UNIT I

INTRODUCTION TO Al 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.**

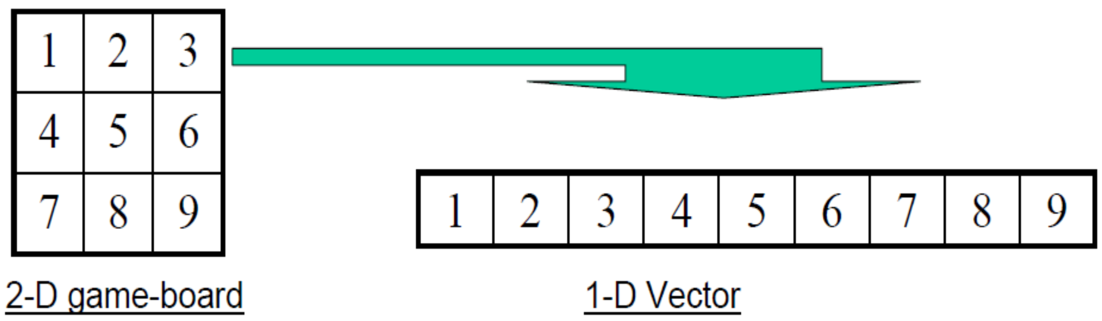
• **It can be used to help overcome its own sheer bulk by helping to narrow the range of possibilities that must usually be considered.**

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).

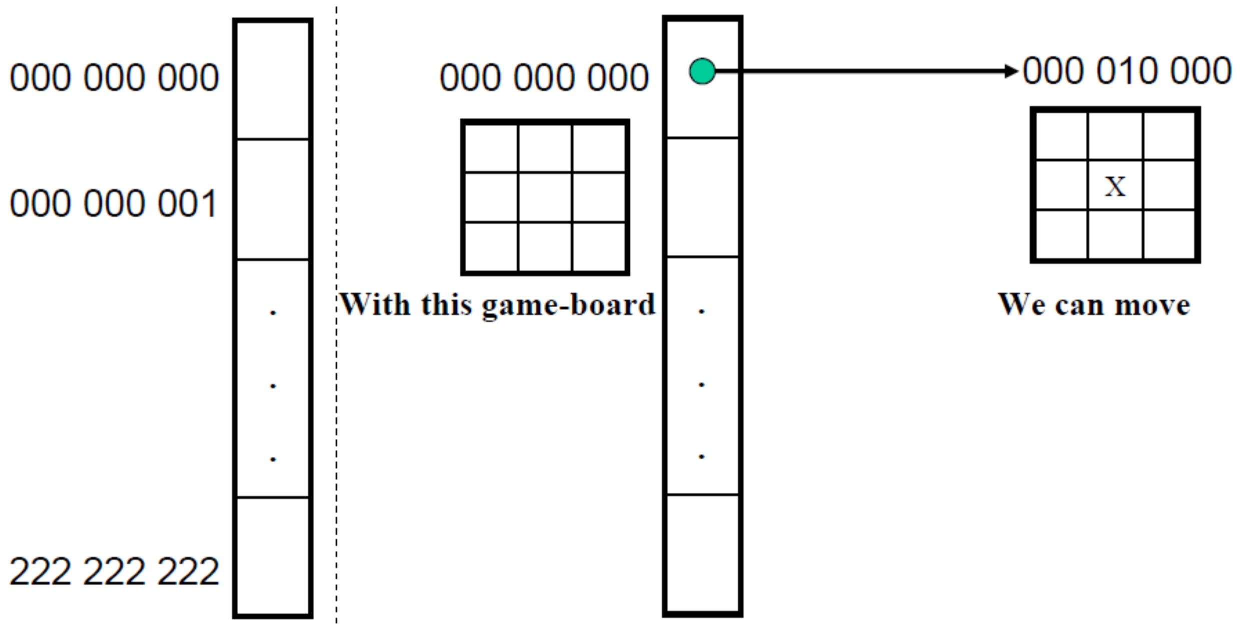
Tic-Tac-Toe

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) <> 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:

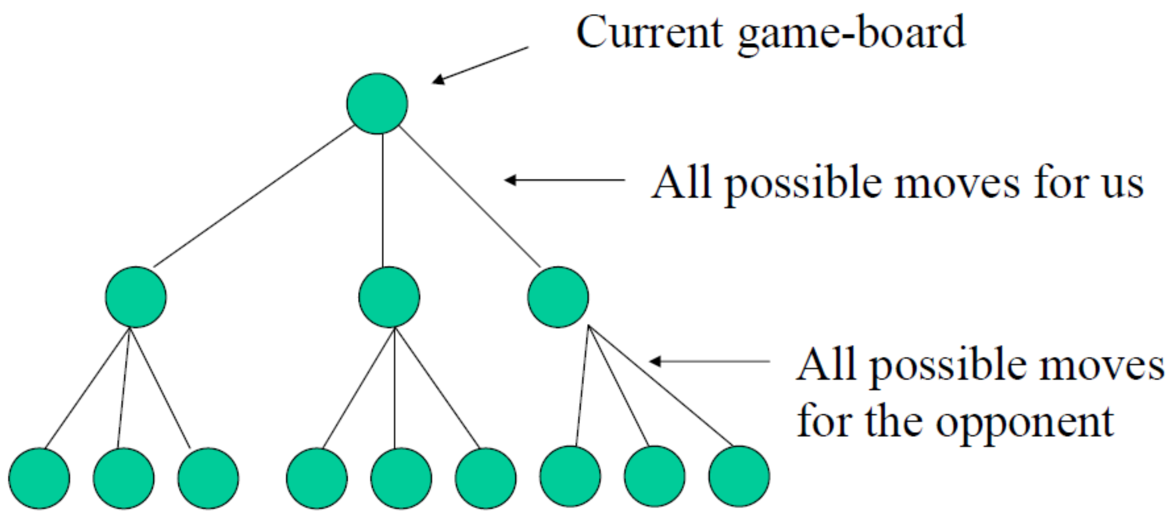
- ✓ Contains possible game-boards generated from the current game-board
- ✓ Each the game-board is augmented a score indicating the possibility of victory of the current turn

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.


Current game-board
All possible moves for us
All possible moves for the opponent

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

$$\begin{tikzcd}
\text{Formulate Goal, Formulate problem} \arrow[d] \\
\text{Search} \arrow[d] \\
\text{Execute}
\end{tikzcd}$$

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 < action, successor> 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

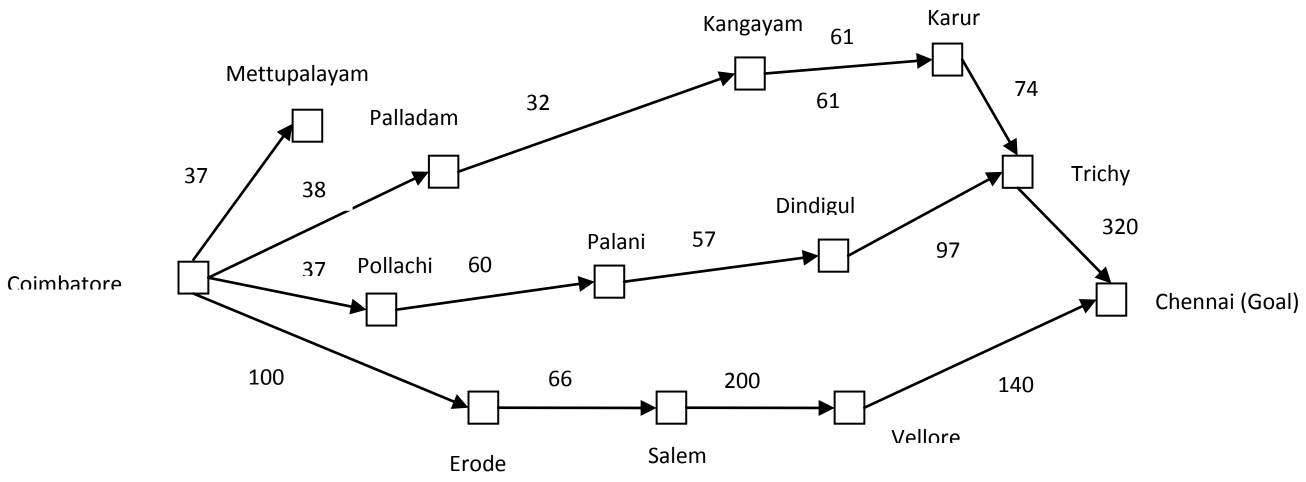


Fig: 1 Route Finding Problem

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 n2n 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)

