1,1 Intelligent system

AI is a combination of computer science psychology and philosophy. In brief we can define AI the study of making computers do things intelligently programs must have capability aI program must have capability and characteristics of Intelligence such as learning reasoning interfacing and receiving and comprehending information.

Understanding of AI

Understanding of related terms intelligence, knowledge, reasoning,cognition learning and a number of other computer related terms. Shows rest on complex problems for which general principles do not help much though there are few useful general principles.

The first view of aI is about duplicating what the human brain does

The second view is that a is about duplicating what the human brain should do what is doing things logically or rationally

ELIZA

The main characteristics of briefly mention here:

Simulation of Intelligence

Quality of response

Coherence

Semantics

ELIZA is a program which makes conversation with user in English just like siri in in iPhones.

Categorization of intelligent systems

In order to design intelligent systems it is important to categorize these systems there are four possible categories of such systems.

1. **Systems that think like humans:** It is an area of cognitive science. the Stimuli are converted into mental representation and cognitive process manipulate this representation to build new representation that is used to generate actions.
2. Systems that act like human required that the overall behavior of the system should be human like which could be achieved by observation. Turing test is an example
3. **System which think rationally:** Thinking rationally or logically logical formula and theories used for synthesizing outcomes.
4. **System that acts rationally:** In the final category of intelligent systems where by rational behavior we mean doing the right things.

Components of AI program

Any AI program should have knowledge base and navigational capability which contains control strategy and inference mechanism

Knowledge base: Program should be learning in nature and update its knowledge accordingly which consists of fats and rules and has the following characteristics:

It is a voluminous in nature and requires proper structuring.
It may be incomplete and imprecise.
It may be dynamic and keep on changing.

Control strategy: It determines which rule to be applied. Some heuristics or thumb rules based on problem domain may be used.

Inference mechanism: It requires search through knowledge base and derive new knowledge using the existing knowledge with the help of inference rule

1. 2 Foundation of AI

Commonly used techniques and theories are rule-based fuzzy Logic neural networks decision theory statistics probability theory genetic algorithms etc. Since AI is interdisciplinary in nature foundation of AI in various fields such as

Mathematics : AI systems used formal logical methods and Boolean logic analysis of limits to what can be computed probability theory uncertainty that forms the basis for most modern approaches to AI fuzzy logic etc

Neuroscience: Science of medicine helps in studying the functions of brain researchers are working to know as how to how to have a mechanical brain which systems will require computing remapping and interconnections large extent.

Control theory: Machines can modify their behavior in response to the environment.

Linguistics: speech Demonstrates so much of human intelligence

Sub areas of AI

Each one of these fields is an area of research in a itself.
Knowledge representation models
Theorem proving mechanism
Game playing methodologies
Common sense reasoning dealing with uncertainty and decision making
Learning models , influence techniques, pattern recognition searching and matching.
Logic (fuzzy, imperial, model)
Planning and scheduling
Natural language understanding, speech reorganization, and understanding spoken utterances
Computer vision
Model for intelligent tutoring systems
Robotic

Data mining
Expert problem solving
Neural networks, aI tools,
web agents

Applications

Find applications in almost all areas of real life applications. Broadly speaking, businesses, engineering, medicine, education and manufacturing are the main areas.

Business: Financial strategies, give advice

Engineering: Check design, offer suggestions to create new product, expert systems for all Engineering applications

Manufacturing: Assembly, inspection and maintenance

Medicine: Monitoring ,diagnosing and prescribing

Education: Teaching

Fraud detection
Object identification
Space shuttle scheduling
Information retrieval

2 Problem solving: state space search and control strategies

Problem solving is a method. Of driving solutions Steps beginning from initial description of the problem with the desired solution. It has been conveniently one of the focus area of artificial intelligence. And can be categorized as a systematic search using a range of possible steps to achieve some predefined solutions. In my the problems are frequently modeled as you state space problem where state space is a set of all possible States from the start to Gold state. A general purpose method is applicable. To a wide variety of problems where as special purpose method is a tailor made for Ethical problems and of an exploit very specific features of the problem. The order of applications of the rule to the current state is called control strategy.

2.1

General problem solving:

Production system :

Production system is one of the formulation that helps AI problem programs to do such process more conveniently instead fees problem. The system comprises of start(Initial.) States And both States. Of the problem along with one or more database consisting of suitable and necessary information for the particular task. Knowledge representation scheme of are used to structure information in the database. Production rule has left side that determines the applicability of the rule and right side that describes the action to be performed with the rule is applied.

Usefulness of election system in describing search following are other advantages of it as a Formalism in artificial intelligence.

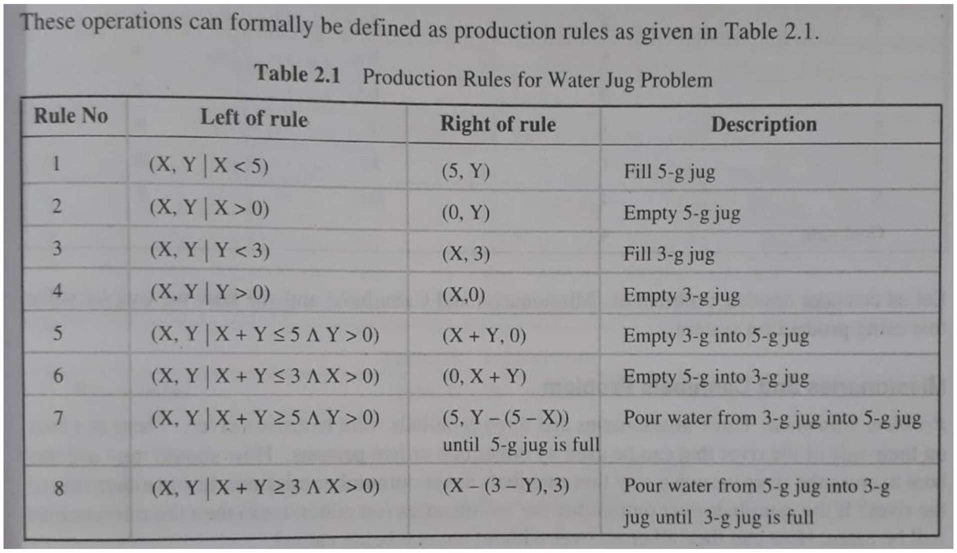
1. It is a good way to Model the strong state driven nature of intelligent action.
2. Unique can be easily added to the to account for new situations without disturbing the rest of the system.
3. Despite important in a real time environment and applications value into the database changes the behavior of the system.

Water jug problem.

Problem statement: We have to check a If I gallon and other 3 gallon Jug with you measure in marks on them. There is endless supply of water through tap. Our task is to get 4 gallons of water in 5 gallon Jug.

Solution: State space for this problem can be described as a set of ordered pairs of integers(X, Y) Such that. X represents the number of gallons of water in 5 gallons and Y for 3 gallon subject.

1. Start Start state is (0,0)
2. Goal state is (4, N) For any value of N<=3.



These operations can formally be defined as production rules as given in Table 2.1.

Table 2.1 Production Rules for Water Jug Problem

Rule No	Left of rule	Right of rule	Description
1	$(X, Y \mid X < 5)$	$(5, Y)$	Fill 5-g jug
2	$(X, Y \mid X > 0)$	$(0, Y)$	Empty 5-g jug
3	$(X, Y \mid Y < 3)$	$(X, 3)$	Fill 3-g jug
4	$(X, Y \mid Y > 0)$	$(X,0)$	Empty 3-g jug
5	$(X, Y \mid X + Y \leq 5 \wedge Y > 0)$	$(X + Y, 0)$	Empty 3-g into 5-g jug
6	$(X, Y \mid X + Y \leq 3 \wedge X > 0)$	$(0, X + Y)$	Empty 5-g into 3-g jug
7	$(X, Y \mid X + Y \geq 5 \wedge Y > 0)$	$(5, Y - (5 - X))$ until 5-g jug is full	Pour water from 3-g jug into 5-g jug
8	$(X, Y \mid X + Y \geq 3 \wedge X > 0)$	$(X - (3 - Y), 3)$	Pour water from 5-g jug into 3-g jug until 3-g jug is full

SOLUTION PATH 1

RULE APPLIED	5G JUG	3G JUG	STEP NO
START STATE	0	0	
1	5	0	1
8	2	3	2
4	2	3	3
6	0	2	4
1	5	2	5
8	4	3	6
GOAL STATE	4		

SOLUTION PATH 2

RULE APPLIED	5G JUG	3G JUG	STEP NO
START STATE	0	0	
3	0	3	1
5	3	0	2
3	3	3	3
7	5	1	4
2	0	1	5
5	1	0	6
3	1	3	7
5	4	0	8
GOAL STATE	4	-	-

Missionaries and cannibals problem:

Problem statement: 3 Missionaries and 3 cannibals went to cross a river. There is a Boat on On this side of the river that can be used by either one or two persons. How should they use this boat to cross the river in such a way that cannibals never outnumber missionaries on either side of the river? If the cannibals ever outnumber the missionaries then the missionary's will be eaten.

