Module-2
Problem Solving using AI Agent

Contents

- 2.1 Problem Solving by intelligent agents;
- 2.2 Problem Formulation,
- 2.3 State Space Representation,
- 2.4 Search Problems: Playing Chess, 8-Puzzle,Water Jug Problem,
- 2.5 Problem Reduction,
- 2.6 Production Systems,
- 2.7 8-Puzzle Production System.
- **Chapter 2: 2.1, 2.2, 2.3, 2.4:2.4.1, 2.4.2, 2.4.4, 2.5, 2.6, 2.7)**

2.1Problem Solving by intelligent agents

- An agent is anything which can be viewed as perceiving its environment through sensors and acting upon that environment through actuators.
- A human agent has ears, eyes and other organs for sensors and hands, legs, mouth etc, for actuators.
- Problem-solving agent is one kind of goal based agent that decides what to do by finding sequences of actions that leads to desirable states.
  - It first formulates a goal and a problem.

Reference: Problem solving agent follows this four phase problem solving process:

Goal Formulation: .

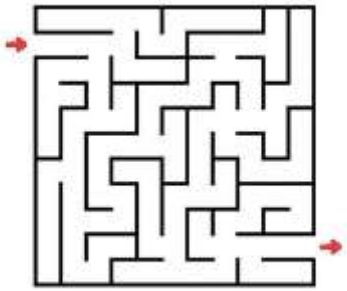
Imagine an AI agent designed to navigate a maze to reach a specific destination.

- **Goal Formulation:** The goal formulation involves defining what the AI agent ultimately wants to achieve. In this case, the goal is to reach the destination in the maze.
- **Goal Statement:** Reach the destination (D) in the maze.

Problem Formulation:

Problem formulation involves breaking down the overall goal into a specific set of actions or steps that the AI agent needs to take to achieve the goal.

- **State Space:** The possible states of the agent within the maze (locations it can be).
- **Actions:** The possible actions the agent can take (move up, move down, move left, move right).
- **Initial State:** The starting point of the agent in the maze.
- **Transition Model:** How the agent's state changes when it takes an action.
- **Goal Test:** A condition to check if the agent has reached the destination.



Search: Identify possible solutions (Sequence of Actions to reach the goal state)

After the Goal and Problem Formulation, the agent simulates sequences of actions and has to look for a sequence of actions that reaches the goal.

This process is called **search,** and the sequence is called a **solution**.

The agent might have to simulate multiple sequences that do not reach the goal, but eventually, it will find a solution, or it will find that no solution is possible.

A search algorithm takes a problem as input and outputs a sequence of actions.

Execution: After the search phase, the agent can now execute the actions that are recommended by the search algorithm, one at a time. This final stage is known as the execution phase.

Three Parameters of an Intelligent Systems **knowledge base**, **operators**, and **control strategy**

- In the context of AI agents, knowledge base, operators, and control strategy play crucial roles in problem-solving. Let's break down each concept:
- **Knowledge Base:**
  - **Definition:** The knowledge base is the repository of information that the AI agent possesses about the world. It includes facts, rules, and any relevant information that the agent uses to make decisions.
  - **Example:** In a medical diagnosis system, the knowledge base could contain information about symptoms, diseases, and rules for associating symptoms with possible medical conditions.
- **Operators:**
  - **Definition:** Operators are the actions or functions that the AI agent can perform to transition from one state to another. They represent the allowable moves or transformations that the agent can make in the problem-solving process.
  - **Example:** In a robot navigation system, operators could be movements such as "move forward," "turn left," or "turn right."

- **Control Strategy:**
  - **Definition:** The control strategy refers to the plan or approach that the AI agent follows to select actions and make decisions. It dictates how the agent uses its knowledge base and operators to achieve its goals.
  - **Example:** In a chess-playing AI, the control strategy could involve searching through possible moves, evaluating board positions, and selecting the move that leads to a favorable outcome.

The aim of any search technique is the application of an appropriate sequence of **operators** to an initial state to achieve the goal

The objective can be achieved in two ways :

(1) Forward reasoning—It refers to the application of operators to those structure in the knowledge base that describe the task domain in order to produce a modified state. Such a method is also referred to as bottom-up or data-driven reasoning.

(2) **Backward reasoning**—It breaks down the goal (problem) statement into subgoals (problems) which are easier to solve and whose solutions are sufficient to solve the original problem.

A problem solving agent or system uses either forward or backward reasoning. Each of its operator works to produce a new state in the knowledge base which is said to represent problems in a state-space. Operators

Example : In a medical diagnosis system, forward reasoning involves starting with the patient's symptoms (known facts) and using medical knowledge and rules to infer potential diseases or conditions.

Symptoms → Diseases

Example Backword Reasoning: The MYCIN expert system is a real life example of how backward

Diseases → symptoms

- A- If you are running, B - you will high rate of heartbeat
- Final state is B
- Is XYZ is Running then XYZ will get High rate of hearbeat
- If B is True -→ B is having High rate of heartbeat, then he must be running

2.2 Problem Formulation

Before a solution can be found, the important point is that the problem must be very precisely defined. That is :

- What is the explicit goal of the problem statement ?
- What are the criteria for success ?
- What is the initial state or start state ?
- Is transformation of situation possible by rules, operations etc. ?

A problem can be defined formally by four components :
(i) The initial state (starting state).
(ii) State space : It involves a description of all the possible actions available, i.e., the set of all states reachable from the initial state. The state space forms a graph in which the nodes are states and the arcs between nodes are actions.
(iii) Goal test : It determines whether a given state is a goal state.
(iv) Path cost : It is a function that assigns a numeric cost to each path.

A solution to a problem is a path from the initial state to a goal state. The quality of the solution is measured by the path cost function and an optimal solution has the lowest path cost among all solutions.

Problem formulation is done in terms of the initial state, state space, goal test and path cost.

To formulize a problem the following steps are needed :

(i) Define the problem precisely, giving the specifications of what the initial situation(s) and the final situation(s) will be.

(ii) Analyze the problem because a few important features can have an immense impact on the suitability of different techniques available for solving the problem.

(iit) Represent the knowledge completely which is necessary to solve problem in a given domain.

(iv) Choose the best technique(s) and apply it (them) to the particular problem.

Depending upon the control strategy to be used the performance of problem solving procedure can be improved or degraded.

It is thus clear that to a create a program for solving a problem simply the formal description of the problem has to be generated using the knowledge about the given problem.

This is called operationalization.

Any problem can be solved by the following series of steps:

1. Define a state space which contains all the possible configurations of the relevant objects and even some impossible ones.

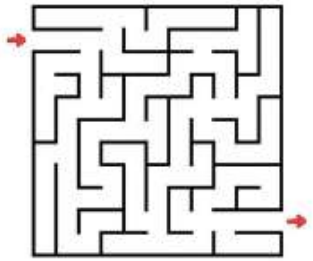
2. Specify one or more states within that space which would describe possible situations from which the problem solving process may start. These states are called the initial states.

3. Specify one or more states which would be acceptable as solutions to the problem. These states are called goal states.

4. Specify a set of rules which describe the actions (operators) available and a control strategy to decide the order of application of these rules.

The problem can be solved by collecting all knowledge of the problem and using an appropriate control strategy; move through the problem space until a path from an initial state to goal state is found.

This is the process of search which is fundamental to the problem solving process.



The most common methods of problem solving representation in AI are:

1. State space representation and
2. Problem reduction.

State space representation

Suppose we are asked to prepare a cup of tea; what should be done? All the ingredients such as tea leaves, milk powder, sugar, kettle, heating arrangement etc must be made available. The following steps are needed:

1. Boil required quantity of water.
2. Take some of the boiled water in a cup, add necessary amount of tea leaves to make decoction (extraction of flavour by boiling).
3. Add milk powder to the some boiling water to make milk.
4. Mix decoction and milk.
5. Add sugar to your taste.
6. Tea is prepared.

Now think a bit about what has exactly happened. We started with the ingredients (*initial state*), followed a sequence of steps (called *states*) and at last had a cup of tea (*goal state*). We added only needed amount of tea leaves, milk powder and sugar (*operators*). Fig. (1) shows the sequence of operations.