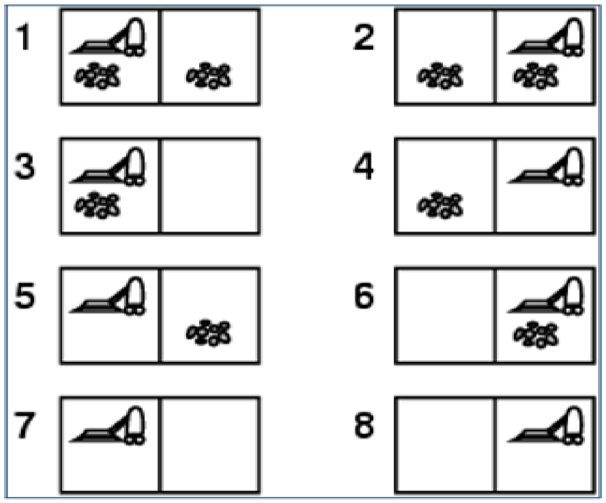


Fig 1.2 Vacuum World

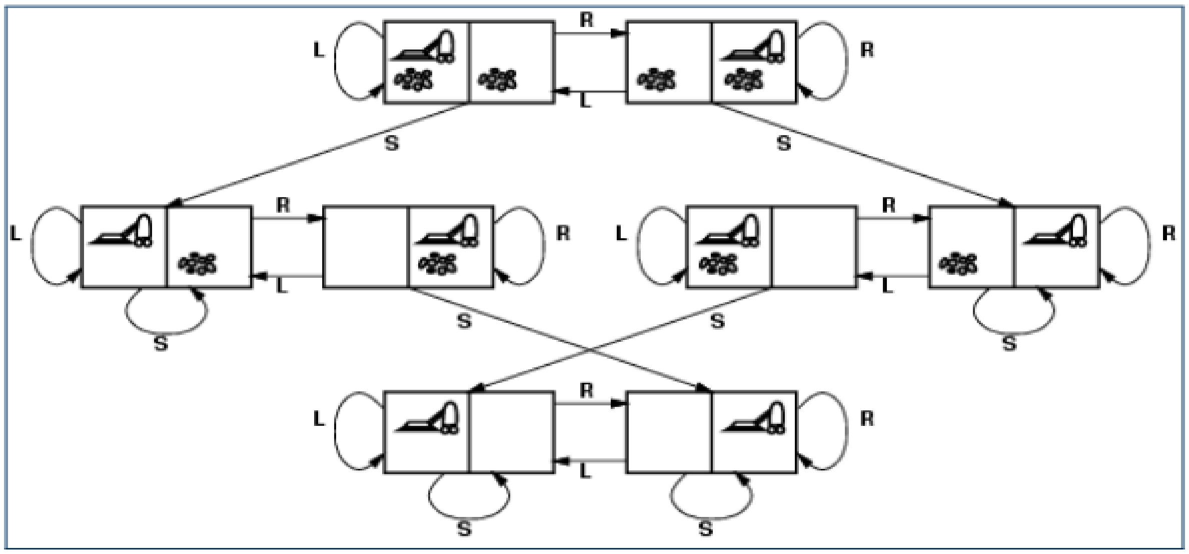


Fig: 1.3 State Space for the Vacuum World

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.

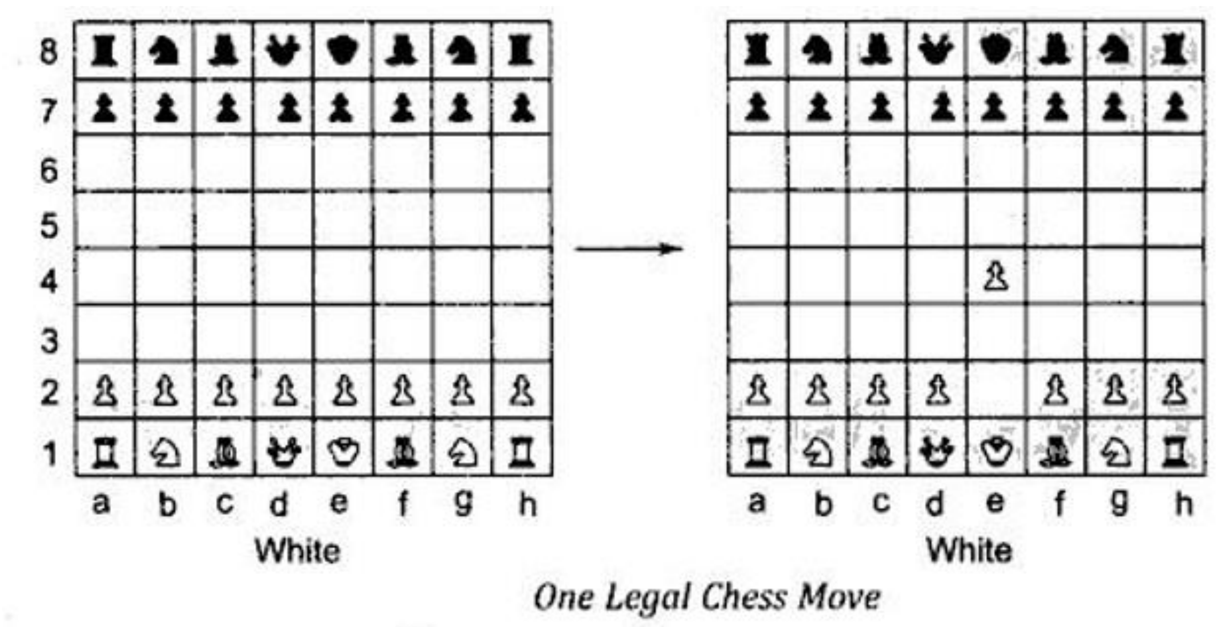


Fig 1.4:The legal states that results from set of rules

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

5. (x, y) If x >0	→(0, y)	Empty the 4-gallon jug on the ground
6. (x, y) If y >0	→(x, 0)	Empty the 3-gallon jug on the ground
7. (x, y) If x +y ≥4,y >0	→(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 ≥3,x >0	→(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 ≤4,y >0	→(x +y, 0)	Pour all the water from the 3-gallon jug into the 4-gallon jug
10. (x, y) If x +y ≤3,x >0	→(0, x +y)	Pour all the water from the 4-gallon jug into the 3-gallon jug
11. (0, 2)	→(2, 0)	Pour the 2 gallons from the 3-gallon Jug into the 4-gallon jug
12. (2, y)	→(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.

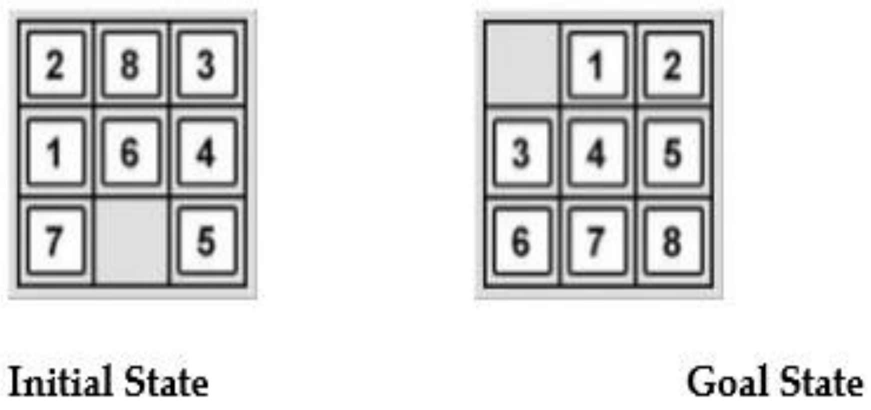


Fig 1.5 8 Puzzle Problem

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)

