicts that can arise when several rules match at once.

4. A rule applier which is the computational system that implements the control strategy and applies the rules.

In order to solve a problem

- ❖ We must first reduce it to one for which a precise statement can be given. This can be done by defining the problem's state space (start and goal states) and a set of operators for moving that space.
- ❖ The problem can be solved by searching for a path through the space from the initial state to a goal state.
- ❖ The process of solving the problem can be usefully modeled as a production system.

2.3.1 Control strategies

By considering control strategies we can decide which rule to apply next during the process of searching for a solution to problem.

The two requirements of good control strategy are that

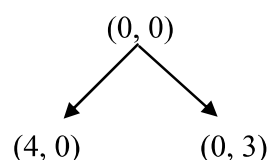
- ❖ **It should cause motion:** consider water jug problem, if we implement control strategy of starting each time at the top of the list of rules, it will never leads to solution. So we need to consider control strategy that leads to solution.
- ❖ **It should be systematic:** choose at random from among the applicable rules. This strategy is better than the first. It causes the motion. It will lead to the solution eventually. Doing like this is not a systematic approach and it leads to useless sequence of operators several times before finding final solution.

2.3.1.1 Systematic control strategy for the water jug problem

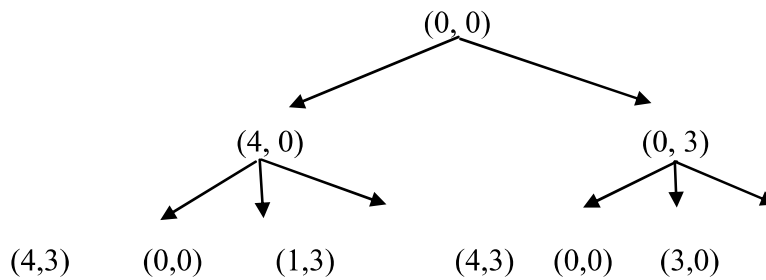
2.3.1.1.1 Breadth First Search (Blind Search)

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.



❖ Continue this process until some rule produces a goal state.

Algorithm

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

2.3.1.1.2 Depth First Search

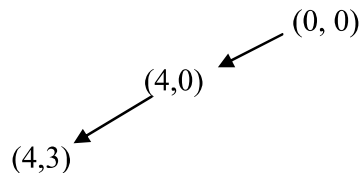
Algorithm

1. If the initial state is the goal state, quit and return success.
2. Otherwise do the following until success or failure is signaled:
 - a. Generate a successor, E, of initial state. If there are no more successor, signal failure.
 - b. Call depth first search, with E as the initial state.
 - c. If the success is returned, signal success. Otherwise continue in this loop.

Backtracking

- ❖ In this search, we pursue a single branch of the tree until it yields a solution or until a decision to terminate the path is made.
- ❖ It makes sense to terminate a path if it reaches dead-end, produces a previous state. In such a state backtracking occurs.

- ❖ Chronological Backtracking: order in which steps are undone depends only on the temporal sequence in which steps were initially made.
- ❖ Specifically most recent step is always the first to be undone. This is also simple backtracking.



Advantages of Depth First search

- ❖ DFS requires less memory since only the nodes on the current path are stored.
- ❖ By chance DFS may find a solution without examining much of the search space at all.

Advantages of Breath First search

- ❖ BFS cannot be trapped exploring a blind alley.
- ❖ If there is a solution, BFS is guaranteed to find it.
- ❖ If there are multiple solutions, then a minimal solution will be found.

Traveling Salesman Problem (with 5 cities):

A salesman is supposed to visit each of 5 cities shown below. There is a road between each pair of cities and the distance is given next to the roads. Start city is A. The problem is to find the shortest route so that the salesman visits each of the cities only once and returns to back to A.