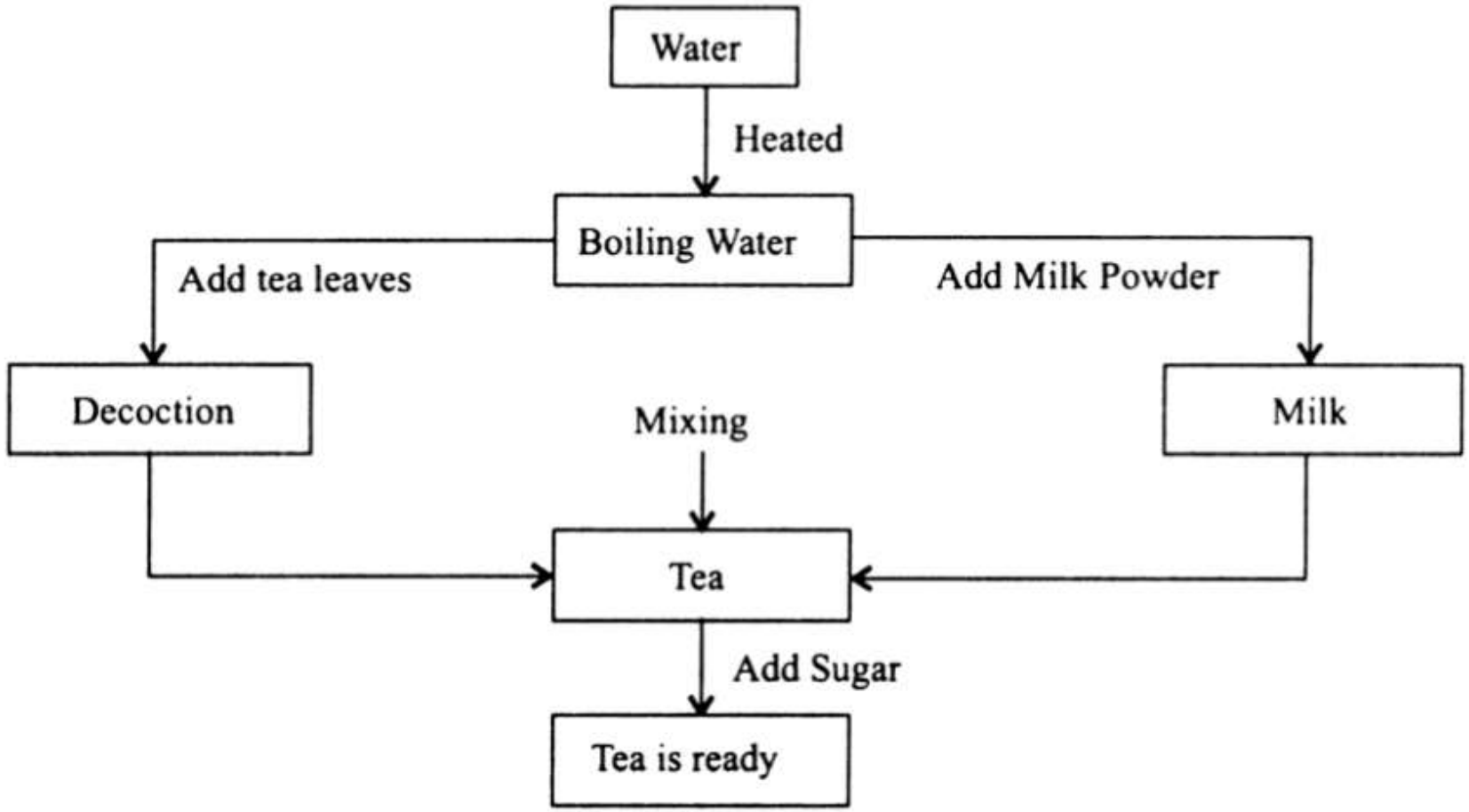


Fig. (1). State space representation for tea making.

Three Basic Components of an Implicit State Space Representation

1. A description with which to label a *start node*. This description is some data structure modeling the initial state of the (agent's) environment.
2. Functions which transform a state description representing one state of the environment into one that represents the state that results after an action. These functions are usually called *operators*. They are models of the effects of actions. When an operator is applied to a node it generates one of the node's successors.
3. A *goal condition*, which can be either True-False valued function on a state description or a list of actual instances of state descriptions that correspond to goal states.

2.4 Search Problems: Playing Chess

2.4.1 Playing Chess—An Example of State Space Search

To build a program that could *play chess* the following needs to be done selecting:

1. The starting position of the chess board.
2. The rules which define the legal moves.
3. Goal of winning the game.

This is a simple game wherein it is easy to provide a formal and complete problem description. Each board configuration can be thought of as representing a different state of the game. A change of state occurs when one of the players moves a piece. A goal state is any of the possible board configurations corresponding to a checkmate. Let us define these requirements related with the problem in a little more detail.

Starting Position **:** It can be described as 8 × 8 array where each position contains a symbol standing for the appropriate piece in the chess opening position.

Goal **:** It can be any board position in which the opponent doesn't have any legal move left and his king is under attack.

Legal moves **:** They provide the way of getting from initial state to goal state; and can be a set of rules consisting of two parts, the left side of which serves as a pattern to be matched against the current board position and the right side of the rule describes the change to be made in the board position after the rule (operator) is made applicable. There are several ways in which these rules can be written. Fig. (2). shows one legal move for chess.

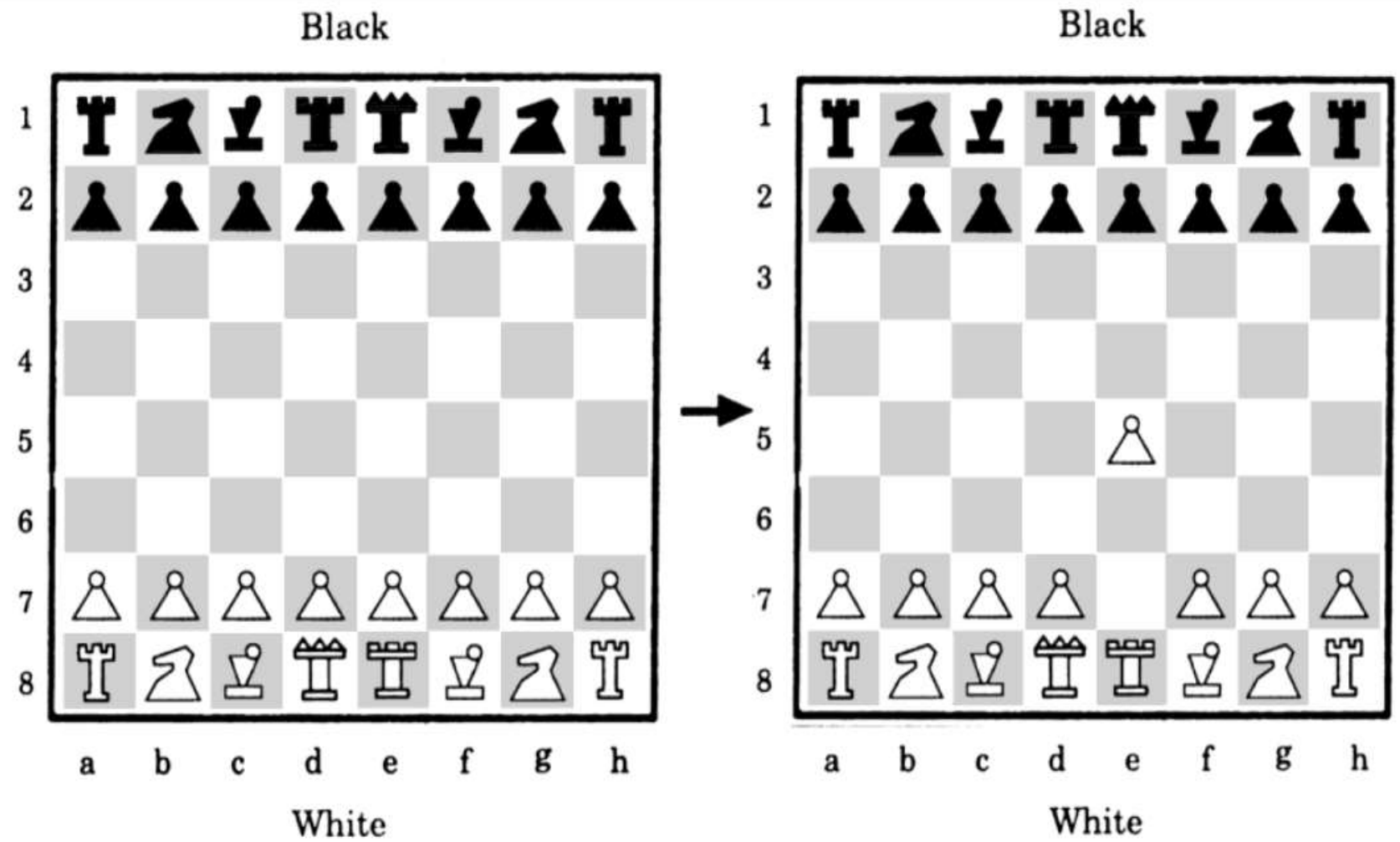


Fig. (2). One legal chess move

An iterative use of the steps enumerated above leads to a state space search. This connects some given situation with some desired situation, through a series of board positions obtained after the application of given operations (legal moves). An operator is nothing but representation of an action. It usually includes information about what must be true in the world before the action can take place and how the world (of the state space) is changed by the action of the operator. If we write the rules as above the problem becomes voluminous.