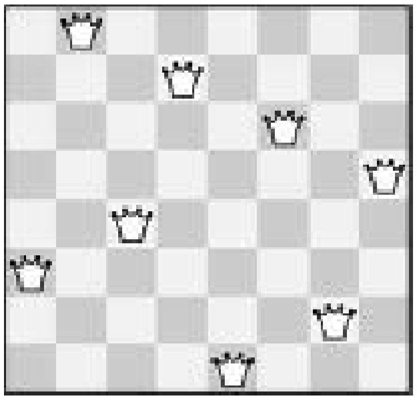


Fig 1.6 Solution to the 8 queens problem

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 Ci → Ai where Ci is the condition part and Ai 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.

$$\begin{tikzcd}
 & (0,0) \arrow[dl] \arrow[dr] & \\
(4,0) & & (0,3)
\end{tikzcd}$$

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

$$\begin{tikzcd}
 & & (0,0) \arrow[dl] \arrow[drr] & & & \\
 & (4,0) \arrow[dl] \arrow[d] \arrow[dr] & & & (0,3) \arrow[dl] \arrow[d] \arrow[dr] & \\
(4,3) & (0,0) & (1,3) & (4,3) & (0,0) & (3,0)
\end{tikzcd}$$

- ❖ 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.

$$\begin{tikzcd}
 & & (0,0) \arrow[dl] \\
 & (4,0) \arrow[dl] & \\
(4,3) & &
\end{tikzcd}$$

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.

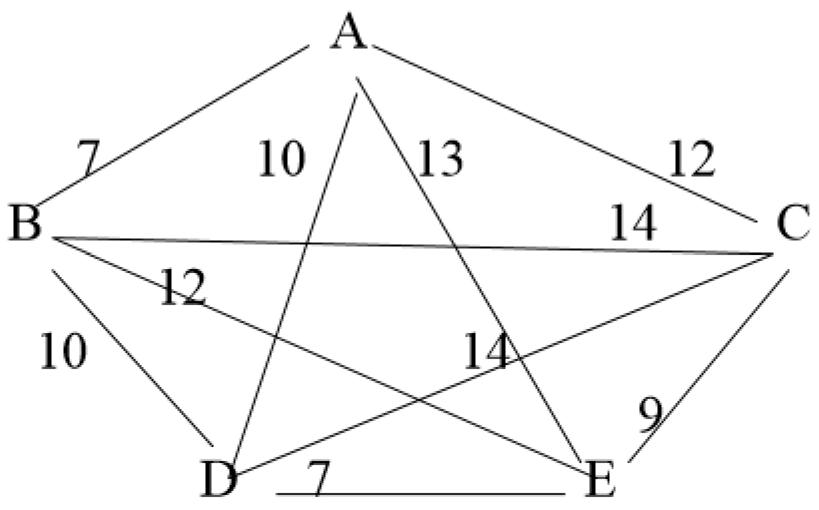


Fig Travelling Salesman Problem

•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 0(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 ∝ 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 $\int(x^2 + 3x + \sin^2 x * \cos^2 x)\, dx$

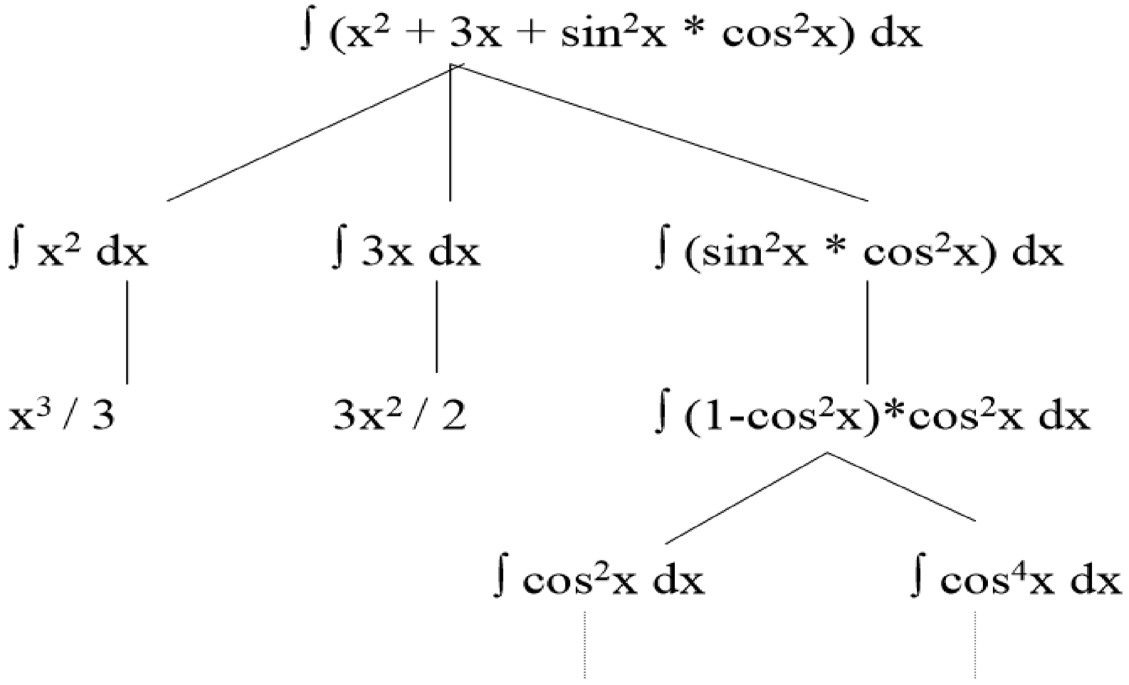


Fig 1.7 Decomposition problem

•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 secto 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 = 0.16ft.

•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.

CHAPTER - 3

PROBLEM SOLVING METHODS, HEURISTIC SEARCH TECHNIQUES

Search techniques are the general problem solving methods. When there is a formulated search problem, a set of search states, a set of operators, an initial state and a goal criterion we can use search techniques to solve a problem.

3.1 Matching:

Problem solving can be done through search. Search involves choosing among the rules that can be applied at a particular point, the ones that are most likely to lead to a solution. This can be done by extracting rules from large number of collections.

How to extract from the entire collection of rules that can be applied at a given point?

- ❖ This can be done by Matching between current state and the precondition of the rules.

One way to select applicable rules is to do simple search through all the rules comparing each one's preconditions to the current state and extracting the one that match. But there are two problems with this simple solution.

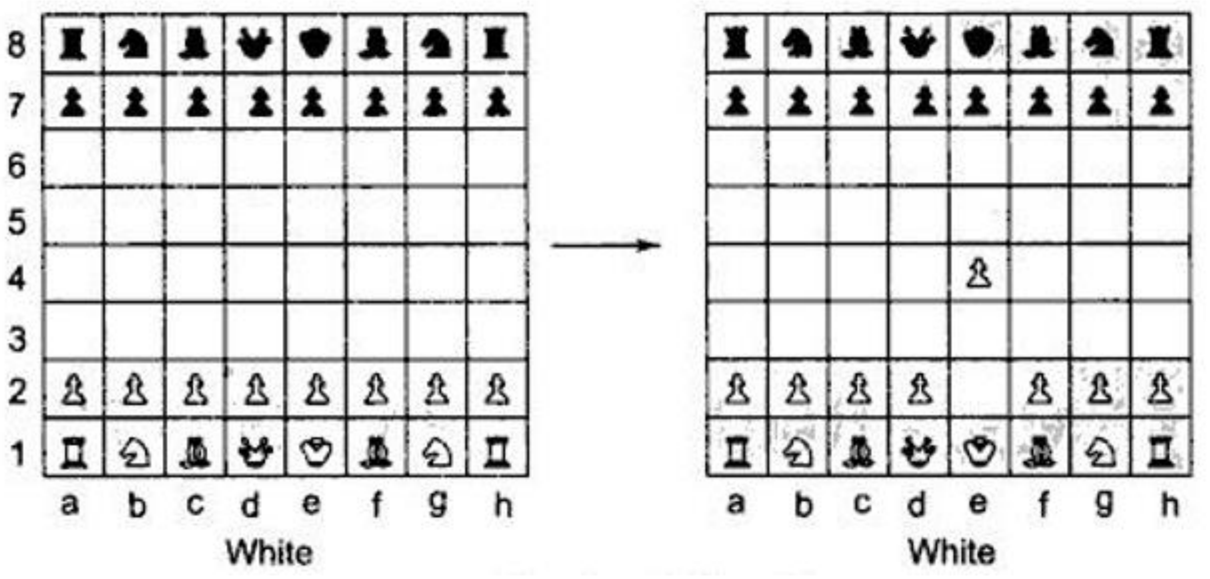
1.In big problems large number of rules are used. Scanning through all of this rules at every step of the search would be hopelessly inefficient.