Solution: State space for this problem can be described as the set of ordered pairs of left and right banks of the river as (L,R) Each bank is represented as a list in [nM,mC,B]. Is the number of Missionaries M, m Is a number of cannibals C . And B represents boat

1. start state : ([3M,3C,1B], [0M,0C,0B]) ,1B means boat is present
2. Any state : ([n1M,m1C,--], [n2M,m2C,--]), with constraints /conditions at any state as n1(!=0)>=m1; n2(!=0)>=m2;n1+n2=3, m1+m2=3
3. Goal state: ([0M,0C,0B], [3M,3C,1B])

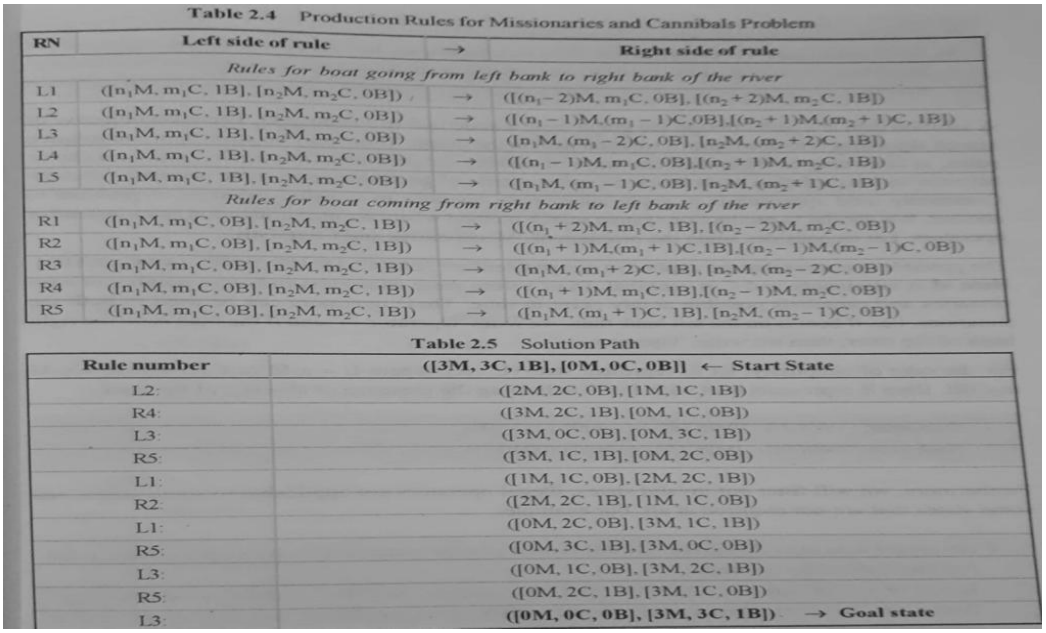


Table 2.4 Production Rules for Missionaries and Cannibals Problem

RN	Left side of rule	→	Right side of rule
	Rules for boat going from left bank to right bank of the river		
L1	$([n_1M, m_1C, 1B], [n_2M, m_2C, 0B])$	→	$([(n_1-2)M, m_1C, 0B], [(n_2+2)M, m_2C, 1B])$
L2	$([n_1M, m_1C, 1B], [n_2M, m_2C, 0B])$	→	$([(n_1-1)M, (m_1-1)C, 0B], [(n_2+1)M, (m_2+1)C, 1B])$
L3	$([n_1M, m_1C, 1B], [n_2M, m_2C, 0B])$	→	$([n_1M, (m_1-2)C, 0B], [n_2M, (m_2+2)C, 1B])$
L4	$([n_1M, m_1C, 1B], [n_2M, m_2C, 0B])$	→	$([(n_1-1)M, m_1C, 0B], [(n_2+1)M, m_2C, 1B])$
L5	$([n_1M, m_1C, 1B], [n_2M, m_2C, 0B])$	→	$([n_1M, (m_1-1)C, 0B], [n_2M, (m_2+1)C, 1B])$
	Rules for boat coming from right bank to left bank of the river		
R1	$([n_1M, m_1C, 0B], [n_2M, m_2C, 1B])$	→	$([(n_1+2)M, m_1C, 1B], [(n_2-2)M, m_2C, 0B])$
R2	$([n_1M, m_1C, 0B], [n_2M, m_2C, 1B])$	→	$([(n_1+1)M, (m_1+1)C, 1B], [(n_2-1)M, (m_2-1)C, 0B])$
R3	$([n_1M, m_1C, 0B], [n_2M, m_2C, 1B])$	→	$([n_1M, (m_1+2)C, 1B], [n_2M, (m_2-2)C, 0B])$
R4	$([n_1M, m_1C, 0B], [n_2M, m_2C, 1B])$	→	$([(n_1+1)M, m_1C, 1B], [(n_2-1)M, m_2C, 0B])$
R5	$([n_1M, m_1C, 0B], [n_2M, m_2C, 1B])$	→	$([n_1M, (m_1+1)C, 1B], [n_2M, (m_2-1)C, 0B])$

Table 2.5 Solution Path

Rule number	([3M, 3C, 1B], [0M, 0C, 0B]] ← Start State
L2:	([2M, 2C, 0B], [1M, 1C, 1B])
R4:	([3M, 2C, 1B], [0M, 1C, 0B])
L3:	([3M, 0C, 0B], [0M, 3C, 1B])
R5:	([3M, 1C, 1B], [0M, 2C, 0B])
L1:	([1M, 1C, 0B], [2M, 2C, 1B])
R2:	([2M, 2C, 1B], [1M, 1C, 0B])
L1:	([0M, 2C, 0B], [3M, 1C, 1B])
R5:	([0M, 3C, 1B], [3M, 0C, 0B])
L3:	([0M, 1C, 0B], [3M, 2C, 1B])
R5:	([0M, 2C, 1B], [3M, 1C, 0B])
L3:	**([0M, 0C, 0B], [3M, 3C, 1B])** → **Goal state**

State space search

Simple two production system state space is another method of problem representing the facility easy search. Using this method one can also find a path from start State goal state while solving a problem is state space basically consists of four components:

1. A Set S Containing start states of the problem.
2. A set G containing gold States Of the problem.
3. Set of notes in the graph or tree each node represents the state in problem solving process.
4. Set of ARCS connecting node each arc. Response to the operator that is a step in problem solving process.

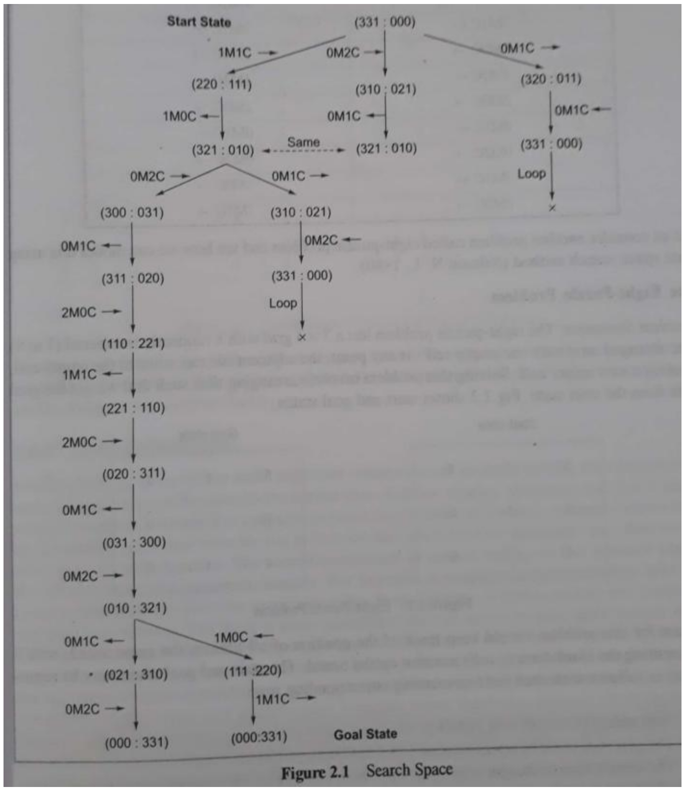


Figure 2.1 Search Space

8 puzzle problem:

Problem statement: 8 puzzle problem has 3 cross 3 grid with 8 randomly Number 12 age. Tiles. Arranged on it with one empty cell at any point the adjacent time can move to the empty shell. Creating a new empty cell. Solve this problem involves arranging X such that we get the coal start from the starting state two goals State.