Once again there can be many ways. A simple one can be wherein the rule in fig. (2) can be rewritten as shown in fig. (3).

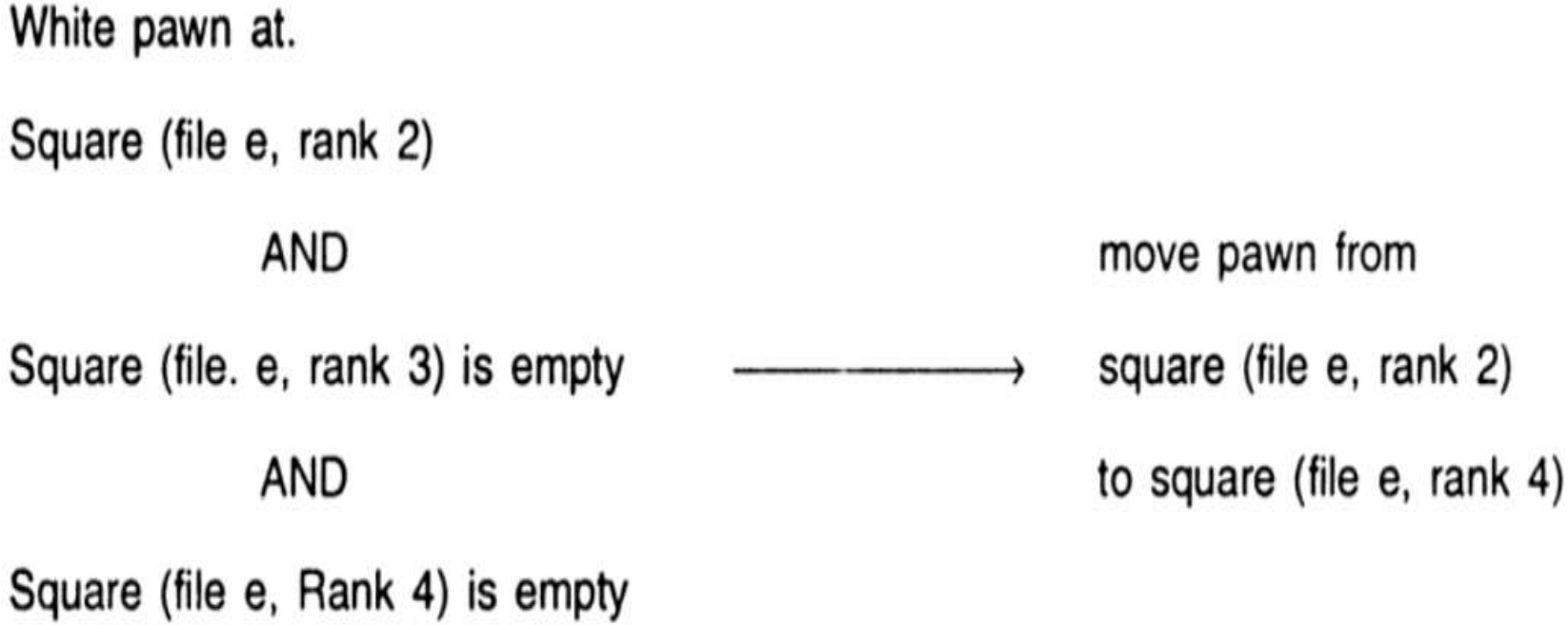


Fig. (3). An alternative to represent chess move in fig. (2).

Conclusion

The problem, *play chess*, has been described as a problem of moving around in a state space; where each state corresponds to a legal position of the board. Starting from initial state and using a set of rules to move from one state to another can give rise to one of a set of final states. The combination of all these states gives rise to what is called *the state space*.

2.4.2 The Eight Tile Puzzle

The eight tile puzzle consists of a 3×3 board which hold eight movable tiles which are numbered 1 to 8. One square is empty, thereby permitting adjacent tile to be shifted. The object is to reach a specified goal state such as shown in fig. (4).

3	8	1
6	2	5
	4	9

Start state

1	2	3
8		4
7	6	5

Goal state

Fig. (4). Start and goal configuration of the eight-tile puzzle.

The states of the eight tile puzzle are the different permutations of the tiles within the frame. The standard formulation is as follows :

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.

Goal **:** There are many goal configurations possible. One such goal state is shown in figure.

Legal moves (or states) **:** They generate the legal states that result from trying the four actions (blank moves *Left*, *Right*, *Up* or *Down*).

An optimal or good solution is one that maps an initial arrangement of tiles to the goal configuration with the smallest number of moves.

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

The search space for the eight-tile puzzle problem may be depicted as the tree shown in fig (5).