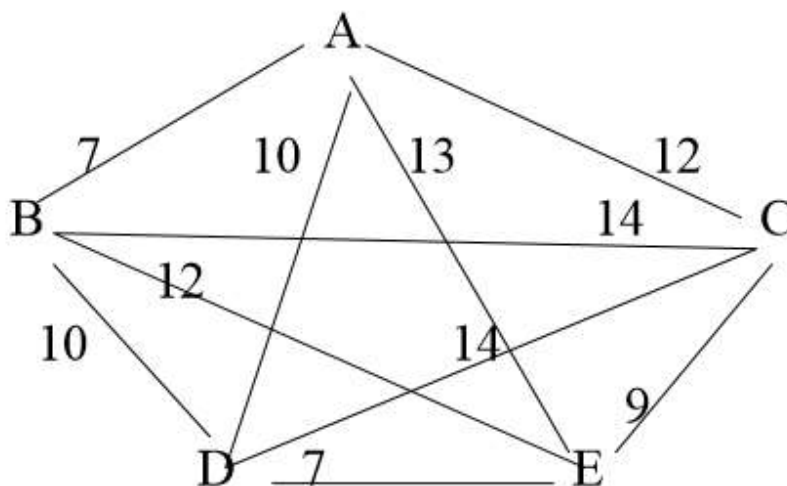


Fig Travelling Salesman Problem

- A simple, motion causing and systematic control structure could, in principle solve this problem.
- Explore the search tree of all possible paths and return the shortest path.
- This will require $4!$ paths to be examined.
- If number of cities grow, say 25 cities, then the time required to wait a salesman to get the information about the shortest path is of $O(24!)$ which is not a practical situation.
- This phenomenon is called combinatorial explosion.
- We can improve the above strategy as follows.

Branch and Bound

- ❖ Begin generating complete paths, keeping track of the shortest path found so far.
- ❖ Give up exploring any path as soon as its partial length become greater than the shortest path found so far.
- ❖ This algorithm is efficient than the first one, still requires exponential time $\propto$ some number raised to N .

2.3.2 Heuristic Search

- Heuristics are criteria for deciding which among several alternatives be the most effective in order to achieve some goal.
- Heuristic is a technique that improves the efficiency of a search process possibly by sacrificing claims of systematic and completeness. It no longer guarantees to find the best answer but almost always finds a very good answer.
- Using good heuristics, we can hope to get good solution to hard problems (such as travelling salesman) in less than exponential time.
- There are **general-purpose** heuristics that are useful in a wide variety of problem domains.
- We can also construct **special purpose** heuristics, which are domain specific.

2.3.2.1 General Purpose Heuristics

- A general-purpose heuristics for combinatorial problem is nearest neighbor algorithms which works by selecting the locally superior alternative.
- For such algorithms, it is often possible to prove an upper bound on the error which provide reassurance that one is not paying too high a price in accuracy for speed.
- In many AI problems, it is often hard to measure precisely the goodness of a particular solution.

- For real world problems, it is often useful to introduce heuristics based on relatively unstructured knowledge. It is impossible to define this knowledge in such a way that mathematical analysis can be performed.
- In AI approaches, behavior of algorithms are analyzed by running them on computer as contrast to analyzing algorithm mathematically.
- There are at least many reasons for the adhoc approaches in AI.
 - ❖ It is a lot more fun to see a program do something intelligent than to prove it.
 - ❖ AI problem domains are usually complex, so generally not possible to produce analytical proof that a procedure will work.
 - ❖ It is even not possible to describe the range of problems well enough to make statistical analysis of program behavior meaningful.
- But still it is important to keep performance question in mind while designing algorithm.
 - One of the most important analysis of the search process is straightforward i.e., “Number of nodes in a complete search tree of depth D and branching factor F is F^D ”.
- This simple analysis motivates to
 - ❖ Look for improvements on the exhaustive search.
 - ❖ Find an upper bound on the search time which can be compared with exhaustive search procedures.