2.It is not clearly visible to find out which condition will be satisfied.

Some of the matching techniques are described below:

3.1.1 Indexing: To overcome above problems indexing is used. In this instead of searching all the rules the current state is used as index into the rules, and select the matching rules immediately e.g consider the chess game playing. Here the set of valid moves is very large. To reduce the size of this set only useful moves are identified. At the time of playing the game, the next move will very much depend upon the current move. As the game is going on there will be only 'few' moves which are applicable in next move. Hence it will be wasteful effort to check applicability of all moves. Rather, the important and valid legal moves are directly stored as rules and through indexing the applicable rules are found. Here, the indexing will store the current board position. The indexing make the matching process easy, at the cost of lack of generality in the statement rules. Practically there is a tradeoff between the ease of writing rules and simplicity of matching process. The indexing technique is not very well suited for the rule base where rules are written in high level predicates. In PROLOG and many theorem proving systems, rules are indexed by predicates they contain. Hence all the applicable rules can be indexed quickly.

Example:



1.8 One Legal Chess Move

3.1.1.1 Matching with variable: In the rule base if the preconditions are not stated as exact description of particular situation, the indexing technique does not work well. In certain situations they describe properties that the situation must have. In situation, where single conditions is matched against a single element in state description, the unification procedure can be used. However in practical situation it is required to match complete set of rules that match the current state. In forward and backward chaining system, the depth first searching technique is used to select the individual rule. In the situation where multiple rules are applicable, conflict resolution technique is used to choose appropriate applicable rule. In the case of the situations requiring multiple match, the unification can be applied recursively, but more efficient method is used to use RETE matching algorithm.

3.1.1.2 **Complex matching variable**: A more complex matching process is required when preconditions of a rule specify required properties that are not stated explicitly in the description of current state. However the real world is full of uncertainties and sometimes practically it is not possible to define the rule in exact fashion. The matching process become more complicated in the situation where preconditions approximately match the current situation e.g a speech understanding program must contain the rules that map from a description of a physical wave form to phones. Because of the presence of noise the signal becomes more variable that there will be only approximate match between the rules that describe an ideal sound and the input that describes that unideal world. Approximate matching is particularly difficult to deal with, because as we increase the tolerance allowed in the match the new rules need to be written and it will increase number of rules. It will increase the size of main search process. But approximate matching is nevertheless superior to exact matching in situation such as speech understanding, where exact matching may result in no rule being matched and the search process coming to a grinding halt.

3.2 HEURISTIC SEARCH TECHNIQUES

3.2.1 Hill Climbing

- Hill climbing is the optimization technique which belongs to a family of local search. It is relatively simple to implement, making it a popular first choice. Although more advanced algorithms may give better results in some situations hill climbing works well.

- Hill climbing can be used to solve problems that have many solutions, some of which are better than others. It starts with a random (potentially poor) solution, and iteratively makes small changes to the solution, each time improving it a little. When the algorithm cannot see any improvement anymore, it terminates. Ideally, at that point the current solution is close to optimal, but it is not guaranteed that hill climbing will ever come close to the optimal solution.

- For example, hill climbing can be applied to the traveling salesman problem. It is easy to find a solution that visits all the cities but is be very poor compared to the optimal solution. The algorithm starts with such a solution and makes small improvements to it, such as switching the order in which two cities are visited. Eventually, a much better route is obtained.

- Hill climbing is used widely in artificial intelligence, for reaching a goal state from a starting node. Choice of next node and starting node can be varied to give a list of related algorithms.

- Hill climbing attempts to maximize (or minimize) a function f (x), where x are discrete states. These states are typically represented by vertices in a graph, where edges in the graph encode nearness or similarity of a graph. Hill climbing will follow the graph from vertex to vertex, always locally increasing (or decreasing) the value of f, until a local maximum (or local minimum) x_m is reached. Hill climbing can also operate on a continuous space: in that case, the algorithm is called gradient ascent (or gradient descent if the function is minimized).*.

- Problems with hill climbing: local maxima (we've climbed to the top of the hill, and missed the mountain), plateau (everything around is about as good as where we are),ridges (we're on a ridge leading up, but we can't directly apply an operator to improve our situation, so we have to apply more than one operator to get there).Solutions include: backtracking, making big jumps (to handle plateaus or poor local maxima), applying multiple rules before testing (helps with ridges).Hill climbing is best suited to problems where the heuristic gradually improves the closer it gets to the solution; it works poorly where there are sharp drop-offs. It assumes that local improvement will lead to global improvement.

- Search methods based on hill climbing get their names from the way the nodes are selected for expansion. At each point in the search path a successor node that appears to lead most quickly to the top of the hill (goal) selected for exploration. This method requires that some information be available with which to evaluate and order the most promising choices. Hill climbing is like depth first searching where the most promising child is selected for expansion.

- Hill climbing is a variant of generate and test in which feedback from the test procedure is used to help the generator decide which direction to move in the search space. Hill climbing is often used when a good heuristic function is available for evaluating states but when no other useful knowledge is available. For example, suppose you are in an unfamiliar city without a map and you want to get downtown. You simply aim for the tall buildings. The heuristic function is just distance between the current location and the location of the tall buildings and the desirable states are those in which this distance is minimized.

3.2.1 Simple Hill Climbing

The simplest way to implement hill climbing is the simple hill climbing whose algorithm is as given below:

Algorithm: Simple Hill Climbing:

```
Step 1: Evaluate the initial state. It it is also a goal state, then return it and quit. Otherwise continue with the initial state as the current state.
Step 2: Loop until a solution is found or until there are no new operators left to be applied in the current state:
    (a)Select an operator that has not yet been applied to the current state and apply it to produce a new state.
    (b)Evaluate the new state.
        (i)If it is a goal state, then return it and quit.
        (ii)If it is not a goal state, but it is better than the current state, then make it the current state.
        (iii)If it is not better than the current state, then continue in the loop.
```

The key difference between this algorithm and the one we gave for generate and test is the use of an evaluation function as a way to inject task specific knowledge into the control process. It is the use of such knowledge that makes this heuristic search method. It is the same knowledge that gives these methods their power to solve some otherwise intractable problems

To see how hill climbing works, let’s take the puzzle of the four colored blocks. To solve the problem we first need to define a heuristic function that describes how close a particular configuration is to being a solution. One such function is simply the sum of the number of different colors on each of the four sides. A solution to the puzzle will have a value of 16. Next we need to define a set of rules that describe ways of transforming one configuration to another. Actually one rule will suffice. It says simply pick a block and rotate it 90 degrees in any direction. Having provided these definitions the next step is to generate a starting configuration. This can either be done at random or with the aid of the heuristic function. Now by using hill climbing, first we generate a new state by selecting a block and rotating it. If the resulting state is better than we keep it. If not we return to the previous state and try a different perturbation.

3.2.2 Problems in Hill Climbing

3.2.2.1 Steepest – Ascent Hill Climbing:

An useful variation on simple hill climbing considers all the moves form the current state and selects the best one as the next state. This method is called steepest – ascent hill climbing or gradient search. Steepest Ascent hill climbing contrasts with the basic method in which the first state that is better than the current state is selected. The algorithm works as follows.

Algorithm: Steepest – Ascent Hill Climbing

```
Step 1: Evaluate the initial state. If it is also a goal state, then return it and quit .Otherwise, continue with the initial state as the current state.

Step 2: Loop until a solution is found or until a complete iteration produces no change to current state:

    (a)Let SUCC be a state such that any possible successor of the current state will be better than SUCC.

    (b)For each operator that applies to the current state do:

        i. Apply the operator and generate a new state.

        ii. Evaluate the new state. If it is a goal state, then return it and quit. If not, compare it to SUCC. If it is better then set SUCC to this state. If it is not better, leave SUCC alone.

    (c) If the SUCC is better than current state, then set current state to SUCC.
```

To apply steepest- ascent hill climbing to the colored blocks problem, we must consider all perturbations of the initial state and choose the best. For this problem this is difficult since there are so many possible moves. There is a trade-off between the time required to select a move and the number of moves required to get a solution that must be considered when deciding which method will work better for a particular problem. Usually the time required to select a move is longer for steepest – ascent hill climbing and the number of moves required to get to a solution is longer for basic hill climbing.