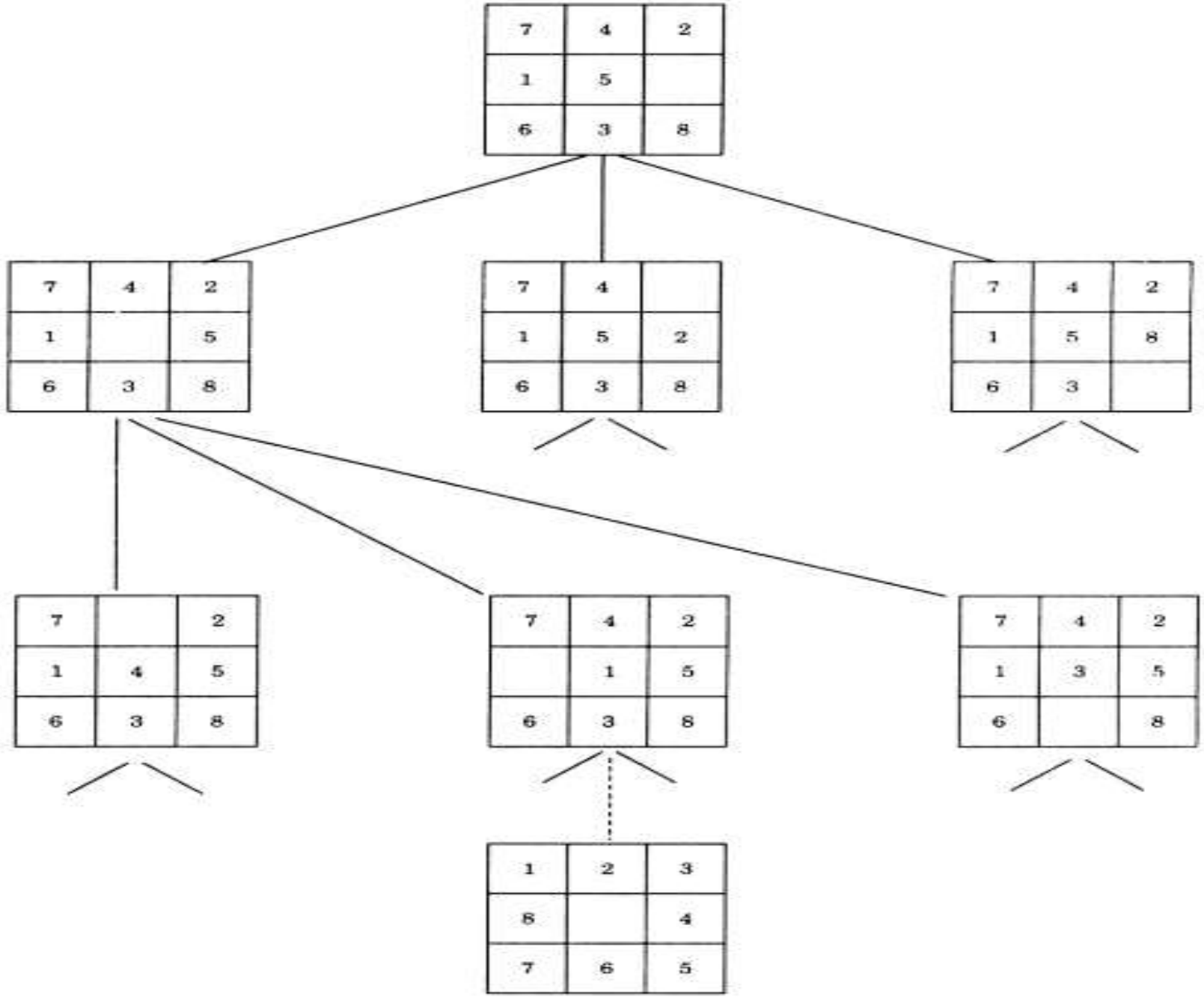
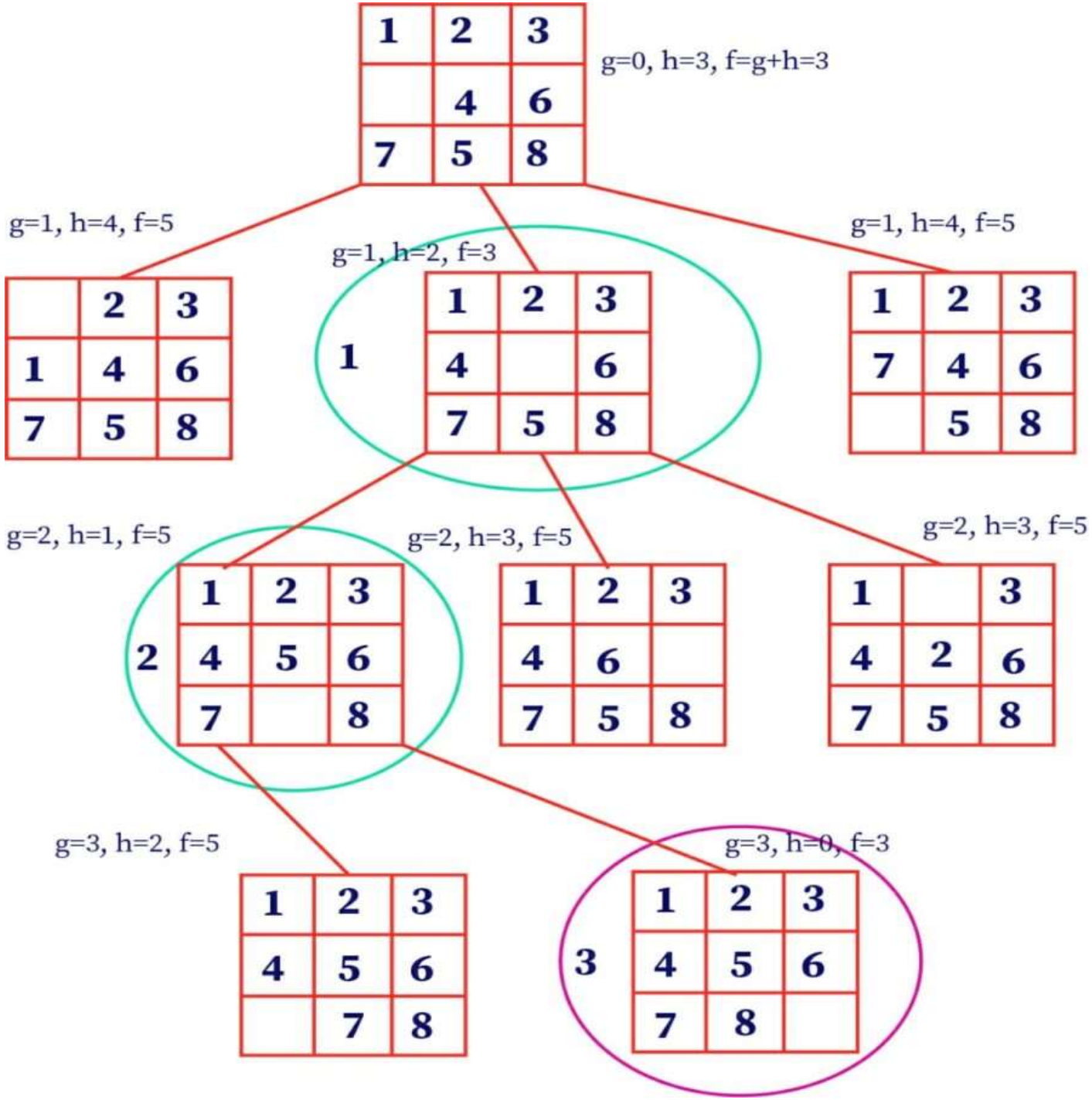
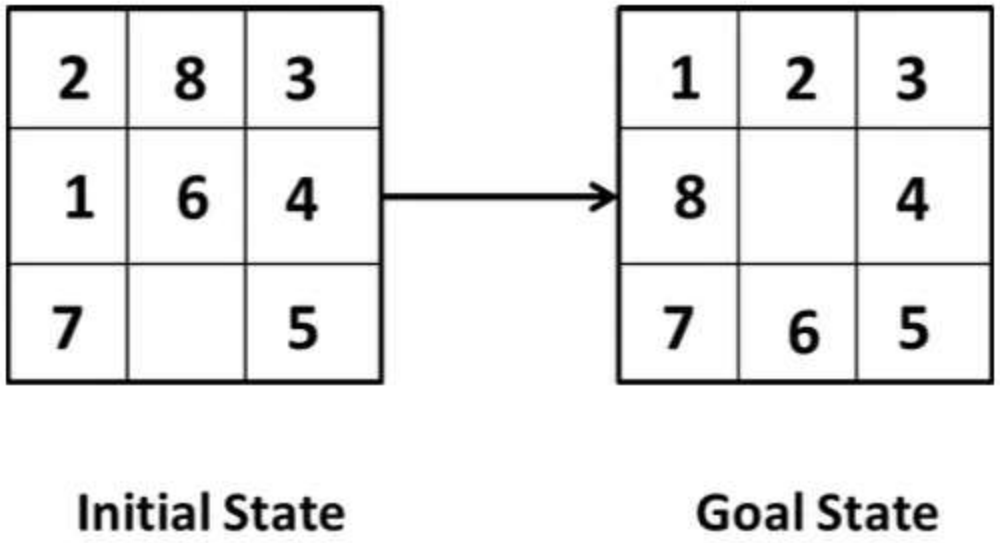


Fig. (5). A tree diagram for the eight-tile puzzle.

- f-score = h-score + g-score
- h-score: estimated distance from the current node to the goal node.
- g-score: total nodes traversed from the start node to the current node.