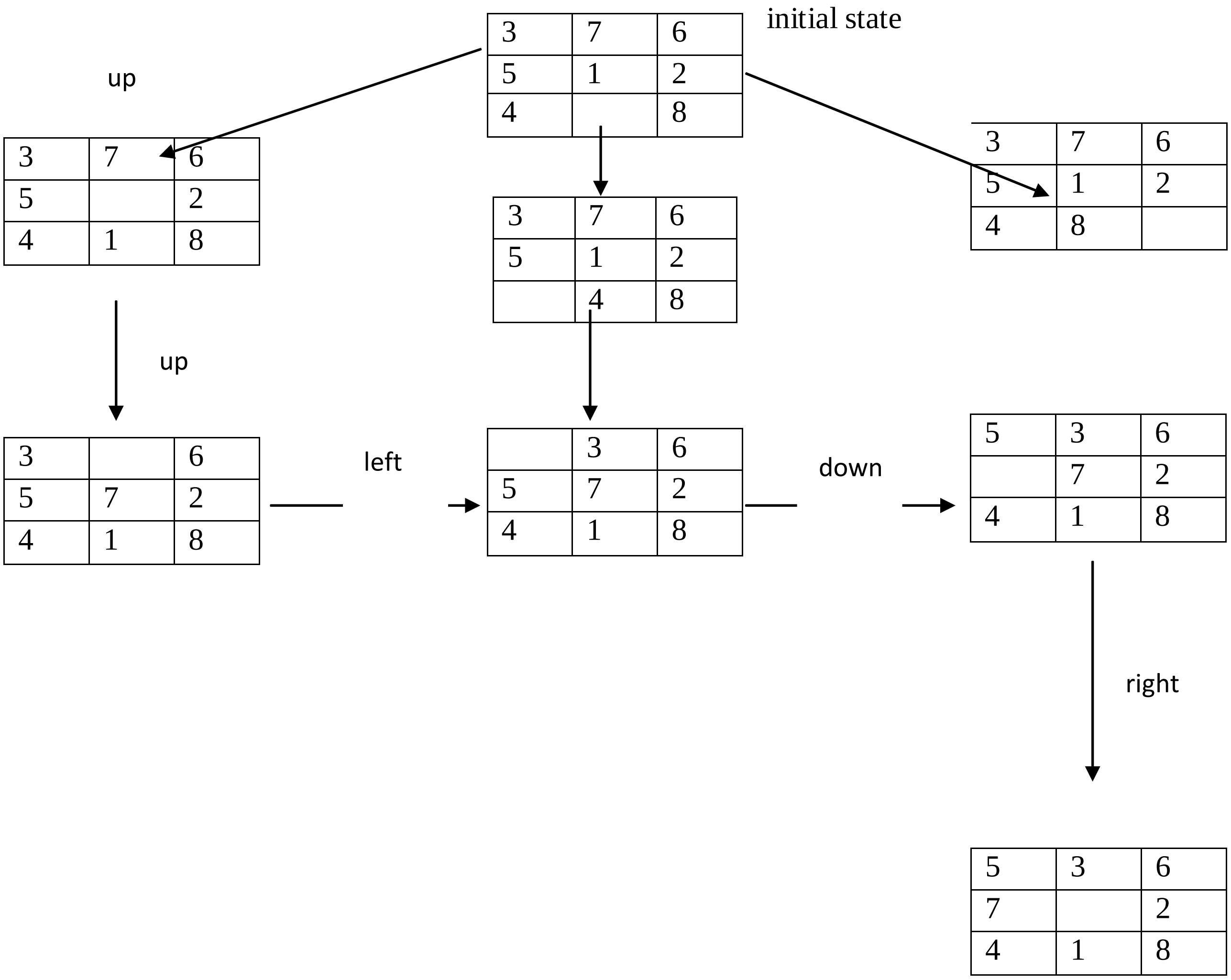
start state

3	7	6
5	1	2
4		8

goal state

5	3	6
7		2
4	1	8

1. START state [[3,7,6],[5,1,2],[4,0,8]]
2. the goal state could represent as [[5,3,6],[7,0,2],[4,1,8]]
3. the operators can be thought of moving (up,down,left,right} the directions in which blank space effectively moves



3	7	6
5	1	2
4		8

3	7	6
5		2
4	1	8

3	7	6
5	1	2
	4	8

3	7	6
5	1	2
4	8	

3		6
5	7	2
4	1	8

	3	6
5	7	2
4	1	8

5	3	6
	7	2
4	1	8

5	3	6
7		2
4	1	8

Control strategies:
control strategy is one of the most important component of problem solving that describes the order of applications of the rule to the current state. Control strategy should be such that it causes the motion towards the solution. It is that it should explore the solution space in a systematic manner. If you select a control strategy where we select a rule randomly for the applicable room but definitely it causes the motion and eventually with the to the solution.

The following strategies which are used in the control strategy.

1. Exhaustive
2. uninformed
3. blind searches in nature.

There are two directions in which. Such a such good proceed.
Data driven search call forward chaining from the start State.
Goal driven search called backward chaining from the goal state.
Forward chaining: The process of forward chaining begins with unknown facts and works towards a conclusion. Rules are expressed in the form of if then rules.
Backward chaining: Is it Goal directed strategy that begins with the goal state and continuous working backwards generating more sub goals that must also be satisfied to satisfy main goal and till we reach to start State.

Characteristics of a problem.
Types of problem there are three types of problems in real life ignorable, recoverable and irrecoverable.
Ignorable: These are the problems where we can ignore the solution steps. Which problems can be solved using simple control strategies?
Recoverable: These are the problems where solutions can be undone. Such problems can be solved by backtracking to control strategies can be implemented using push down stack.
Irrecoverable: These problems with solutions steps cannot be undone. For example any two players game such as chess playing cards Snakes and Ladders etc. Are examples of this strategy such problems can be solved by playing process.

Decomposability of a problem: Divide the problem into a set of independent smaller some problems solved them and combine the solution to get the final solution. The process of dividing some problem continues till we get the receipt of smallest sub problems for which small collection of specific rules are used. Divide and conquer technique is commonly used method for solving such problems. Which can be solved in parallel processing Environment?

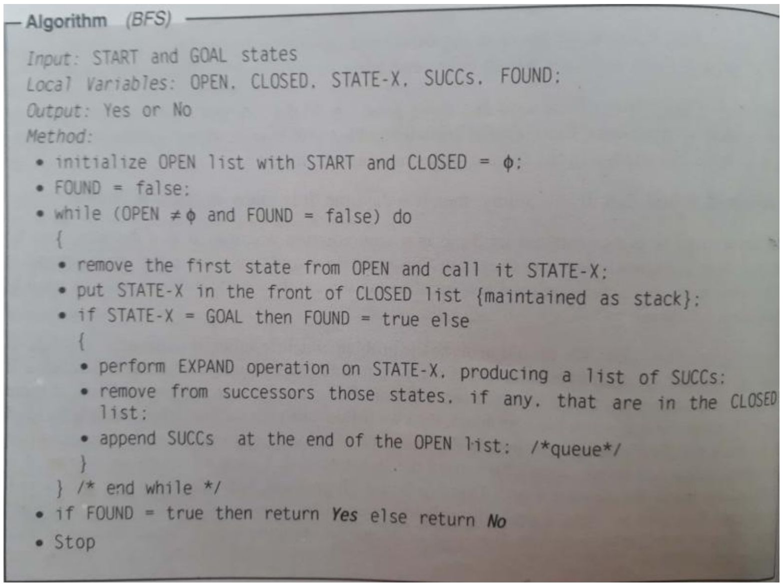
Role of knowledge: Knowledge plays an important role in solving any problem knowledge could be in the form of rules. And Facts which helps generating search space for finding the solution.

Consistency of knowledge base used in solving problem: Which should that knowledge base used to solve problem is consistent inconsistent knowledge base will lead to wrong solution.
Like if it is humid it will rain, If it is sunny then it is a daytime.