Both basic and steepest ascent hill climbing may fail to find a solution. Either algorithm may terminate not by finding a goal state but by getting to a state from which no better states can be generated. This will happen if the program has reached either a local maximum, a plateau, or a ridge.

Hill climbing Disadvantages

Local Maximum: A local maximum is a state that is better than all its neighbors but is not better than some other states farther away.

Plateau: A Plateau is a flat area of the search space in which a whole set of neighboring states have the same value.

Ridge: A ridge is a special kind of local maximum. It is an area of the search space that is higher than surrounding areas and that itself has a slope.

Ways out

- ❖ Backtrack to some earlier node and try going in a different direction.
- ❖ Make a big jump to try to get in a new section.
- ❖ Moving in several directions at once.

Hill climbing is a **local method:** Decides what to do next by looking only at the immediate consequence of its choice.

Global information might be encoded in heuristic functions.

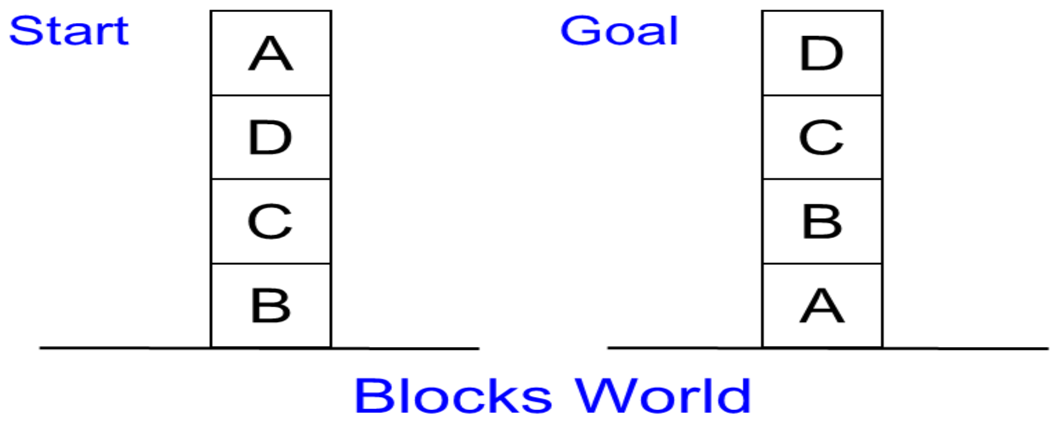
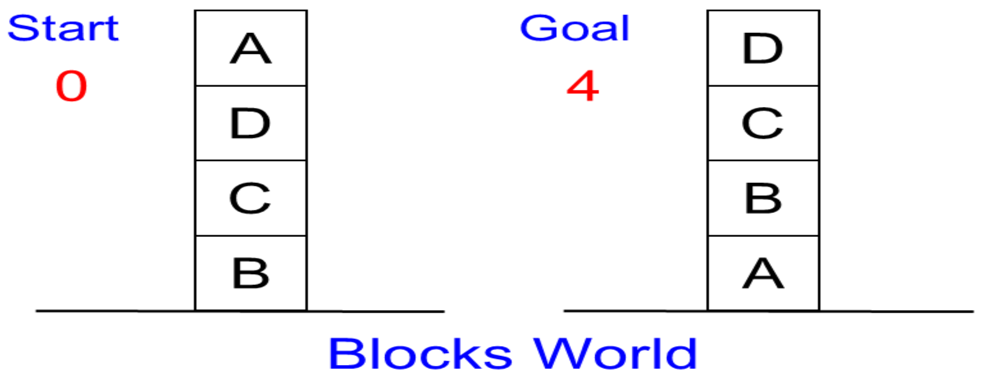
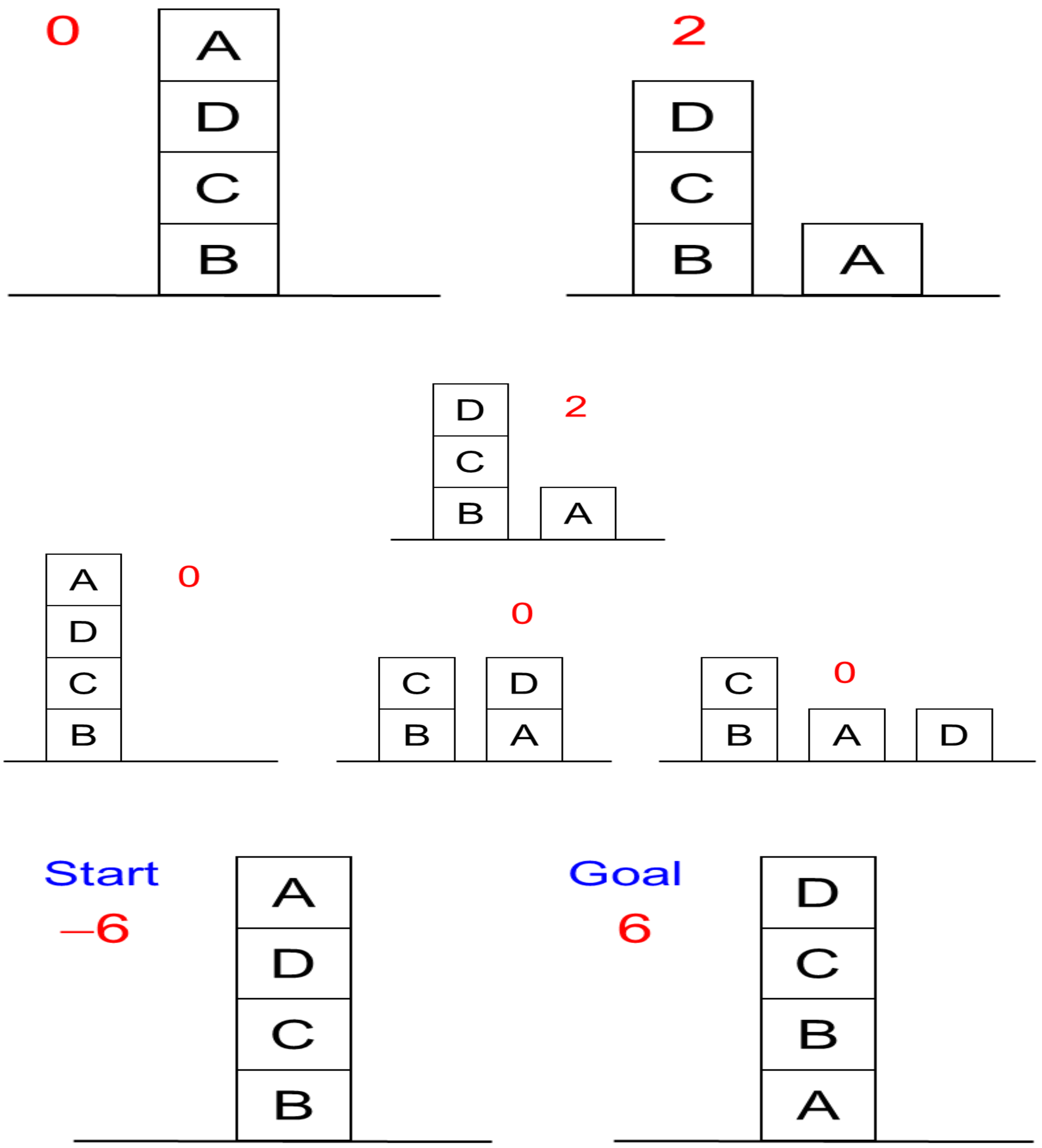


Fig 1.9 Three Possible Moves



Local heuristic:

- ❖ +1 for each block that is resting on the thing it is supposed to be resting on.
- ❖ -1 for each block that is resting on a wrong thing.