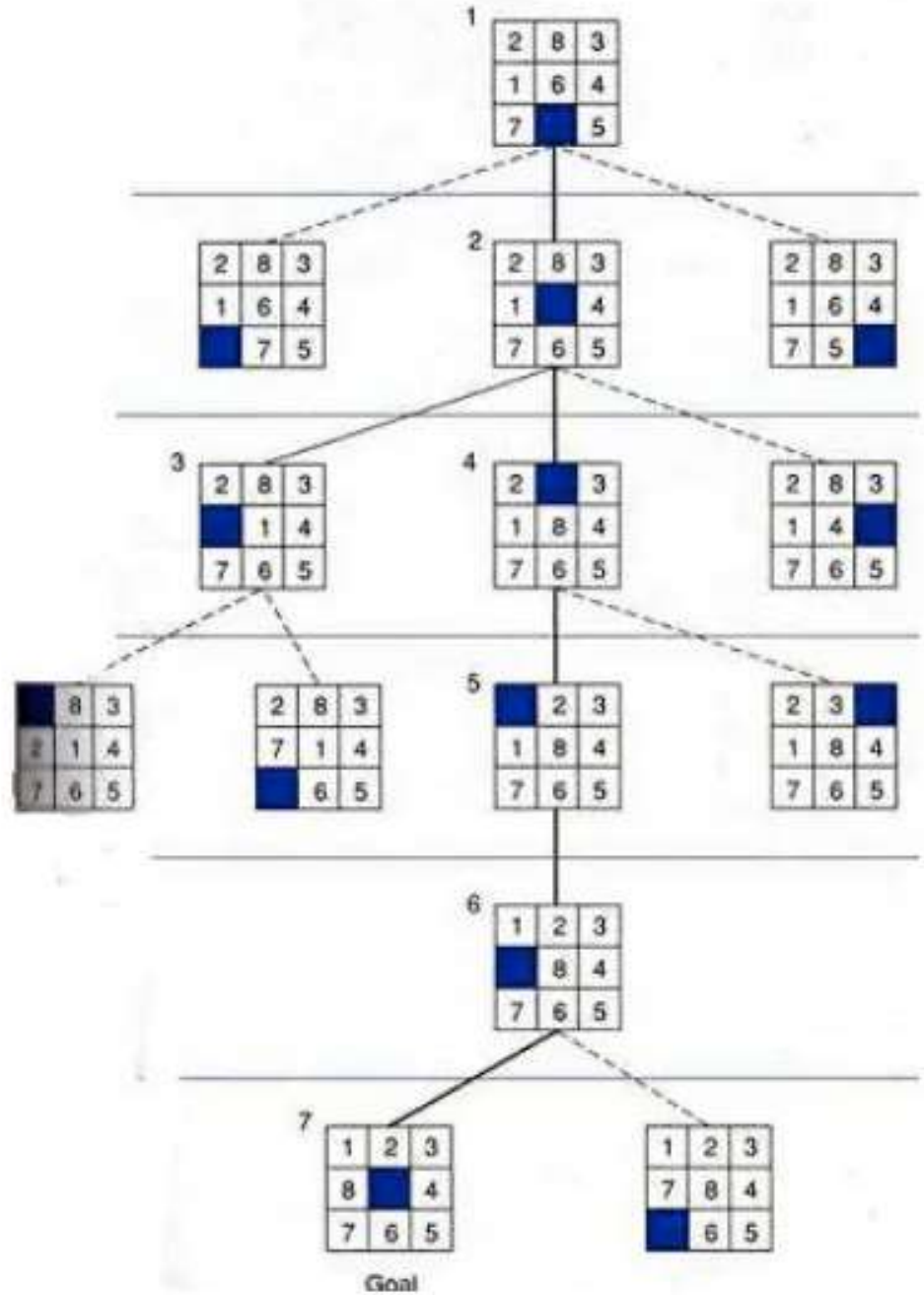


Figure 1: Solution of 8 Puzzle problem

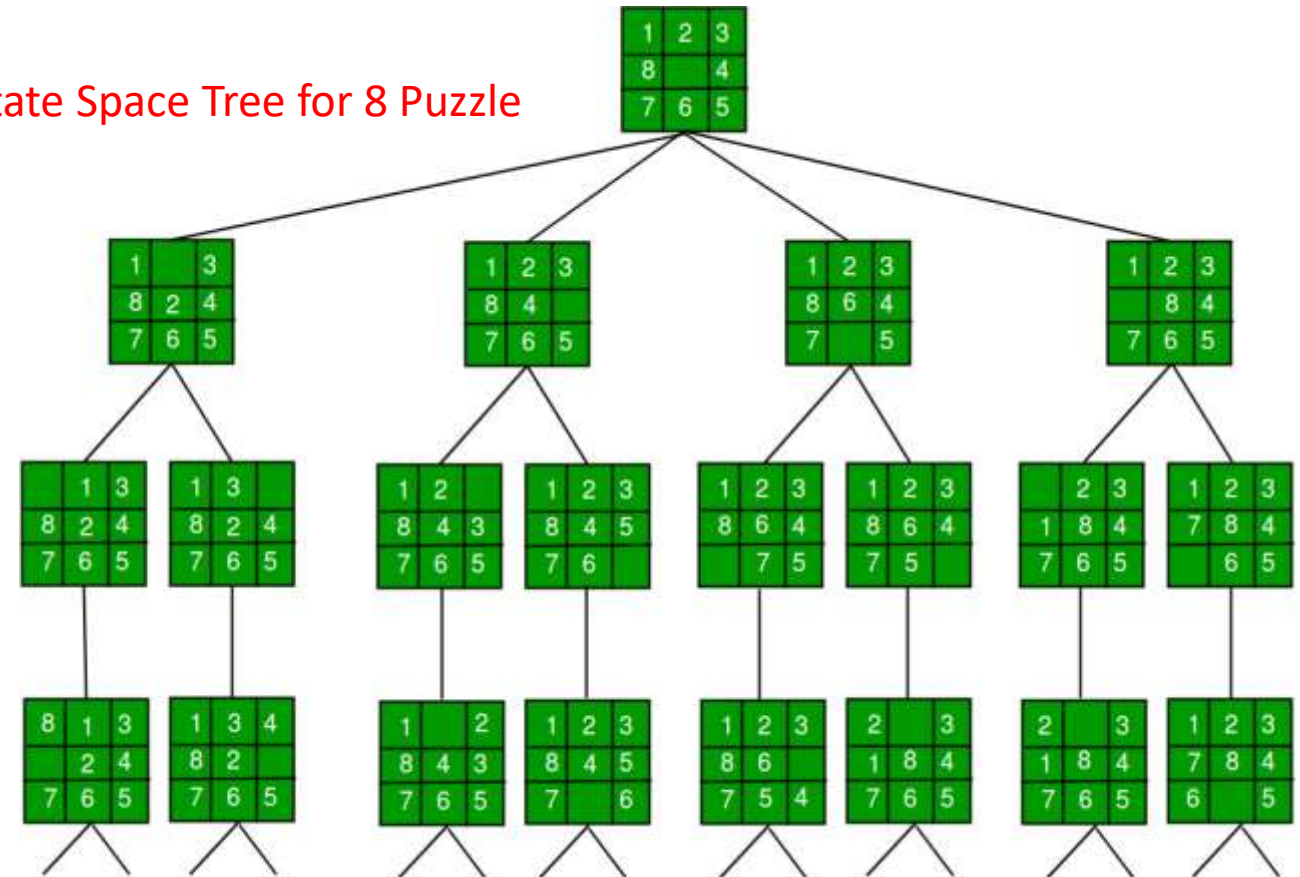
Initial configuration

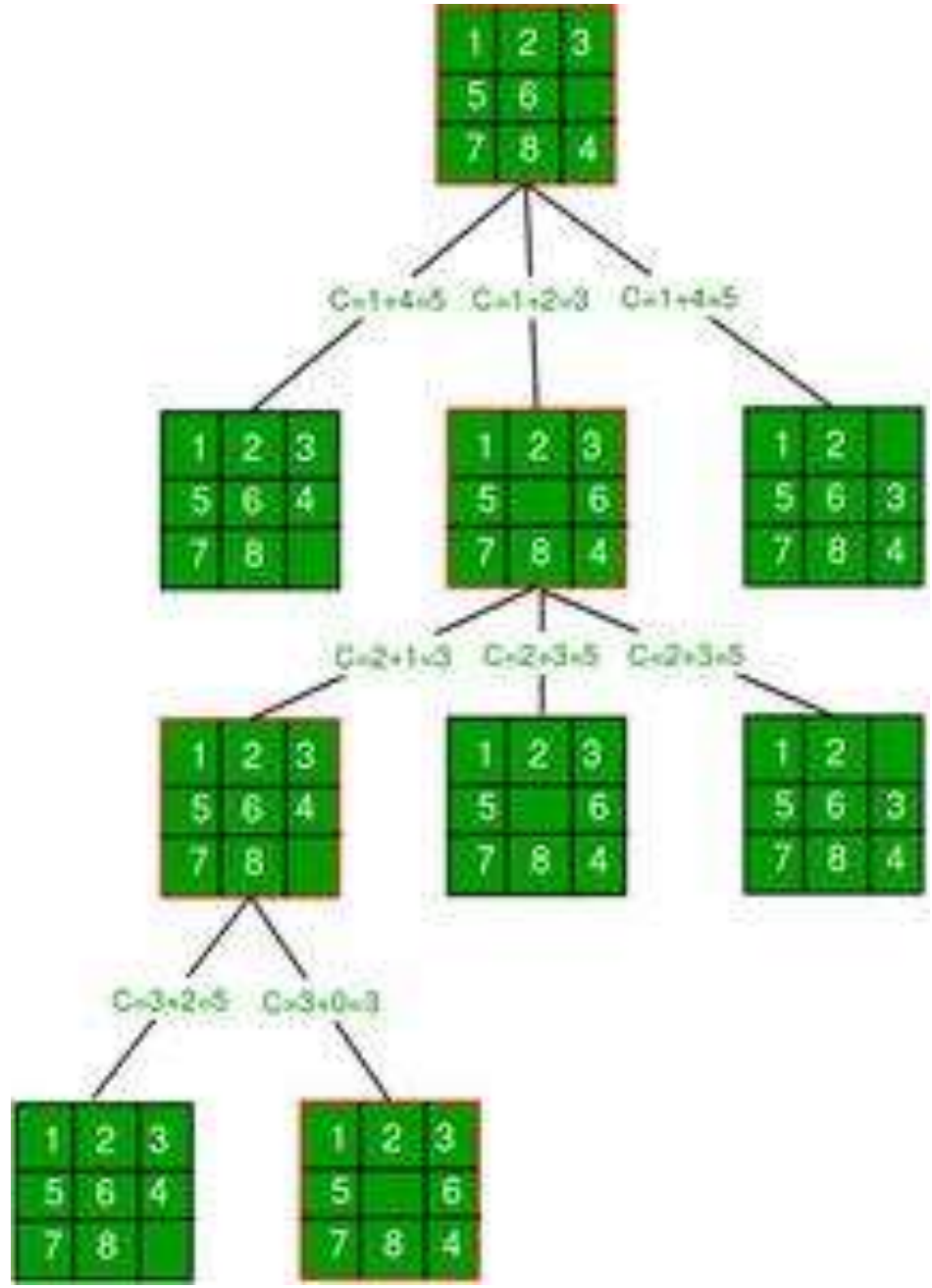
1	2	3
5	6	
7	8	4

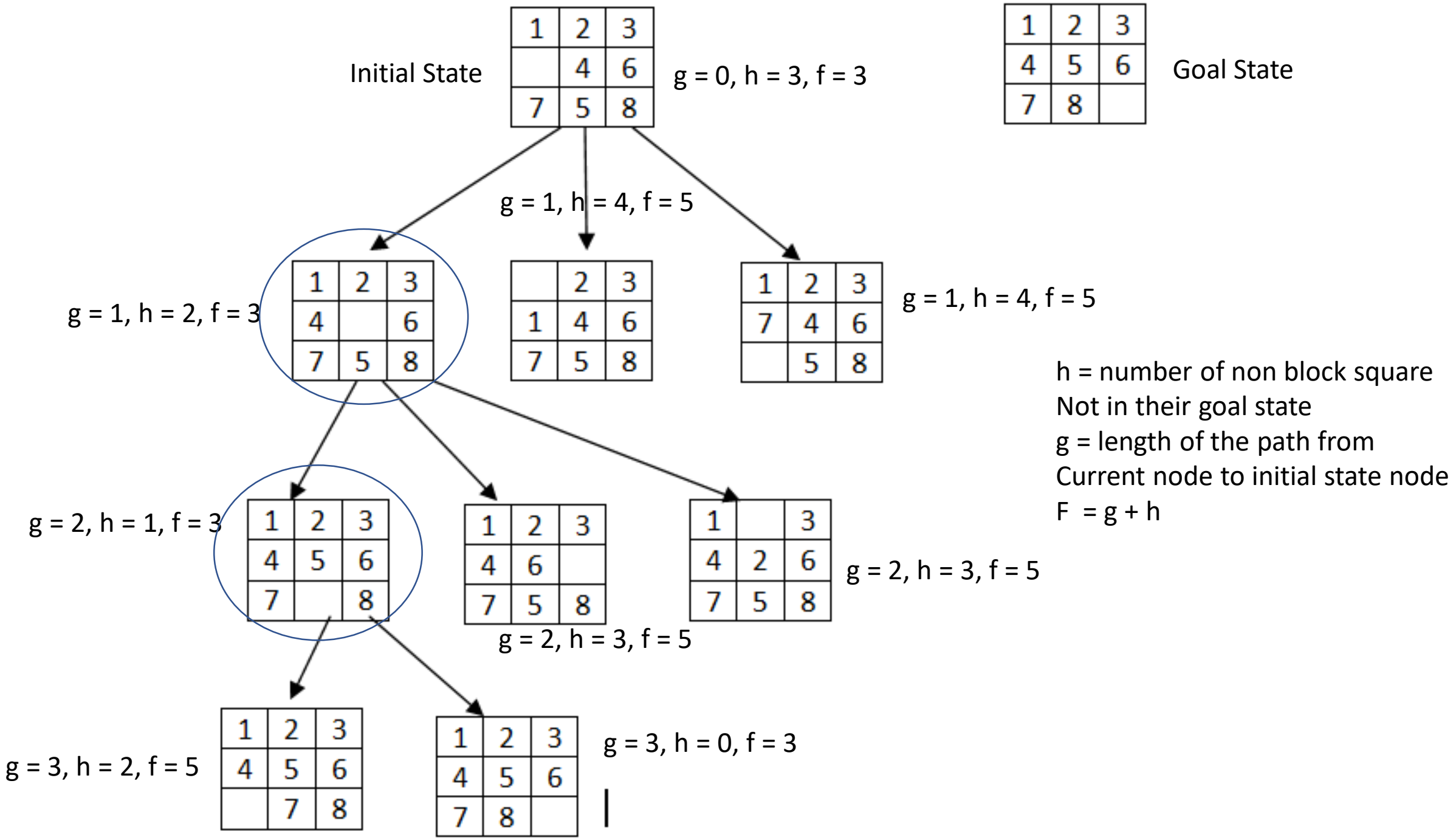
Final configuration

1	2	3
5	8	6
	7	4

State Space Tree for 8 Puzzle






Initial State
1 2 3
4 6
7 5 8
g = 0, h = 3, f = 3
1 2 3
4 5 6
7 8
Goal State
g = 1, h = 4, f = 5
g = 1, h = 2, f = 3
1 2 3
4 6
7 5 8
2 3
1 4 6
7 5 8
1 2 3
7 4 6
5 8
g = 1, h = 4, f = 5
h = number of non block square
Not in their goal state
g = length of the path from
Current node to initial state node
F = g + h
g = 2, h = 1, f = 3
1 2 3
4 5 6
7 8
1 2 3
4 6
7 5 8
g = 2, h = 3, f = 5
1 3
4 2 6
7 5 8
g = 2, h = 3, f = 5
g = 3, h = 2, f = 5
1 2 3
4 5 6
7 8
1 2 3
4 5 6
7 8
g = 3, h = 0, f = 3

2.4.4 Water Jug Problem

In order to show the generality of state space representation let us take another problem, water jug problem, which is stated as :

We are given two jugs, a four-gallon one and three-gallon one. Neither has any measuring markers on it. There is a pump which can be used to fill the jugs with water. How can we get exactly two gallons of water into the four-gallon jug ?

The state space for this problem can be described as the set of ordered pairs of integers (x, y) such that X = 0, 1, 2, 3 or 4 and Y = 0, 1, 2, or 3; X is the number of gallons of water in the four-gallon jug and Y the quantity of water in the three-gallon jug.

The start state is (0, 0) and the goal state is (2, n) for any value of n, as the problem does not specify how many gallons need to be filled in the three-gallon jug (0, 1, 2, 3). So the problem has *one* initial state and *many* goal state. Some problems may have many initial states and *one* or *many* goal states.

The operators to be used to solve the problem can be described as shown in fig (8).

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 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 $X + Y \geq 4$ and $Y > 0 \rightarrow (4, Y - (4-X))$	Pour water from the 3-gallon jug into the 4-gallon jug until the 4-gallon jug is full.