2.4 PROBLEM CHARACTERISTICS

Heuristic search is a very general method applicable to a large class of problem. In order to choose the most appropriate method (or combination of methods) for a particular problem it is necessary to analyze the problem along several key dimensions:

2.4.1 Is the problem decomposable into a set of independent smaller sub problems?

Example: Suppose we want to solve the problem of computing the integral of the following expression $\int (x^2 + 3x + \sin^2 x * \cos^2 x) dx$

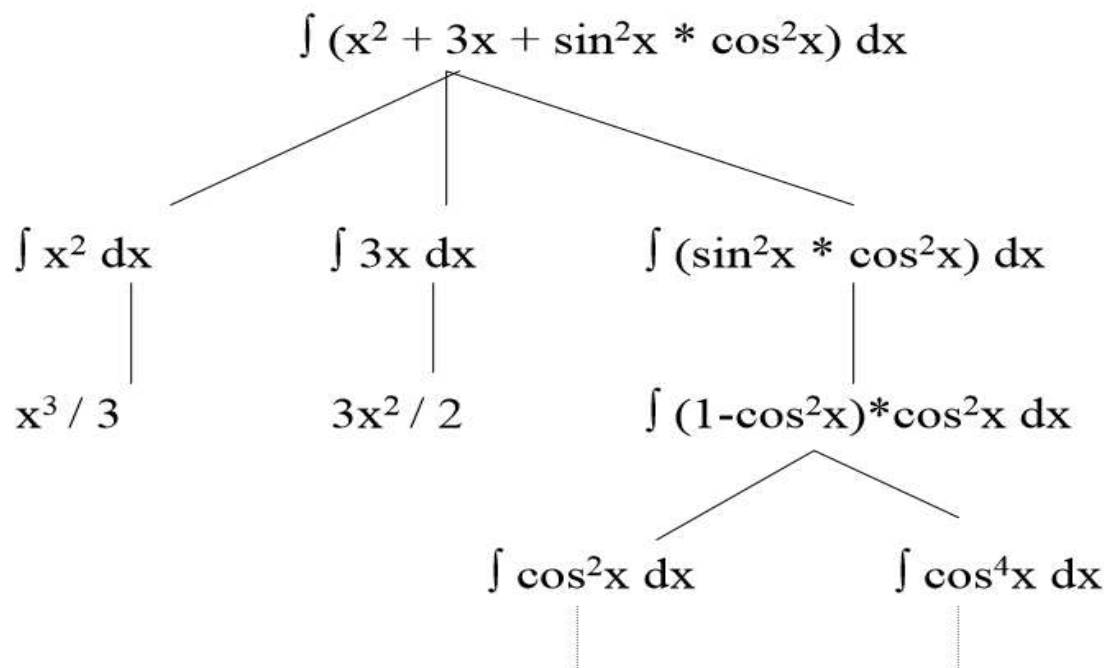


Fig 1.7 Decomposition problem

- We can solve this problem by breaking it down into three smaller sub problems, each of which we can then be solved using a small collection of specific rules.

- Decomposable problems can be solved by the divide and-conquer technique.

- Use of decomposing problems:

- Each sub-problem is simpler to solve

- Each sub-problem can be handed over to a different processor. Thus can be solved in parallel processing environment.

- There are non-decomposable problems. For example, Block world problem is non decomposable.

2.4.2. Can solution steps be ignored or at least undone if they prove to be unwise?

- In real life, there are three types of problems: Ignorable, Recoverable and Irrecoverable.

- Let us explain each of these through examples.

Example1 :(Ignorable): In theorem proving-(solution steps can be ignored)

- Suppose we have proved some lemma in order to prove a theorem and eventually realized that lemma is no help at all, then ignore it and prove another lemma.

- Can be solved by using simple control strategy.