Requirement of solutions: We should analyze the problem whether the solution required is absolute or relative. We call solution to be absolute if you have to find exact solution. Where as it is relative if you have reasonable good and approximate solution.

Exhaustive search

Breadth first search: In BFS Explain all the states of one step away from the start state and then expand all the states Two Steps from the start State then three steps until a goal state is reached all states are exact examined at the same death before going deeper. BFS always gives an optimal path or solutions.



```
Algorithm (BFS)

Input: START and GOAL states
Local Variables: OPEN, CLOSED, STATE-X, SUCCs, FOUND;
Output: Yes or No
Method:
 • initialize OPEN list with START and CLOSED = $\phi$;
 • FOUND = false;
 • while (OPEN $\neq \phi$ and FOUND = false) do
   {
   • remove the first state from OPEN and call it STATE-X;
   • put STATE-X in the front of CLOSED list {maintained as stack};
   • if STATE-X = GOAL then FOUND = true else
     {
     • perform EXPAND operation on STATE-X, producing a list of SUCCs;
     • remove from successors those states, if any, that are in the CLOSED
       list;
     • append SUCCs at the end of the OPEN list; /*queue*/
     }
   } /* end while */
 • if FOUND = true then return Yes else return No
 • Stop
```

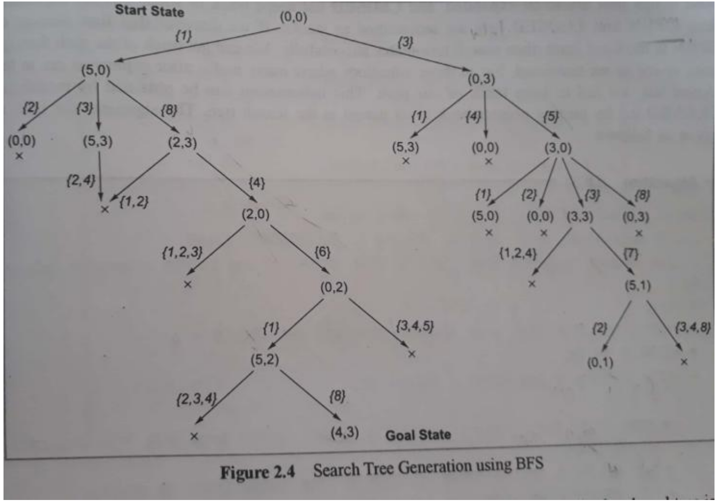
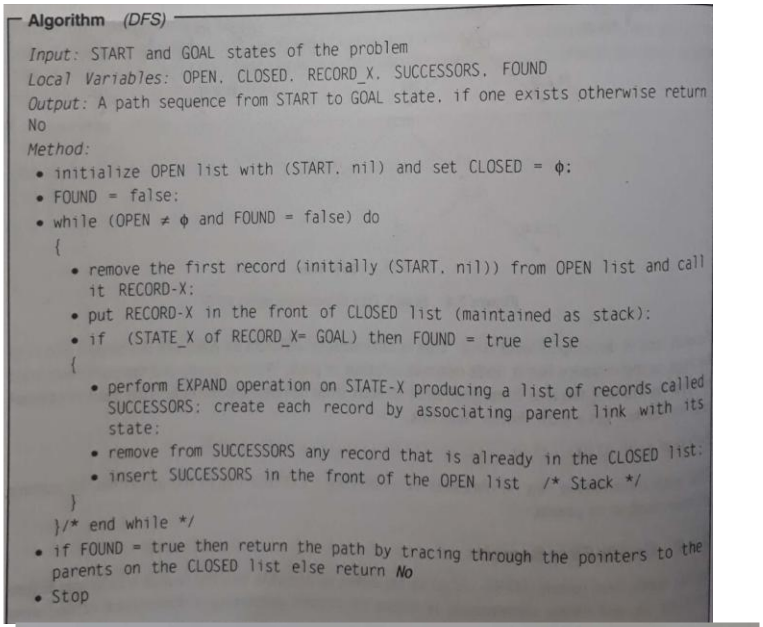
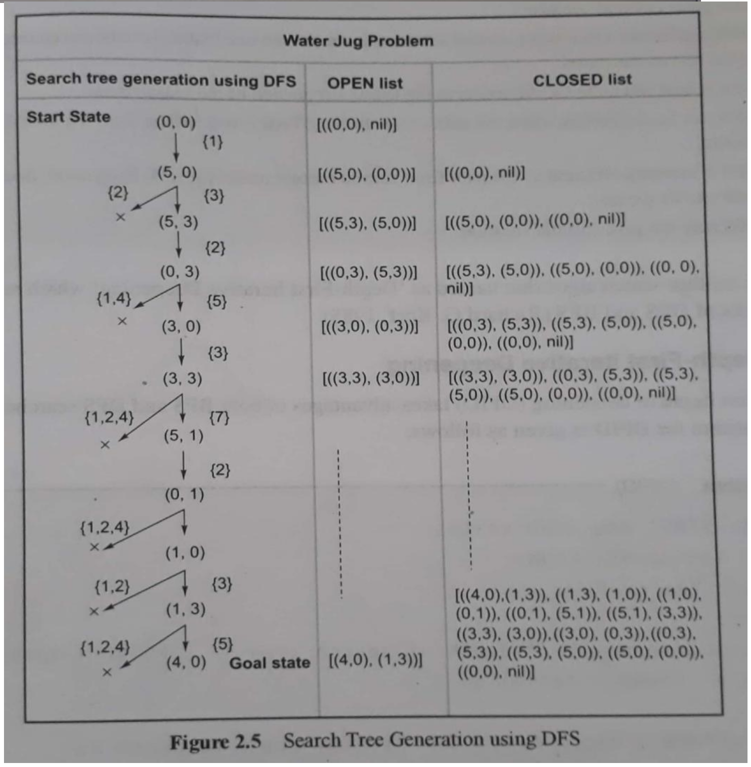


Figure 2.4 Search Tree Generation using BFS

Depth first search. In as far as possible or before backing up and trying alternative it works by always generating a descendant of the most recently expanded until some Dept of is reached and then back to the next most recently expanded node generate one of its descendants. dfs Is memory efficient and it only store a single path from the root to leaf node along with the remaining and expanded siblings for each node in the path.



```
Algorithm  (DFS)
Input: START and GOAL states of the problem
Local Variables: OPEN, CLOSED, RECORD_X, SUCCESSORS, FOUND
Output: A path sequence from START to GOAL state, if one exists otherwise return
No
Method:
• initialize OPEN list with (START, nil) and set CLOSED = ϕ;
• FOUND = false;
• while (OPEN ≠ ϕ and FOUND = false) do
  {
    • remove the first record (initially (START, nil)) from OPEN list and call
      it RECORD-X;
    • put RECORD-X in the front of CLOSED list (maintained as stack);
    • if  (STATE_X of RECORD_X= GOAL) then FOUND = true  else
    {
      • perform EXPAND operation on STATE-X producing a list of records called
        SUCCESSORS; create each record by associating parent link with its
        state;
      • remove from SUCCESSORS any record that is already in the CLOSED list;
      • insert SUCCESSORS in the front of the OPEN list  /* Stack */
    }
  }/* end while */
• if FOUND = true then return the path by tracing through the pointers to the
  parents on the CLOSED list else return No
• Stop
```



Water Jug Problem

Search tree generation using DFS	OPEN list	CLOSED list
Start State (0, 0)	[((0,0), nil)]	
(5, 0)	[((5,0), (0,0))]	[((0,0), nil)]
(5, 3)	[((5,3), (5,0))]	[((5,0), (0,0)), ((0,0), nil)]
(0, 3)	[((0,3), (5,3))]	[((5,3), (5,0)), ((5,0), (0,0)), ((0, 0), nil)]
(3, 0)	[((3,0), (0,3))]	[((0,3), (5,3)), ((5,3), (5,0)), ((5,0), (0,0)), ((0,0), nil)]
(3, 3)	[((3,3), (3,0))]	[((3,3), (3,0)), ((0,3), (5,3)), ((5,3), (5,0)), ((5,0), (0,0)), ((0,0), nil)]
(5, 1)		
(0, 1)		
(1, 0)		
(1, 3)		
(4, 0) Goal state	[((4,0), (1,3))]	[((4,0),(1,3)), ((1,3), (1,0)), ((1,0), (0,1)), ((0,1), (5,1)), ((5,1), (3,3)), ((3,3), (3,0)),((3,0), (0,3)),((0,3), (5,3)), ((5,3), (5,0)), ((5,0), (0,0)), ((0,0), nil)]