8.	(X, Y) if $X + Y \geq 3$ and $X > 0 \rightarrow (X - (3-Y), 3))$	Pour water from the 4-gallon jug into the 3-gallon jug until the 3-gallon jug is full.
9.	(X, Y) if $X + Y \leq 4$ and $Y > 0 \rightarrow (X + Y, 0)$	Pour all the water from the 3-gallon jug into the 4-gallon jug.
10.	(X, Y) if $X + Y \leq 3$ and $X > 0 \rightarrow (0, X + Y)$	Pour all the water from the 4-gallon jug into the 3-gallon jug.
11.	$(0, 2) \rightarrow (2, 0)$	Pour the 2-gallon water from 3-gallon jug into the 4-gallon jug.
12.	$(2, Y) \rightarrow (0, Y)$	Empty the 2-gallon in the 4-gallon jug on the ground.

Fig. (8). Production rules (operators) for the water jug problem.

Issues in Water Jug Problem

Rules needs to be clear and should have a condition on left side

There are several sequences of operators which will solve the problem, two such sequences are shown in fig. (9).

Water in four-gallon jug (X)	Water in three-gallon jug (Y)	Rule applied
0	0	
0	3	2
3	0	9
3	3	2
4	2	7
0	2	5 or 12
2	0	9 or 11

Fig. (9a). A solution to water jug problem.

Issues in Water Jug Problem

- **Rules 3 and 4 is not used and can be avoided**
- **Rules 12 & 11 is redundant & Can be avoided**

X	Y	Rule applied (Control strategy)
0	0	
4	0	1
1	3	8
1	0	6
0	1	10
4	1	1
2	3	8

Fig. (9b). 2^{nd} solution to water jug problem.

Need of State Space Representation & Actions(Operators) for State transition

Thus the *state space representation* forms the basis of most of the AI methods for problem solving. Its structure corresponds to the structure of problem solving, conforming to the following two steps:

(a) Converts the given situation into some desired situation using permissible operations.

(b) Combines the known operators, each representing a rule defining a single step, in the space.

2.5 Problem reduction

In this method a complex problem is broken down or decomposed into a set of primitive sub-problems. Solutions for these primitive sub-problems can be obtained unless the goals are interactive. The solutions for all the sub-problems collectively give the solution for the complex problem.