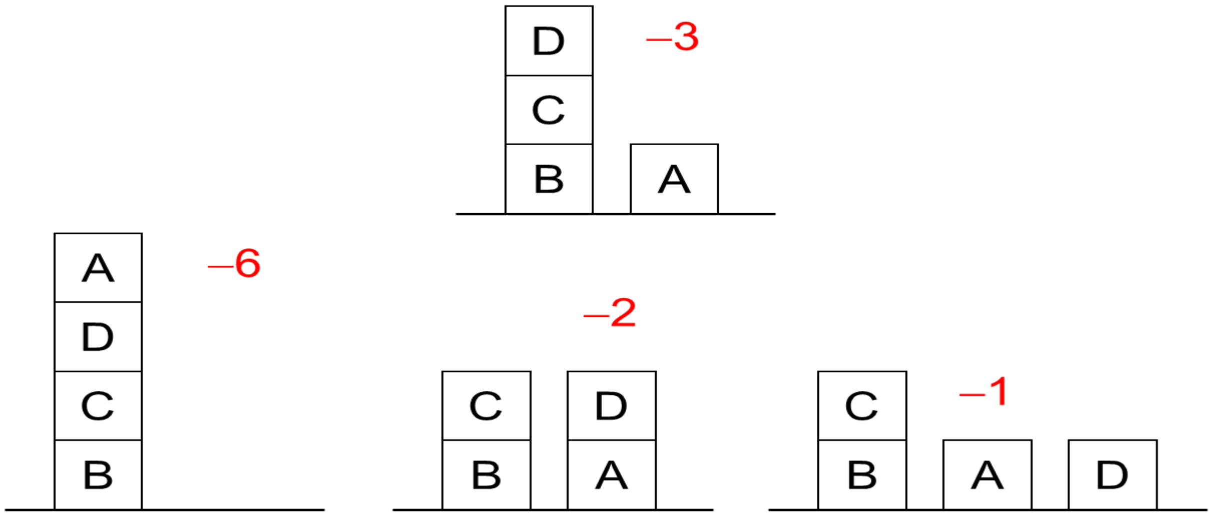


Blocks World

Global heuristic:

For each block that has the correct support structure: +1 to every block in the support structure.

For each block that has a wrong support structure: -1 to every block in the support structure.



Hill climbing conclusion

- ❖ Can be very inefficient in a large, rough problem space.
- ❖ Global heuristic may have to pay for computational complexity.
- ❖ Often useful when combined with other methods, getting it started right in the right general neighbourhood.

3.2.3 Simulated Annealing

The problem of local maxima has been overcome in simulated annealing search. In normal hill climbing search, the movements towards downhill are never made. In such algorithms the search may stuck up to local maximum. Thus this search cannot guarantee complete solutions. In contrast, a random search(or movement) towards successor chosen randomly from the set of successor will be complete, but it will be extremely inefficient. The combination of hill climbing and random search, which yields both efficiency and completeness is called simulated annealing.

The simulated annealing method was originally developed for the physical process of annealing. That is how the name simulated annealing was found and restored. In simulated annealing searching algorithm, instead of picking the best move, a random move is picked. The standard simulated annealing uses term objective function instead of heuristic function. If the move improves the situation it is accepted otherwise the algorithm accepts the move with some probability less than

This probability is

$P= e^{-\Delta E/kT}$

Where -ΔE is positive charge in energy level, t is temperature and k is Boltzman constant. As indicated by the equation the probability decreases with badness of the move (evaluation gets worsened by amount -ΔE). The rate at which -ΔE is cooled is called annealing schedule. The proper annealing schedule is maintained to monitor T.

This process has following differences from hill climbing search:

- ❖ The annealing schedule is maintained.
- ❖ Moves to worse states are also accepted.
- ❖ In addition to current state, the best state record is maintained.

The algorithm of simulated annealing is presented as follows:

Algorithm: “simulated annealing”

1. Evaluate the initial state. Mark it as current state. Till the current state is not a goal state, initialize best state to current state. If the initial state is the best state, return it and quit.
2. Initialize T according to annealing schedule.
3. Repeat the following until a solution is obtained or operators are not left:
    a. Apply yet unapplied operator to produce a new state
    b. For new state compute -ΔE= value of current state – value of new state. If the new state is the goal state then stop, or if it is better than current state, make it as current state and record as best state.
    c. If it is not better than the current state, then make it current state with probability P.
    d. Revise T according to annealing schedule
4. Return best state as answer.

3.3 Best-First Search:

It is a general heuristic based search technique. In best first search, in the graph of problem representation, one evaluation function (which corresponds to heuristic function) is attached with every node. The value of evaluation function may depend upon cost or distance of current node from goal node. The decision of which node to be expanded depends on the value of this evaluation function. The best first can understood from following tree. In the tree, the attached value with nodes indicates utility value. The expansion of nodes according to best first search is illustrated in the following figure.

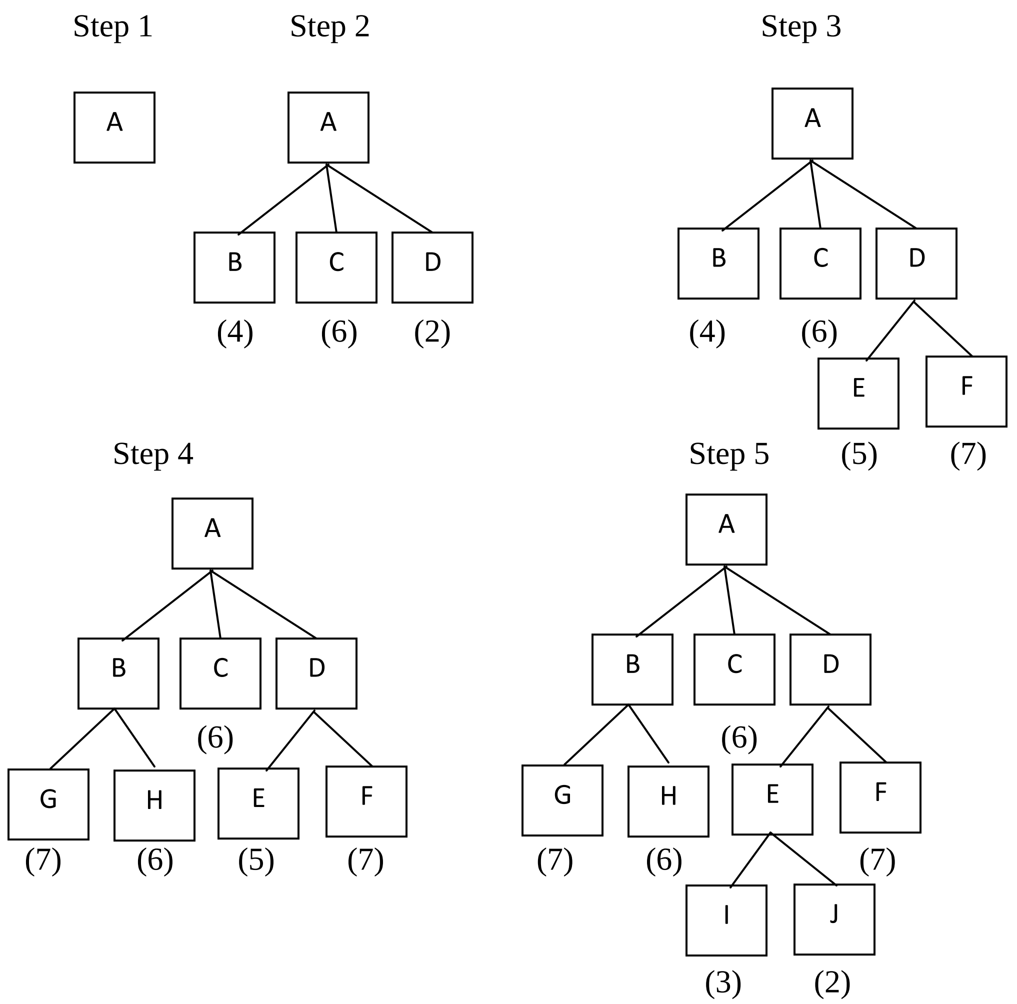


fig:1.10 Tree getting expansion according to best first search

Here, at any step, the most promising node having least value of utility function is chosen for expansion.