Figure 2.5 Search Tree Generation using DFS

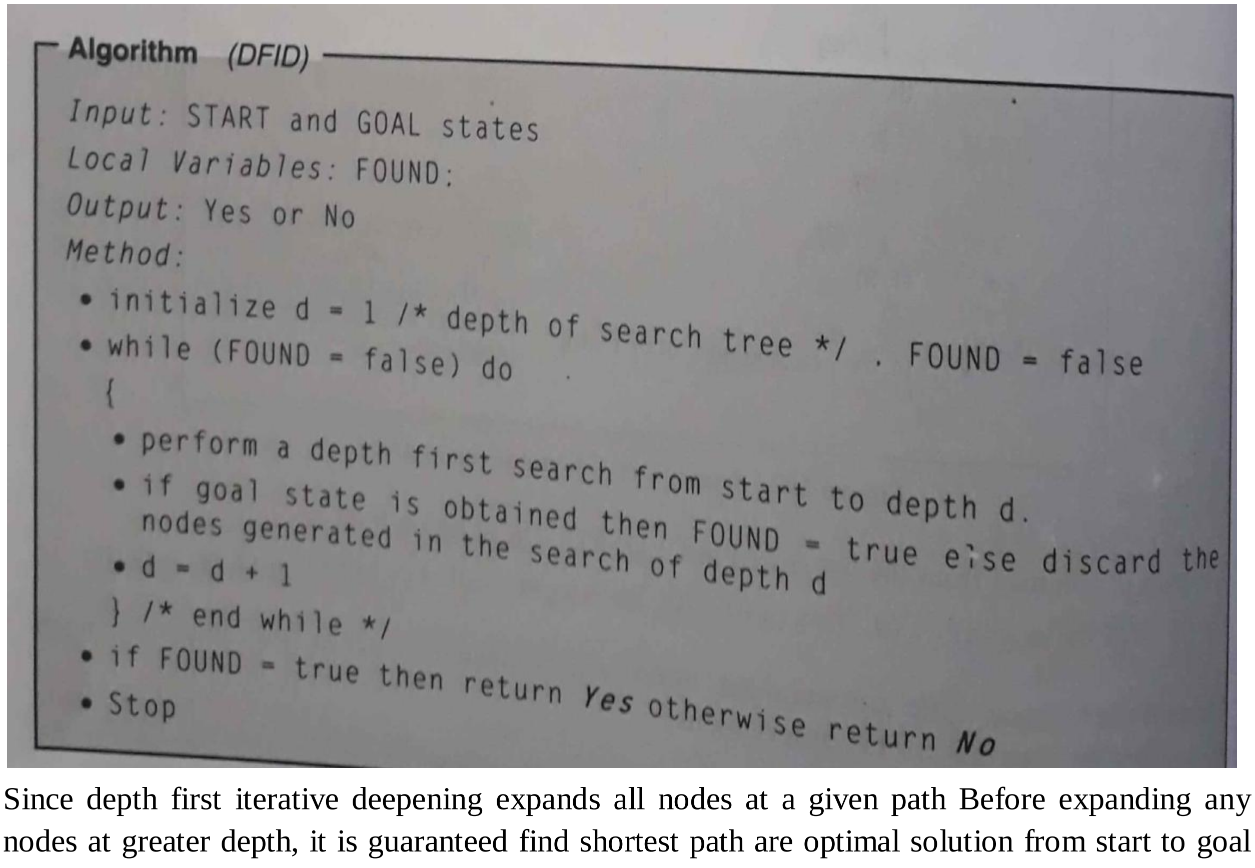
the path is obtained from the list stored in CLOSED .The solution path is
(0,0)->(5,0)->(5,3)->(0,3)->(3,0)->(3,3)->(5,1)->(0,1)->(1,0)->(1,3)->(4,0)

Comparisons:

1. BFS is effective when the search tree has a low branching factor
2. BFS can work even in trees that are infinitely deep
3. BFS requires a lot of memory as number of nodes in level of the tree increases exponentially
4. BFS is superior when the goal exists in the upper right portion of search tree.
5. BFS gives optimal solution
6. DFS is effective when there are a few sub trees in the search tree that have only one connection point to the rest of the states
7. DFS is best when the goal exists in the lower left portion of the search tree
8. DFS can be dangerous when the path closer to the start and farther from the goal has been chosen
9. DFS is memory efficient as the path from the start to current node is stored .Each node should contain state and its parent
10. DFS may not give optimal solutions.

DEAPTH FIRST ITERATIVE DEEPENING

advantages of both BFs and DFS search on trees.



Algorithm *(DFID)*

```
Input: START and GOAL states
Local Variables: FOUND:
Output: Yes or No
Method:
 • initialize d = 1 /* depth of search tree */ . FOUND = false
 • while (FOUND = false) do
   {
    • perform a depth first search from start to depth d.
    • if goal state is obtained then FOUND = true else discard the
      nodes generated in the search of depth d
    • d = d + 1
   } /* end while */
 • if FOUND = true then return Yes otherwise return No
 • Stop
```

Since depth first iterative deepening expands all nodes at a given path Before expanding any nodes at greater depth, it is guaranteed find shortest path are optimal solution from start to goal state. The working of algorithm is shown as in figure below

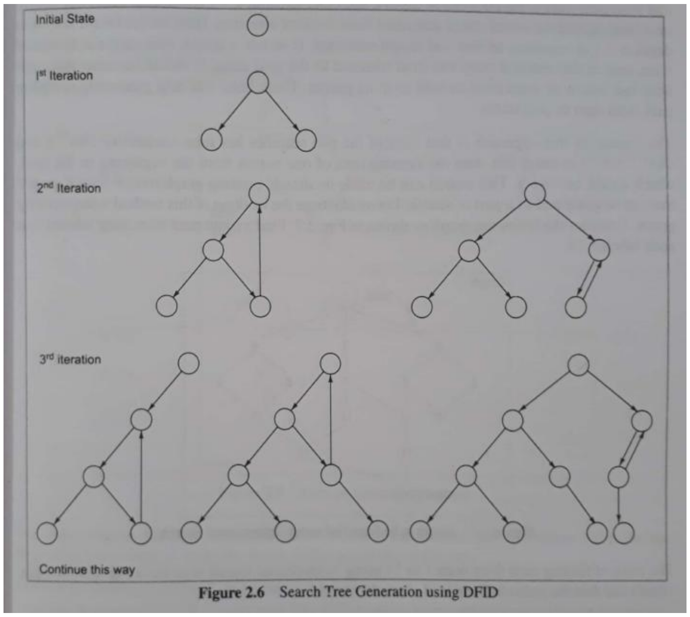


Figure 2.6 Search Tree Generation using DFID

BIDIRECTIONAL SEARCH

Bidirectional search is a Graph Search Algorithm that runs two simultaneous searches. One search forward start date and other moon backward from the goal state and stops when the to meet in the middle. It is useful for those problems which have a single start state and single goal state. It can be applied to bidirectional search for K = 123 ... k iteration consist of generating state in the forward direction from the start date up to the depth of k using BFS from which state using dfs 1 to death and other to the death K + 1 storing States but simply matching against the stood state guaranteed from forward direction. The reason for this approach is that each of the structures has time complexity O(b power of d/2)