Generally speaking, state space search may be good when the solution to a problem is naturally expressed in terms of either a final state or a path from an initial state to a final state. We should be able to define rules for transforming one state into another based on the available actions in the domain. On the other hand, problem reduction is better if it is easy to decompose a problem into independent sub-problems. Of course we have to define rules to do this. It provides a natural explanation of the decision making which allowed to arrive at a solution. Also, it may result in less search than state-space approach.

Production System

- Production Systems are frequently referred as
  - Inferential Systems
  - Rule Based Systems
  - Simply Productions
- The word production in Production Systems denotes
  - Condition-Action Rule

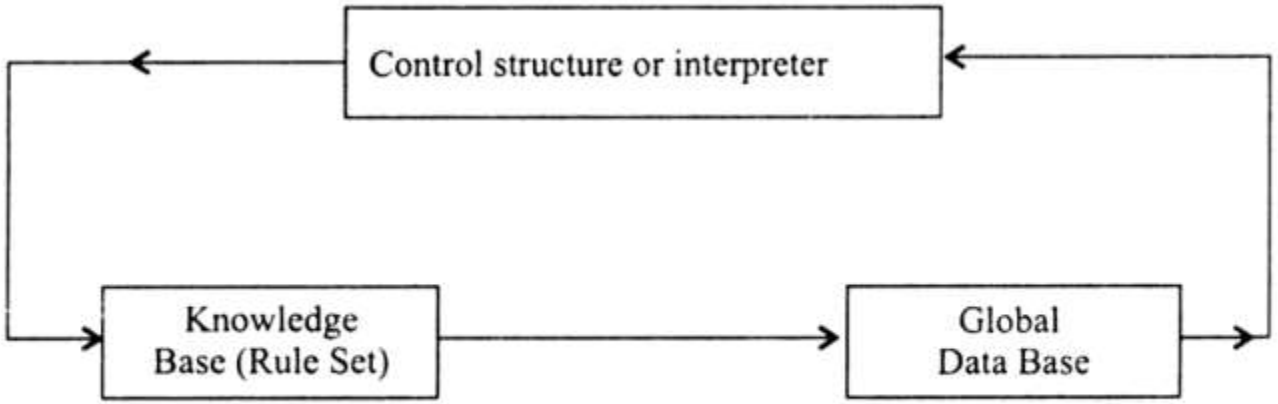


Fig. (10). Components of a production system

Real-World Examples of AI Production Systems in Use

- **Customer Support Chatbots:** AI-powered chatbots in customer support systems use production rules to handle customer inquiries, provide answers, and escalate complex issues to human agents.
- **Fraud Detection Systems:** In financial institutions, AI production systems detect fraudulent activities by analyzing transaction data and applying predefined fraud detection rules.
- **Medical Diagnosis:** AI production systems are used in healthcare for medical diagnosis. They analyze patient symptoms, medical history, and test results to suggest possible diagnoses and treatment options.
- **Traffic Management:** Smart traffic management systems use AI production systems to optimize traffic flow by adjusting signal timings based on real-time traffic conditions and predefined rules.

Production System Components

- **Global Database:** The global database serves as the system's memory, storing facts, data, and knowledge relevant to its operation. It is a repository that production rules can access to make informed decisions and draw conclusions.
- **Production Rules:** Production rules form the core logic of the system. They are a set of guidelines that the system follows while making decisions. These regulations outline the system's reaction to various inputs and circumstances.
- **Control System:** The control system manages the execution of production rules. It determines the sequence in which rules are applied, ensuring efficient processing and optimizing the system's performance.

Production System

- The process of Solving the Problem can usefully be modeled as a Production System.
- If one adopts a system with production rules and a rule-interpreter
  - Then the system is known as a Production System
- In production Systems,
  - The working memory of the system models – Human short-term memory
    - Remembering the current situation and engaging a action
    - During Traffic signal – Passing on to which lane based on the current traffic intensity of the lane.
  - Productions are part of Long-term Memory
    - During Traffic signal – Passing on to which lane based on the current traffic intensity of the lance without considering usually high traffic on the chosen lane.
  - Productions whose conditions are satisfied can add or delete facts in the working memory
    - Choosing a Lane based on the current low traffic intensity but unfortunately sudden occurrence of high traffic in the chosen lanes would cause update the fact that….even chosen lane would be having traffic during certain timeline.

Production Systems

. Rules of production systems

1. *A powerful knowledge representation scheme*—Production systems represent not only knowledge but also action.
2. *The bridge connecting A.I. research to expert systems*—Production system provide a language in which the representation of expert knowledge (in a domain) is very natural.
3. They provide a heuristic model for human behaviour.
4. They are good way to model the strong data-driven nature of intelligent action: As new inputs enter the data base the behaviour of the system changes.
5. New rules can easily be added to account for new situations without disturbing the rest of the system. This is important since no A.I. program is ever completed. Although sometimes confusion arises from interaction among rules it is often less severe than the corresponding complications of modifying the straight-line code.

Production Systems

- Transforming a problem statement into these components of the production system is often Called Problem Representation.
- There are many ways to represent problem
  - Selecting a good representation is one of the important arts involve in applying AI Technique to practical problems

2. Architecture of production system

A typical architecture of a production system is shown fig. (10). It consists of three main components:

(*i*) Rule base
(*ii*) Global data base
(*iii*) Control structure

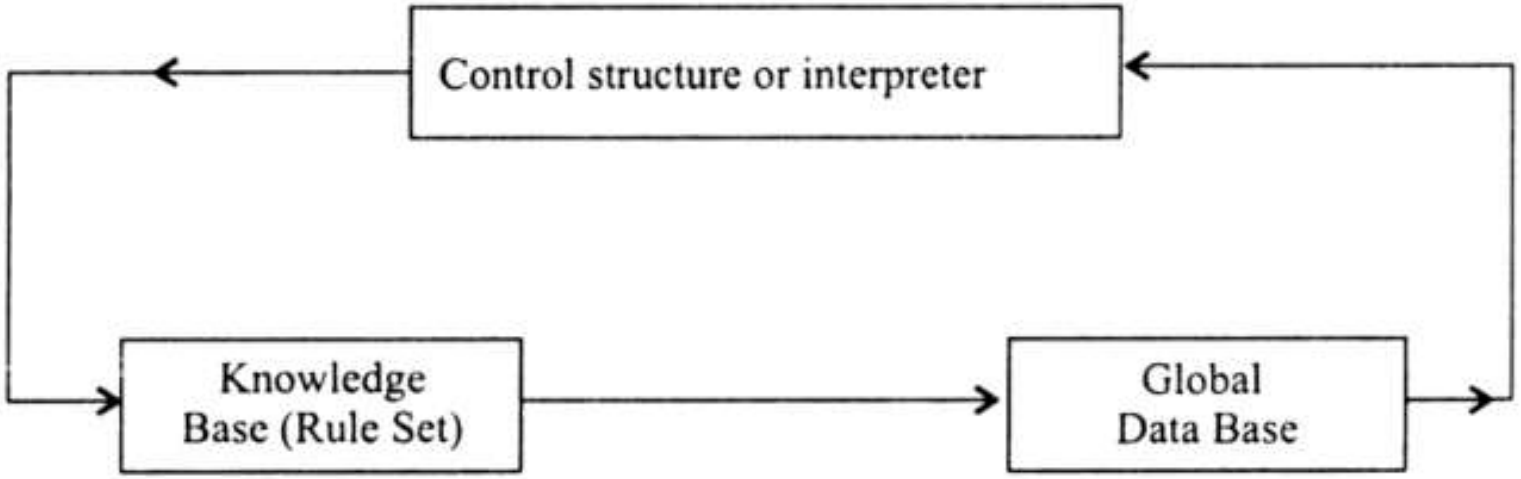


Fig. (10). Components of a production system

Production Systems

- A good Problem solution requires
  - Efficient Control Strategy
  - Good representation for problem states, moves and Goal conditions
- Representation of a problem has a great influence on the effort needed to solve it.
  - Should prefer representation with small state spaces.

Production Systems

- Production rules are conditional
  - IF – THEN Branches.
  - These rules apply on the global database.
- Each rule may have a precondition.
- If the precondition of the rule is satisfied then only the corresponding rule is applied to a state.
- The application of the rule makes change in the state of the problem under conditions.
- Knowledge base in Production systems is decoded in a declarative forms which comprises of a set of rules which are of the form
  - Situation → Action

Production Systems

- Problem situation – Action rules
  - Often called Production Rules or IF-THEN rules
  - A system which uses this form of knowledge is called production system
- Idea of this form of representation is derived from the observation that human experts think in IF-THEN patterns while solving a problem
- Example – if your Car does not start, your reasoning process might be of the following form:

IF the starter does not work
AND the head lights are dim
AND the ignition and oil pressure light don't come on
THEN the battery might be flat

Production System

IF the starter does not work
AND the head lights are dim
AND the ignition and oil pressure light don't come on
THEN the battery might be flat

It may be noted that we cannot be absolutely sure that our conclusion is right, because the battery connections might be poor. So you could state the conclusion in the following form:

THEN the battery might be flat OR the battery connection poor.