In the tree shown above, best first search technique is applied, however it is beneficial sometimes to search a graph instead of tree to avoid the searching of duplicate paths. In the process to do so, searching is done in a directed graph in which each node represents a point in the problem space. This graph is known as OR-graph. Each of the branches of an OR graph represents an alternative problem solving path.

Two lists of nodes are used to implement a graph search procedure discussed above. These are

1. OPEN: these are the nodes that have been generated and have had the heuristic function applied to them but not have been examined yet.
2. CLOSED: these are the nodes that have already been examined. These nodes are kept in the memory if we want to search a graph rather than a tree because whenever a node will be generated, we will have to check whether it has been generated earlier.

The best first search is a way of combining the advantage of both depth first and breath first search. The depth first search is good because it allows a solution to be found without all competing branches have to be expanded. Breadth first search is good because it does not get trapped on dead ends of path. The way of combining this is to follow a single path at a time but switches between paths whenever some competing paths looks more promising than current one does. Hence at each step of best first search process, we select most promising node out of successor nodes that have been generated so far.

The functioning of best first search is summarized in the following steps:

1.It maintains a list open containing just the initial state.

2.Until a goal is found or there are no nodes left in open list do:

   a. Pick the best node from open,

   b. Generate its successor, and for each successor:

      i. Check, and if it has not been generated before evaluate it and add it to open and record its parent.

      ii. If it has been generated before, and new path is better than the previous parent then change the parent.

The algorithm for best first search is given as follows:

Algorithm: Best first search

1.Put the initial node on the list say ‘OPEN’.

2.If (OPEN = empty or OPEN= goal) terminate search, else

3.Remove the first node from open(say node is a)

4.If (a=goal) terminate search with success else

5.Generate all the successor node of ‘a’. Send node ‘a’ to a list called ‘CLOSED’. Find out the value of heuristic function of all nodes. Sort all children generated so far on the basis of their utility value. Select the node of minimum heuristic value for further expansion.

6.Go back to step 2.

The best first search can be implemented using priority queue. There are variations of best first search. Example of these are greedy best first search, A* and recursive best first search.

3.3.2 The A* Algorithm:

The A* algorithm is a specialization of best first search. It most widely known form of best first search. It provides genera guidelines about how to estimate goal distance for general search graph. at each node along a path to the goal node, the A* algorithm generate all successor nodes and computes an estimate of distance (cost) from the start node to a goal node through each of the successors. if then chooses the successor with shortest estimated distance from expansion. It calculates the heuristic function based on distance of current node from the start node and distance of current node to goal node.

The form of heuristic estimation function for A* is defined as follows:

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

where f(n)= evaluation function

g(n)= cost (or distance) of current node from start node.

h(n)= cost of current node from goal node.

In A* algorithm the most promising node is chosen from expansion. The promising node is decided based on the value of heuristic function. Normally the node having lowest value of f (n) is chosen for expansion. We must note that the goodness of a move depends upon the nature of problem, in some problems the node having least value of heuristic function would be most promising node, where in some situation, the node having maximum value of heuristic function is chosen for expansion. A* algorithm maintains two lists. One store the list of open nodes and other maintain the list of already expanded nodes. A* algorithm is an example of optimal search algorithm. A search algorithm is optimal if it has admissible heuristic. An algorithm has admissible heuristic if its heuristic function h(n) never overestimates the cost to reach the goal. Admissible heuristic are always optimistic because in them, the cost of solving the problem is less than what actually is. The A* algorithm works as follows:

A* algorithm:

1.Place the starting node ‘s’ on ‘OPEN’ list.

2.If OPEN is empty, stop and return failure.

3.Remove from OPEN the node ‘n’ that has the smallest value of f*(n). if node ‘n is a goal node, return success and stop otherwise.

4.Expand ‘n’ generating all of its successors ‘n’ and place ‘n’ on CLOSED. For every successor ‘n’ if ‘n’ is not already OPEN , attach a back pointer to ‘n’. compute f*(n) and place it on CLOSED.

5.Each ‘n’ that is already on OPEN or CLOSED should be attached to back pointers which reflect the lowest f*(n) path. If ‘n’ was on CLOSED and its pointer was changed, remove it and place it on OPEN.

6.Return to step 2.

3.4 Constraint Satisfaction

The general problem is to find a solution that satisfies a set of constraints. The heuristics which are used to decide what node to expand next and not to estimate the distance to the goal. Examples of this technique are design problem, labeling graphs robot path planning and crypt arithmetic puzzles.

In constraint satisfaction problems a set of constraints are available. This is the search space. Initial State is the set of constraints given originally in the problem description. A goal state is any state that has been constrained enough. Constraint satisfaction is a two-step process.

1. First constraints are discovered and propagated throughout the system.
2. Then if there is not a solution search begins, a guess is made and added to this constraint. Propagation then occurs with this new constraint.

Algorithm

1. Propogate available constraints:

- ✓ Open all objects that must be assigned values in a complete solution.
- ✓ Repeat until inconsistency or all objects are assigned valid values:

Select an object and strengthen as much as possible the set of constraints that apply to object.

If set of constraints different from previous set then open all objects that share any of these constraints. Remove selected object.

2. If union of constraints discovered above defines a solution return solution.

3. If union of constraints discovered above defines a contradiction return failure.

4. Make a guess in order to proceed. Repeat until a solution is found or all possible solutions exhausted:

- ✓ Select an object with a no assigned value and try to strengthen its constraints.
- ✓ Recursively invoke constraint satisfaction with the current set of constraints plus the selected strengthening constraint.

Crypt arithmetic puzzles are examples of constraint satisfaction problems in which the goal to discover some problem state that satisfies a given set of constraints. some problems of crypt arithmetic are show below

```
    S E N D       D O N A L D         C R O S S
  + M O R E     + G E R A L D      +  R O A D S
  ---------     -------------      -------------
  M O N E Y     R O B E R T        D A N G E R
```

Here each decimal digit is to be assigned to each of the letters in such a way that the answer to the problem is correct. If the same letter occurs more than once it must be assigned the same digit each time. No two different letters may be assigned the same digit.

The puzzle SEND + MORE = MONEY, after solving, will appear like this

```
    S   E   N   D
    9   5   6   7
 +  M   O   R   E
    1   0   8   5
---------------------
 M  O   N   E   Y
 1  0   6   5   2
```

State production and heuristics for crypt arithmetic problem.

Ans.

The heuristics and production rules are specific to the following example:

```
    S   E   N   D
    M   O   R   E
-----------------------
 M  O   N   E   Y
```

Heuristics Rules

1. If sum of two 'n' digit operands yields 'n+1' digit result then the 'n+1'th digit has to be one.
2. Sum of two digits may or may not generate carry.
3. Whatever might be the operands the carry can be either 0 or 1.
4. No two distinct alphabets can have same numeric code.
5. Whenever more than 1 solution appears to be existing, the choice is governed by the fact that no two alphabets can have same number code.

Production Rules:

```
1. x  +  y = yz                 ->  y = 1, z = 0, x = 9
2. ax  +  0y = cz               ->  c = a +1, x + y = z + 10
3. y = x + 1, y + z = x + 10    ->  z = 8 or 9 and prev_carry = 1
4. x  +  y = z + 10             ->  (x, y) = (8, 9) | (7, 9) | (6, 9) |
                                    (5, 9) | (4, 9) | (3, 9) |
                                    (2, 9) | (1, 9) | (7, 8) |
                                    (6, 8) | (5, 8) | (4, 8) |
                                    (3, 8) | (2, 8) | (6, 7) |
                                    (5, 7) | (4 ,7) | (3, 7) |
                                    (5, 6) | (4, 6).
```

To solve the problem SEND + MORE = MONEY using the above rules we proceed

as follows:

Applying rule 1

x = S, Y = M, z = 0

S + M = MO -> M = 1, O = 0 and S = 8 or 9

Applying rule 2

a = E, x = N, y = R, c = N, z = E

EN + OR = NE -> N = E + 1, N + R = E + 10

Also, as E + O + 1 = N and S + M generates cary

S = 9 and not 8

Applying rule 3

y = N, x = E, z = R

N = E + 1, N + R = E + 10 -> R = 8 (as S = 9)

prev_carry = 1

prev_carry = 1 -> D + E = Y + 10

Applying rule 4

x = D, y = E, z = Y

D + E = Y + 10 -> (D, E) = (7, 5)

We conclude that (D, E) = (7, 5) because any other choice always leads to violation of the constraint that - no two distinct alphabates can have same numeric code.