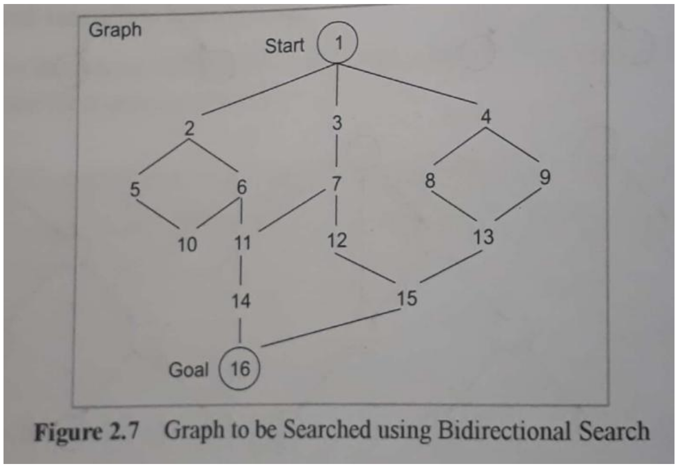


Figure 2.7 Graph to be Searched using Bidirectional Search

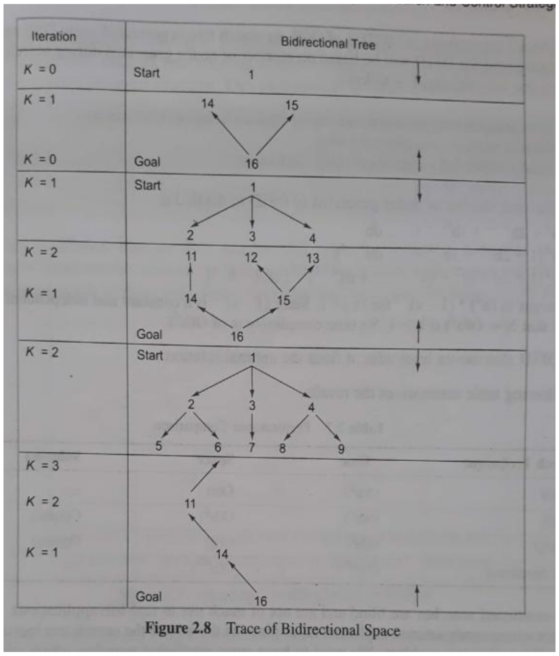


Figure 2.8 Trace of Bidirectional Space

the trace of finding the path from node 1 to 16 using Bidirectional search is given in above figure we can clearly see that the part is obtained is: 1 2 6 11 14 16

Analysis of search methods

completeness :it m it means that an algorithm guarantees is solution if it exist

time complexity: time required by an algorithm to find a solution.

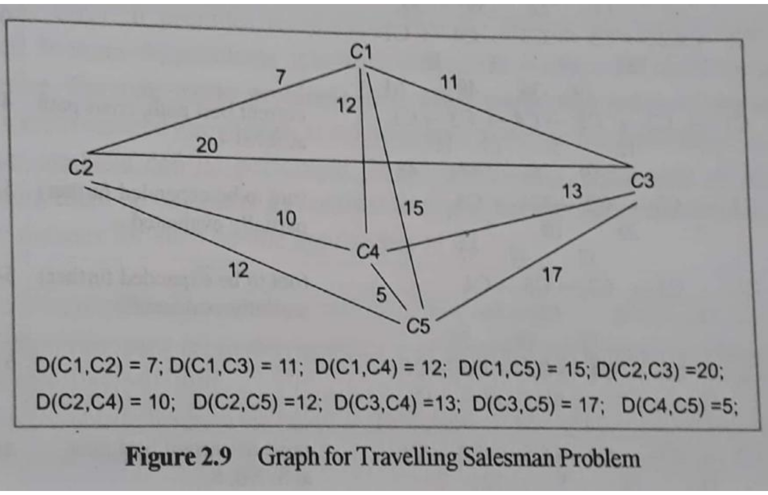
Space complexity: space required by an algorithm to find the solution.

Search technique	Time	Space	Solution
DFS	$O(B^{D)}$	O(D)	----
BFS	$O(B^{D)}$	$O(B^{D)}$	OPTIMAL
DFID	$O(B^{D)}$	O(D)	OPTIMAL
BI-DIRECTIONAL	$O(B^{D/2)}$	$O(B^{D/2)}$	---

travelling salesperson problem

statement: in travelling salesperson problem when is required to find stockist pass visiting all cities once and returning back to starting point there are n cities and distance between each pair of the city is given. if number of cities grow the time required to wait a salesman to get the information about the shortest path is not practical situation this phenomena is called combinatorial explosion.

- Start generating complete paths, keeping track of shortest path found so far.
- Stop exploring any path practical length becomes greater than shortest path length found so far



D(C1,C2) = 7; D(C1,C3) = 11; D(C1,C4) = 12; D(C1,C5) = 15; D(C2,C3) =20;
D(C2,C4) = 10; D(C2,C5) =12; D(C3,C4) =13; D(C3,C5) = 17; D(C4,C5) =5;

Figure 2.9 Graph for Travelling Salesman Problem

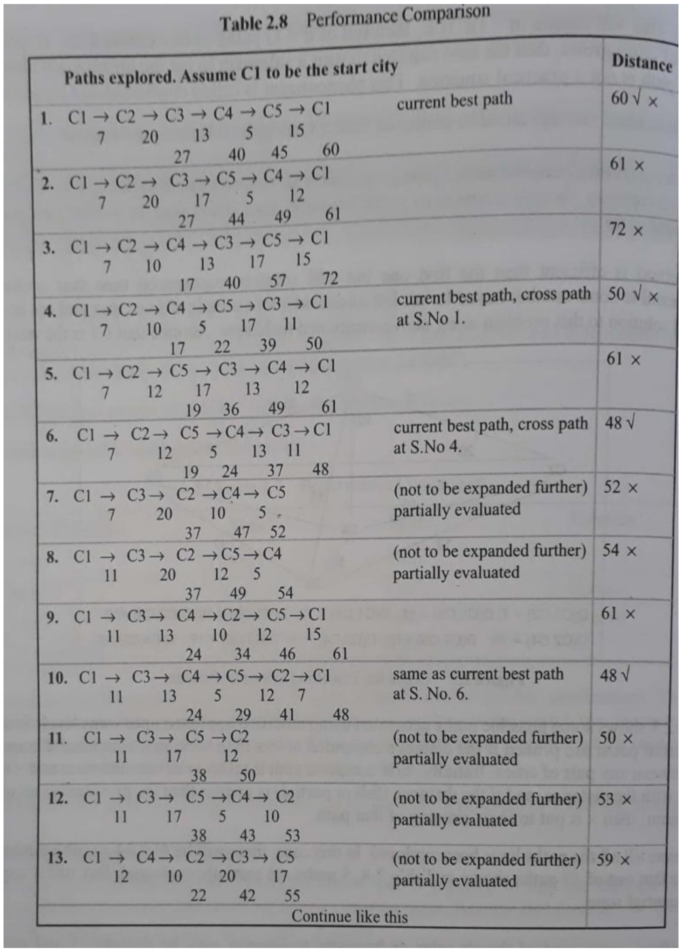


Table 2.8 Performance Comparison

Paths explored. Assume C1 to be the start city		Distance
1. C1 → C2 → C3 → C4 → C5 → C1 7 20 13 5 15 27 40 45 60	current best path	60 √ ×
2. C1 → C2 → C3 → C5 → C4 → C1 7 20 17 5 12 27 44 49 61		61 ×
3. C1 → C2 → C4 → C3 → C5 → C1 7 10 13 17 15 17 40 57 72		72 ×
4. C1 → C2 → C4 → C5 → C3 → C1 7 10 5 17 11 17 22 39 50	current best path, cross path at S.No 1.	50 √ ×
5. C1 → C2 → C5 → C3 → C4 → C1 7 12 17 13 12 19 36 49 61		61 ×
6. C1 → C2→ C5 →C4→ C3→C1 7 12 5 13 11 19 24 37 48	current best path, cross path at S.No 4.	48 √
7. C1 → C3→ C2 →C4→ C5 7 20 10 5 37 47 52	(not to be expanded further) partially evaluated	52 ×
8. C1 → C3→ C2 →C5→C4 11 20 12 5 37 49 54	(not to be expanded further) partially evaluated	54 ×
9. C1 → C3→ C4 →C2→ C5→C1 11 13 10 12 15 24 34 46 61		61 ×
10. C1 → C3→ C4 →C5→ C2→C1 11 13 5 12 7 24 29 41 48	same as current best path at S. No. 6.	48 √
11. C1 → C3→ C5 →C2 11 17 12 38 50	(not to be expanded further) partially evaluated	50 ×
12. C1 → C3→ C5 →C4→ C2 11 17 5 10 38 43 53	(not to be expanded further) partially evaluated	53 ×
13. C1 → C4→ C2 →C3→ C5 12 10 20 17 22 42 55	(not to be expanded further) partially evaluated	59 ×
Continue like this		

HEURISTIC SEARCH TECHNIQUES

it is criteria determining among several alternatives will be the achieve this technique search process possible by sacrificing claims of Systematic and completeness. Two types of heuristic namely