Example2: (Recoverable):8 puzzle-(solution steps can be undone)

- 8 puzzle: Objective is to rearrange a given initial configuration of eight numbered tiles on 3 X 3 board (one place is empty) into a given final configuration (goal state).
- Rearrangement is done by sliding one of the tiles into Empty Square.
- Solved by backtracking so control strategy must be implemented using a push down stack.

Example3: (Irrecoverable): Chess (solution steps cannot be undone)

- A stupid move cannot be undone
- Can be solved by planning process

2.4.3. Is the knowledge Base consistent?

Example: Inconsistent knowledge:

Target problem: A man is standing 150 ft from a target. He plans to hit the target by shooting a gun that fires bullets with velocity of 1500 ft/sec. How high above the target should he aim?

Solution:

- Velocity of bullet is 1500 ft./sec i.e., bullet takes 0.1 sec to reach the target.
- Assume bullet travels in a straight line.
- Due to gravity, the bullet falls at a distance $(1/2)gt^2 = (1/2)(32)(0.1)^2 = 0.16\text{ft.}$
- So if man aims up 0.16 feet high from the target, then bullet will hit the target.
- Now there is a contradiction to the fact that bullet travel in a straight line because the bullet in actual will travel in an arc. Therefore there is inconsistency in the knowledge used.

2.4.4. What is the Role of knowledge?

- In Chess game, knowledge is important to constrain the search for a solution otherwise just the rule for determining legal moves and some simple control mechanism that implements an appropriate search procedure is required.
- Newspapers scanning to decide some facts, a lot of knowledge is required even to be able to recognize a solution.

2.4.5. Is a good solution Absolute or Relative?

- In water jug problem there are two ways to solve a problem. If we follow one path successfully to the solution, there is no reason to go back and see if some other path might also lead to a solution. Here a solution is absolute.
- In travelling salesman problem, our goal is to find the shortest route. Unless all routes are known, the shortest is difficult to know. This is a best-path problem whereas water jug is any-path problem.

- Any path problem can often be solved in reasonable amount of time using heuristics that suggest good paths to explore.

- Best path problems are in general computationally harder than any-path.

2.4.6. Does the task Require Interaction with a Person?

- **Solitary problem**, in which there is no intermediate communication and no demand for an explanation of the reasoning process.
- **Conversational problem**, in which intermediate communication is to provide either additional assistance to the computer or additional information to the user.

2.4.7. Problem classification

- There is a variety of problem-solving methods, but there is no one single way of solving all problems.
- Not all new problems should be considered as totally new. Solutions of similar problems can be exploited.

2.5 PRODUCTION SYSTEM CHARACTERISTICS

Production systems are important in building intelligent matches which can provide us a good set of production rules, for solving problems.

There are four types of production system characteristics, namely

1. Monotonic production system
2. Non-monotonic production system
3. Commutative law based production system, and lastly
4. Partially commutative law based production system

1. **Monotonic Production System (MPS):** The Monotonic production system (MPS) is a system in which the application of a rule never prevents later application of the another rule that could also have been applied at the time that the first rule was selected

2. **Non-monotonic Production (NMPS):** The non-monotonic production system is a system in which the application of a rule prevents the later application of the another rule which may not have been applied at the time that the first rule was selected, i.e. it is a system in which the above rule is not true, i.e. the monotonic production system rule not true.

3. **Commutative Production System (CPS):** Commutative law based production systems is a system in which it satisfies both monotonic & partially commutative.

4. **Partially Commutative Production System (PCPS):** The partially commutative production system is a system with the property that if the application of those rules that is allowable & also transforms from state x to state 'y'.

	Monotonic (Characteristics)	Non-monotonic
Partially commutative	Theorem proving	Robot navigation
Non-partial commutative	Chemical synthesis	Bridge game

Table 1.1 The Four categories of Production System

Well the question may arise here such as:

- can the production systems be described by a set of characteristics?
- Also, can we draw the relationship between problem types & the types of production systems, suited to solve the problems, yes, we can by using above rules.