Thus D = 7 and E = 5

Also, as N = E + 1 = 5 + 1 = 6

Also, as D + E = Y + 10 => Y = 7 + 5 - 10 = 2

Thus,

```
     S = 9, E = 5, N = 6, D = 7,
  +  M = 1, O = 0, R = 8, E = 5,
  ___________________________________
   M = 1, O = 0, N = 6, E = 5, Y = 2.
```

3.5 Means – end Analysis

Means-ends analysis allows both backward and forward searching. This means we could solve major parts of a problem first and then return to smaller problems when assembling the final solution.

The means-ends analysis algorithm can be said as follows:

1.Until the goal is reached or no more procedures are available:

- ✓ Describe the current state the goal state and the differences between the two.
- ✓ Use the difference the describes a procedure that will hopefully get nearer to goal.
- ✓ Use the procedure and update current state

If goal is reached then success otherwise fail.

For using means-ends analysis to solve a given problem, a mixture of the two directions, forward and backward, is appropriate. Such a mixed strategy solves the major parts of a problem first and then goes back and solves the small problems that arise by putting the big pieces together. The means-end analysis process detects the differences between the current state and goal state. Once such difference is isolated an operator that can reduce the difference has to be found. The operator may or may not be applied to the current state. So a sub problem is set up of getting to a state in which this operator can be applied.

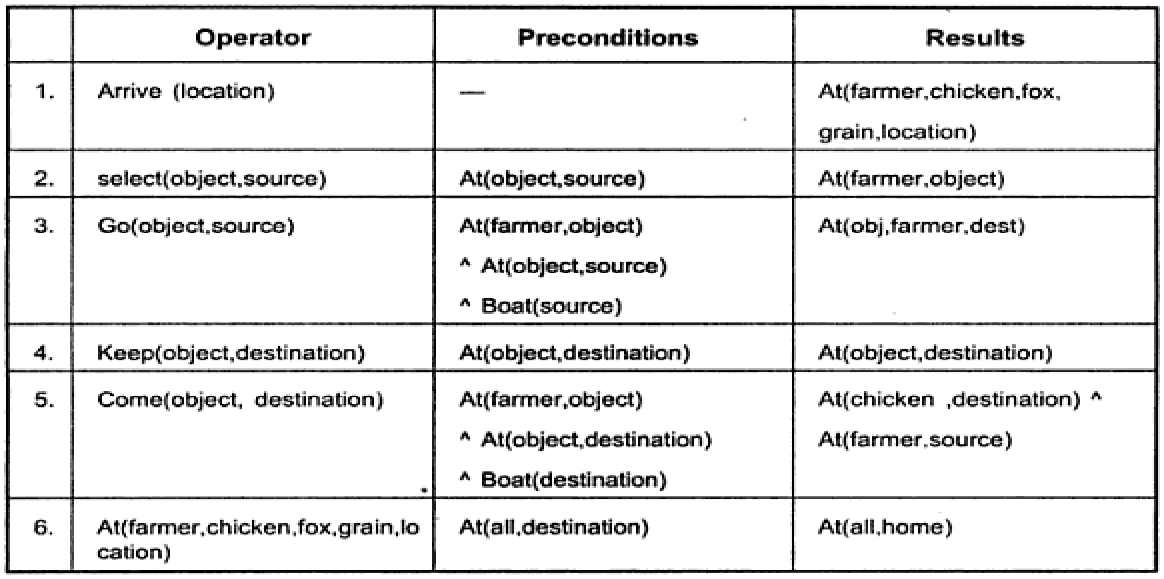
In operator sub goaling backward chaining and forward chaining is used in which first the operators are selected and then sub goals are set up to establish the preconditions of the operators. If the operator does not produce the goal state we want, then we have second sub problem of getting from the state it does produce to the goal. The two sub problems could be easier to solve than the original problem, if the difference was chosen correctly and if the operator applied is really effective at reducing the difference. The means-end analysis process can then be applied recursively.

This method depends on a set of rues that can transform one problem state into another. These rules are usually not represented with complete state descriptions on each side. Instead they are represented as a left side that describes the conditions that must be met for the rules to be applicable and a right side that describes those aspects of the problem state that will be changed by the application of the rule. A separate data structure called a difference table which uses the rules by the differences that they can be used to reduce.

Example Solve the following problem using means-ends analysis

A farmer wants to cross the river along with fox, chicken and grain. He can take only one along with him at a time. If fox and chicken are left alone the fox may eat the chicken. If chicken and grain are left alone the chicken may eat the grain. Give the necessary solution.

Solution:



	Operator	Preconditions	Results
1.	Arrive (location)	—	At(farmer,chicken,fox, grain,location)
2.	select(object,source)	At(object,source)	At(farmer,object)
3.	Go(object,source)	At(farmer,object) ^ At(object,source) ^ Boat(source)	At(obj,farmer,dest)
4.	Keep(object,destination)	At(object,destination)	At(object,destination)
5.	Come(object, destination)	At(farmer,object) ^ At(object,destination) ^ Boat(destination)	At(chicken ,destination) ^ At(farmer,source)
6.	At(farmer,chicken,fox,grain,location)	At(all,destination)	At(all,home)

Difference table.

	Actions	Arrive	Select	GO	Keep	Come
1.	Arrive at the location	*				
2.	Take Chicken		*			
3.	Travel by boat to destination			*		
4.	Leave chicken at destination				*	
5.	Come back at source					*
6.	Take fox	*				
7.	Travel by boat to destination		*			
8.	Keep fox at destination and take chicken back to source.			*	*	
9.	Keep chicken at source and take grain to destination.		*	*		
10.	Keep grain to destination			*		

PART-A

1. What is AI?
2. What are the task domains of artificial intelligence?
3. List the properties of knowledge?
4. What is an AI technique?
5. What are the steps to build a system that solves a problem?
6. What is a state space?
7. Explain the process operationalization?
8. How to define the problem as a state space search?
9. What does the production system consists of?
10. What are the requirements of a good control strategy?
11. What is chronological backtracking?
12. Give the advantages of depth-first search?
13. Give the advantages of breadth-first search?
14. What is combinatorial explosion?
15. Give an example of a heuristic that is useful for combinatorial problems?
16. What is heuristic?
17. Define heuristic function?
18. What is the purpose of heuristic function?
19. Write down the various problem characteristics?
20. What is certain outcome and uncertain outcome problem with examples?
21. What are the classes of problems with respect to solution steps with eg?
22. Illustrate the difference between any-path and best problem with examples?
23. What are the types of problems with respect to task interaction with a person?
24. What is propose and refine?
25. What is monotonic production system?
26. What is nonmonotonic production system?
27. What is commutative and partially commutative production system?

28. What are weak methods?

29. Write generate and test algorithm?
30. How is hill climbing different from generate and test?
31. When hill climbing fails to find a solution?
32. What is local maximum?
33. What is ridge?
34. What is plateau?
35. List the ways to overcome hill climbing problems?
36. Differentiate steepest accent from basic hill climbing?
37. Differentiate simple hill climbing from simulated annealing?
38. What are the components essential to select an annealing schedule?
39. What is best first search process?
40. State 'Graceful decay of admissibility'
41. What is an agenda?
42. What is the limitation of problem reduction algorithm?
43. What is constraint satisfaction?
44. What is operator subgoaling?
45. Define playing chess.

PART-B

1. Explain briefly the various problem characteristics?
2. Illustrate how to define a problem as a state space search with an example?
3. Discuss the merits and demerits of depth-first and breadth-first search with the algorithm?
4. What are the problems encountered during hill climbing and what are the ways available to deal with these problems?
5. Explain the process of simulated annealing with example?
6. Write A* algorithm and discuss briefly the various observations about algorithm?
7. Discuss AO* algorithm in detail?
8. Write in detail about the constraint satisfaction procedure with example?

9. Explain how the steepest accent hill climbing works?

10. Write in detail about the mean end analysis procedure with example?