1. general purpose heuristic that useful in various problem domains

2. special purpose heuristic that domain specific

General purpose heuristics

general purpose heuristics combinatorial problems is nearest neighbor algorithms that work by selecting locally superior alternative

it is a fun to see a program do something intelligent than To prove it

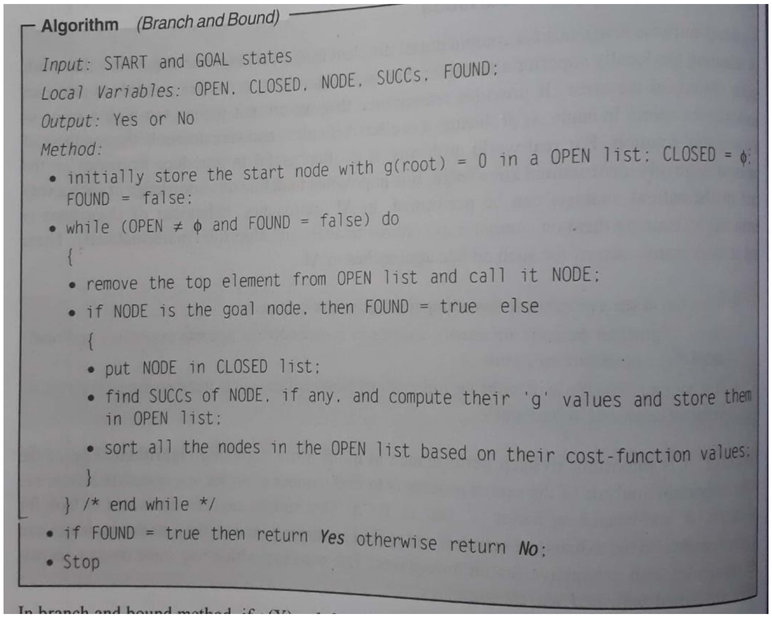
since AI problem domains are complex it is generally not possible produce analytical proof a procedure will work

it is not even possible to describe the range of problems well well enough to me status statistical analysis of program behavior meaningful

the search which use some domain knowledge informed search strategies

Branch and bound search(uniform cost search)

this method uses cost function denoted by g(X) designed assign assign accumulated expense to the path from start node to the current node X applying the sequence of operator. during search process there can be many incomplete path contending for further consideration. short one always extended 1 level further, creating as many new incomplete paths as there are branches.



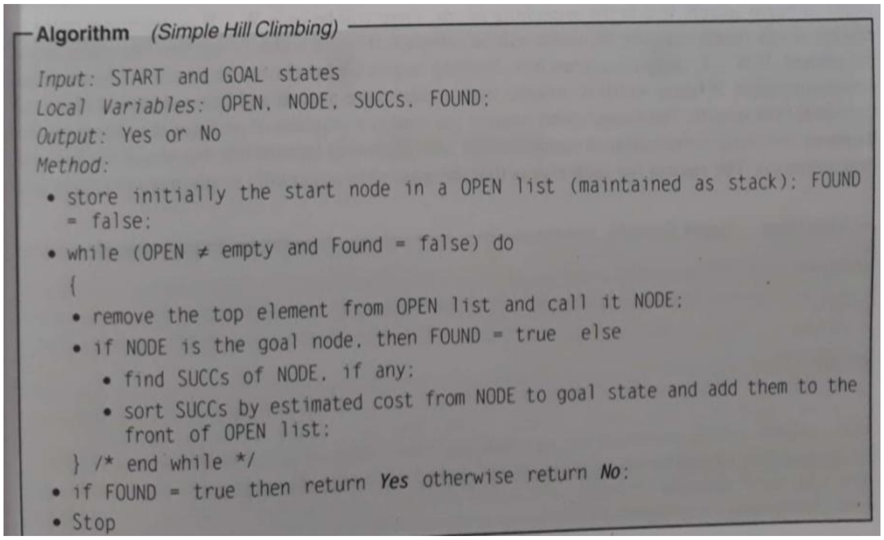
```
Algorithm (Branch and Bound)

Input: START and GOAL states
Local Variables: OPEN. CLOSED. NODE. SUCCs. FOUND:
Output: Yes or No
Method:
  • initially store the start node with g(root) = 0 in a OPEN list: CLOSED = ϕ:
    FOUND = false:
  • while (OPEN ≠ ϕ and FOUND = false) do
    {
      • remove the top element from OPEN list and call it NODE:
      • if NODE is the goal node. then FOUND = true  else
        {
          • put NODE in CLOSED list:
          • find SUCCs of NODE. if any. and compute their 'g' values and store them
            in OPEN list:
          • sort all the nodes in the OPEN list based on their cost-function values:
        }
    } /* end while */
  • if FOUND = true then return Yes otherwise return No:
  • Stop
```

In branch and bound method if g(X)=1 all operator then degenerate Hindi to simple BFs.

from AI. point of view as bad as depth first and breadth first. this can be improved augment it bi dynamic programming ladies that is delete those parts which are redundant

hill climbing

It is an Optimization technique that belong belong belong family of local purchase. it is a relatively simple technique to implement popular first choice is explored. hill climbing problems that have many solutions but some solutions are then other moving through a tree of 5 proceed in depth first order but the choices are according to some heuristic value



```
Algorithm (Simple Hill Climbing)

Input: START and GOAL states
Local Variables: OPEN, NODE, SUCCs, FOUND;
Output: Yes or No
Method:
 • store initially the start node in a OPEN list (maintained as stack); FOUND
   = false;
 • while (OPEN ≠ empty and Found = false) do
   {
    • remove the top element from OPEN list and call it NODE;
    • if NODE is the goal node, then FOUND = true  else
       • find SUCCs of NODE, if any;
       • sort SUCCs by estimated cost from NODE to goal state and add them to the
         front of OPEN list;
   } /* end while */
 • if FOUND = true then return Yes otherwise return No;
 • Stop
```

Problems with hill climbing

Local maximum: it is a state that is better than all neighbor but not better than some other states which are far away

Plateau: it is a flat area of space where all neighbor states have same value. it is not possible best direction in situations make a big jump some direction and try to get new section how to search space

Ridge: It is an area of search space that is higher than surrounding areas but that cannot traversed by single moves in any one direction. It is a special kind of local maxima. Here apply two or more rules before doing the test, i.e. moving in several directions at once

Beam Search

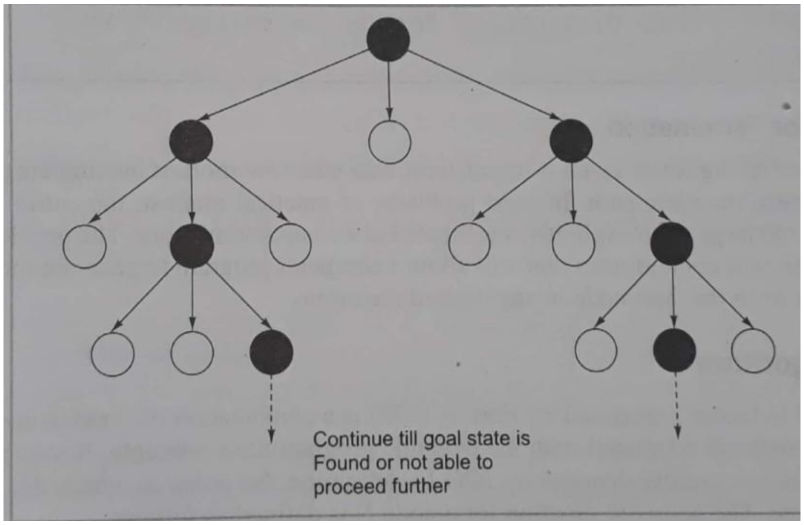
Beam search is a heuristic search algorithm in which W number of best nodes at each level is always expanded. It progresses level by level and moves downward only from the best W nodes at each level. Beam search uses breadth-first search to build its search tree. At each level of the tree. it generates all successors of the states at the current level, sorts them in order of increasing heuristic values. However, it only considers a W number of states at each level. Other nodes are ignored. Best nodes are decided on the heuristic cost associated with the node. Here W is called width of beam search. If B is the branching factor, there will be only W * B nodes under consideration at any depth but only W nodes will be selected. If beam width is smaller, the more states are pruned. If W = l, then it becomes hill climbing search where always best node is chosen from successor nodes. If beam width is infinite, then no states are pruned and beam search is identical to breath-first search.