It would also be possible to use a *certainty* factor (probability factor) and say:

THEN there is a strong evidence (0.9) that the battery is flat. The factor can range from 0.0 (certainly wrong) to 1.0 (certainly right).

This kind of reasoning is called *inexact reasoning*. The above example has illustrated that human reasoning often consists of IF-THEN rules (with certainty factor attached). The rules have the following general form:

IF
< antecedent 1 >
< antecedent 2 >
< antecedent n >

THEN
<Consequence 1> | (with certainty C_1)
<Consequence 2> | (with certainty C_2)
<Consequence n> | (with certainty C_n)

Production Systems

[II] Global Database. The global database is the central data structure used by the production system. Depending upon the application this database (not to be confused with data base of DBMS) may be as simple as a small matrix of numbers or as complex as a large relational indexed file structure. The global database can be accessed by all the rules (no part of the database is local to any of the rules, in particular). Each rule has a precondition which is either satisfied or not by the global database. If the precondition is satisfied, the rule can be applied. Application of the rules changes the data base as a result it is a dynamic structure continually changing as a result of operation of production rules, so it is also called working memory or short-term memory.

Production Systems

[III] Control Structure or Control Strategy. It is essentially an interpreter program to control the order in which the production rules are fixed and resolve conflicts, if more than one rule becomes applicable simultaneously. This structure repreatedly applies rules to the data base until a description of the goal state is produced. The process of identifying the rules which are tried until some sequence of them is found which produces a data base satisfying the termination condition (goal) is called SEARCH PROCESS.

Efficient control strategies require enough knowledge about the problem being solved so that the rule (or sequence of rules) selected has a good chance of being the most appropriate. Two major kinds of control strategies are: irrevocable and tentative. In an irrevocable control strategy an acceptable rule is selected and applied irrevocably without provision for reconsideration later. In a tentative control strategy, an applicable rule is selected (arbitrarily or based upon some good reason), the rule is applied, with a provision to return later to this point in the computation to apply some other rule.

Characteristics of Control Strategy

- It should cause a motion from one state to another state in state space
- It should be systematic in the way productions rules are applied
  - Sequential or forwarded or backward chaining etc
  - Uninfomred Search
    - BFS
    - DFS
  - Informed Search
    - A Star Algorithm (F Score = H Score + G Score)

BFS – approach in control strategy

Algorithm Breadth-First Search

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

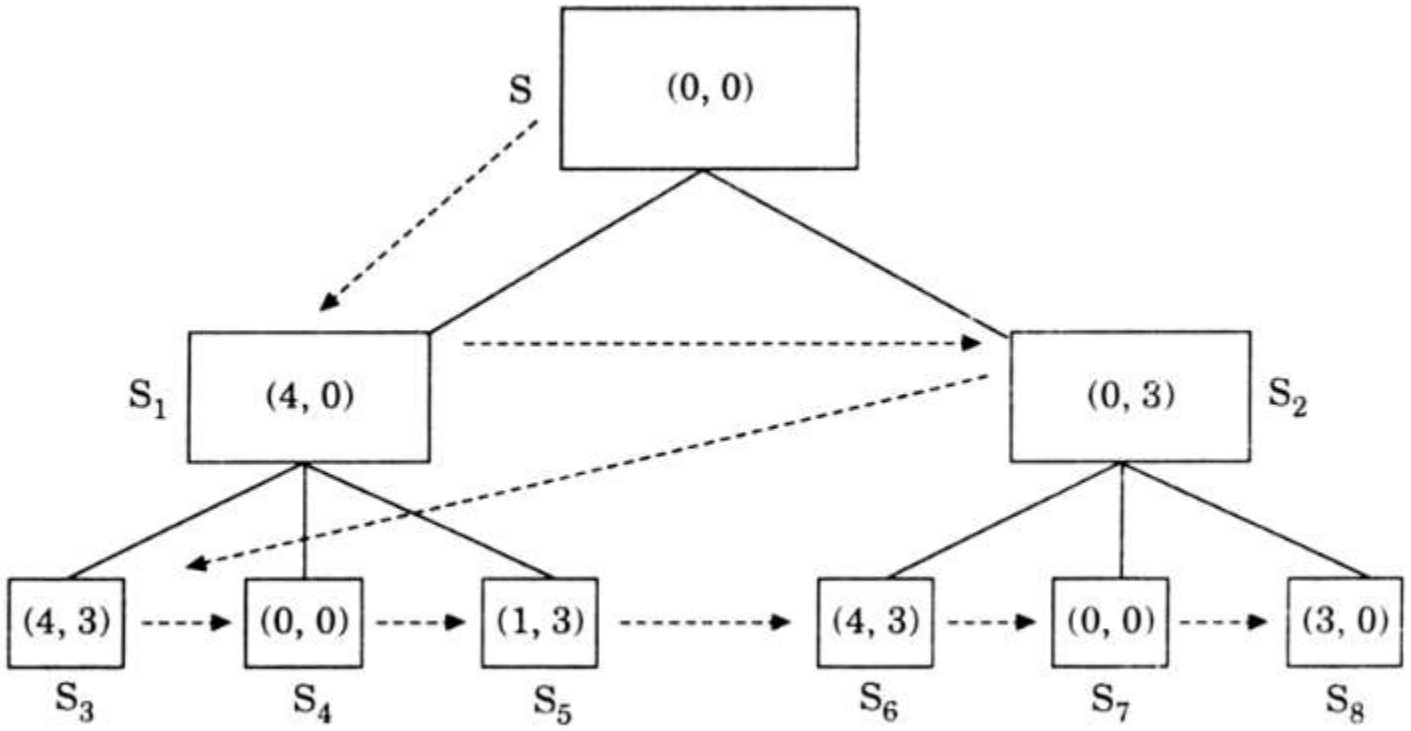


Fig. (12). Search tree after two levels of search.

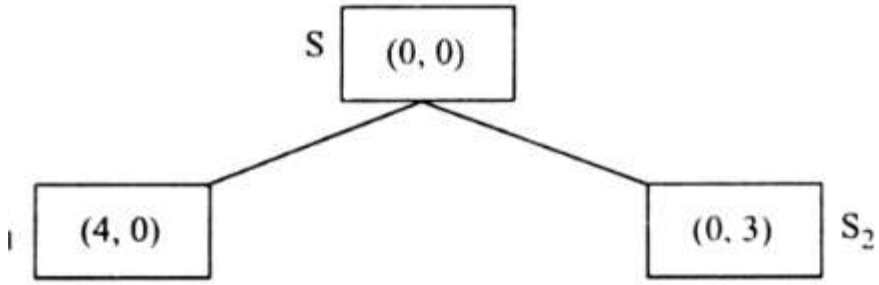


Fig. (11). One level of breadth-first search tree.

DFS – approach in control strategy

- DFS stops
  - If it reaches dead end
  - Gets previous states
  - Reached threshold level of nodes (Depths)

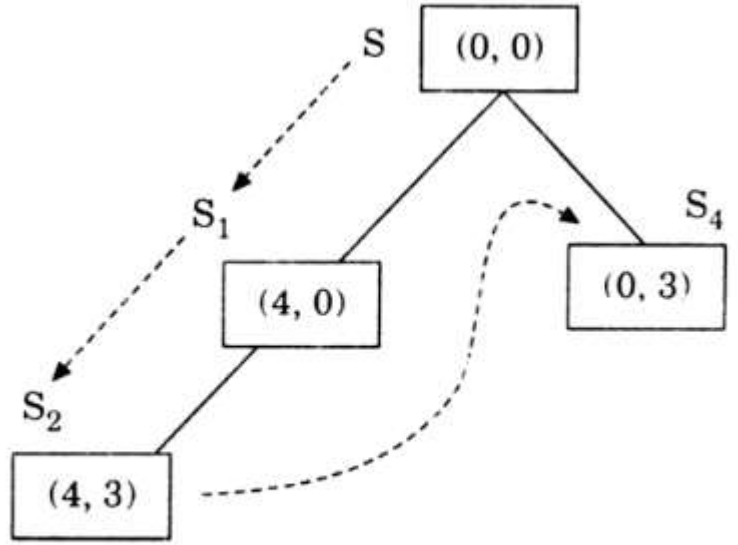


Fig. (13). Search tree using depth-first search.

Algorithm of Depth-First Search

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