Algorithm (Beam Search)
Input: START and GOAL states SUCCs, W_OPEN, found;
Local Variables: OPEN, NODE, FOUND,

Output • Yes or No

Method.

```
node=root_node,Foung== false;
• If NODE 1 s the goal node. = true else find SUCCs if any with its estimated cost and store in OPEN list
• while (FOUND = false and not able to proceed further) do
{
    • sort OPEN list:
    . select top elements from OPEN list and put it in W OPEN list and empty OPEN list;
    • for each NODE from W OPEN list
    {
        • if NODE = Goal state then FOUND true else find SUCCs of NODE. if any with its estimated cost and store in OPEN list:
    }
} /* end while */
if FOUND true then return Yes otherwise return No;
Stop
```



A* Algorithm

it is a combination of 'branch and bound' and 'best search' methods combined with the dynamic programming principle. It uses a heuristic evaluation function usually denoted by f(X) to determine the order in which the search visits nodes in the tree. The heuristic function for a node N is defined as follows:

$f(N)=g(N)+h(N)$

The function g is a measure of the cost of getting from the start node to the current node N, is sum of costs of the rules that were applied along the best path to the current node.

The function h is an estimate of additional cost of getting from the current node N to the goal node.

Generally, A* algorithm is called OR graph / tree search algorithm. A* algorithm incrementally searches all the routes starting from the start node until it finds tl shortest path to a goal. Starting with a given node, the algorithm expands the node with lowest f(X) value. It maintains a set of partial solutions.

let us consider an example of 8 puzzle again and solve it by using A* .The simple evalution function f(x) is defined as follows:

f(x)=g(x)+h(x),where

h(x)=the number of tiles not in the goal position in a given state X

g(x)=depth of node X in the search tree

given

start state

3	7	6
5	1	2
4		8

goal state

5	3	6
7		2
4	1	8

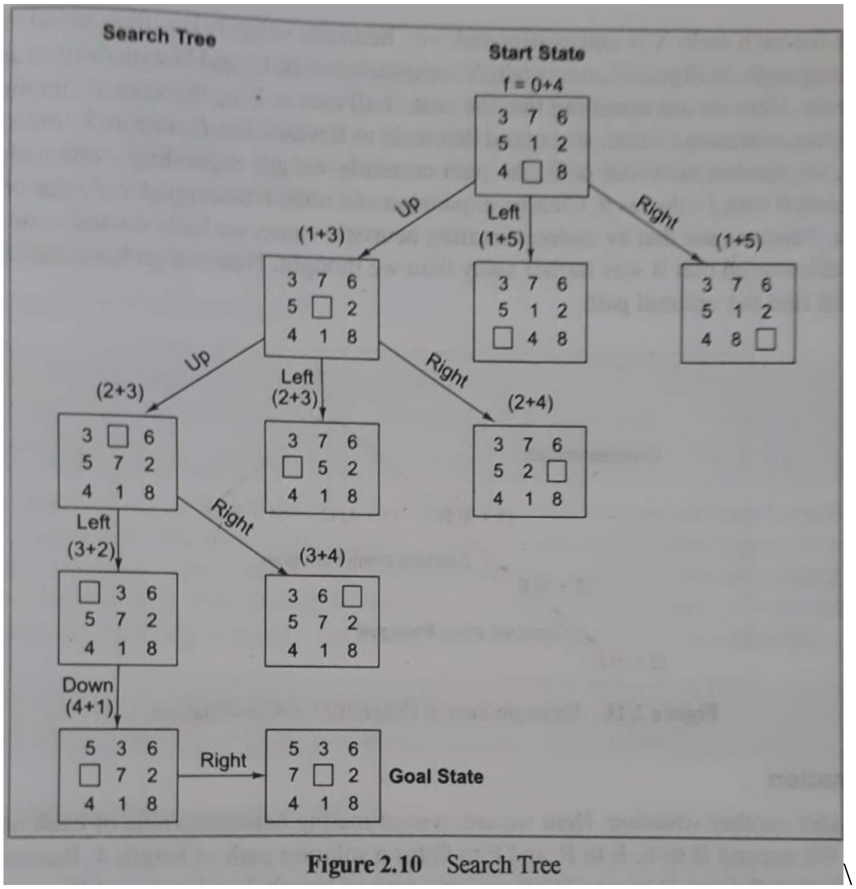


Figure 2.10 Search Tree

\

in the start state f=0+4 (g->level ,h=4(number of tiles misplaced ie 3,7,5,1))

a better estimate of h function might be as follows .the function g may remain same .
h(X)=the sum of distance of the tiles (1 to 8) from the goal position is given state X

1 2 3 4 5 6 7 8

here start state has h(start state)= 1+0+1+0+1+0+2+0=5

optimal solution by A* algorithm

A* algorithm ensures to find an optimal path to a goal, if one exists. Here we assume that h for each node X is underestimated. i.e., heuristic value is less than node X to goal node. In Fig. 2.11,start node A is expanded B ,C and D with f value as 4,5 and 6 respectively. Here assuming that the cost of all arc is 1 the sake of simplicity. Node B has minimum f value, so expand this node to E which has value as 5, C is also 5, we resolve in favor of E, the path currently we expanding, Now node E is expanded to node F with f value as 6, Clearly, expansion of a node F is stopped as f value of C the smallest. Thus. we sec that by underestimating heuristic value.

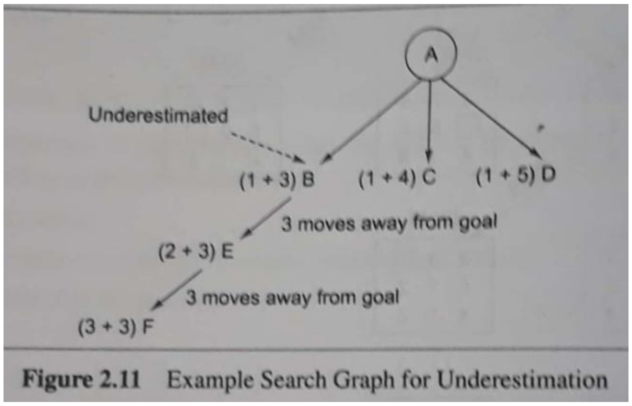


Figure 2.11 Example Search Graph for Underestimation

Overestimation

Let us consider another situation. Here we are overestimating heuristic value of each node in the graph/tree. We expand B to E, E to F, and F to G for a solution path o! length 4. But assume that there is a direct path from D to a solution giving a path of length 2 as h value of D is also overestimated. We will never find it because of overestimating h(D).We may find some other worse solution without ever expanding D. So by overestimating h, we cannot be guaranteed to find the shortest path. Overestimated Figure 2.12 Example Search Graph for Overestimation

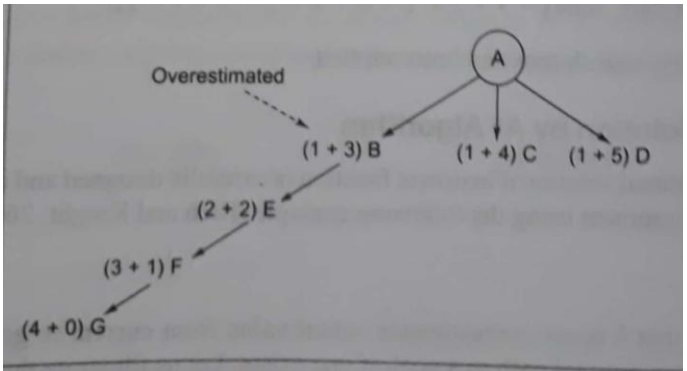


Figure 2.12 Example Search Graph for Overestimation

Admissibility of A*

A search algorithm is admissible, if for any graph, it always terminates in an optimal path from start state to goal state, if path exists. We have seen earlier that if heuristic function 'h' underestimates the actual value from current state to goal state, then it bounds to give an optimal solution and hence is called admissible function. So, we can say that A* always terminates with the optimal path in case h is an admissible heuristic function.

Monotonic Function

A heuristic function h is monotone if

I.✯states Xi and Xj such that Xj is successor of Xi
h(Xi) — h(Xj) cost (Xi, Xj) i.e., actual cost of going from Xi to Xj
2.h(Goal)=0

- A cost function f is monotone if f(N) <=f(succ(N))
- For any admissible cost function f, we can construct a monotonic admissible function.

Iterative-Deepening A*

Iterative-Deepening A* (IDA*) is a combination of the depth-first iterative deepening and A* algorithm. Here the successive iterations are

Corresponding to increasing values of the total cost of a path rather than increasing depth of the search. Algorithm works as follows:

- For each iteration, perform a DFS pruning off a branch when its total cost (g + h) exceeds a given threshold.
- The initial threshold starts at the estimate cost of the start state and increases for each iteration of the algorithm.
- The threshold used for the next iteration is the minimum cost of all values exceeded the current threshold.

• These steps are repeated till we find a goal state.

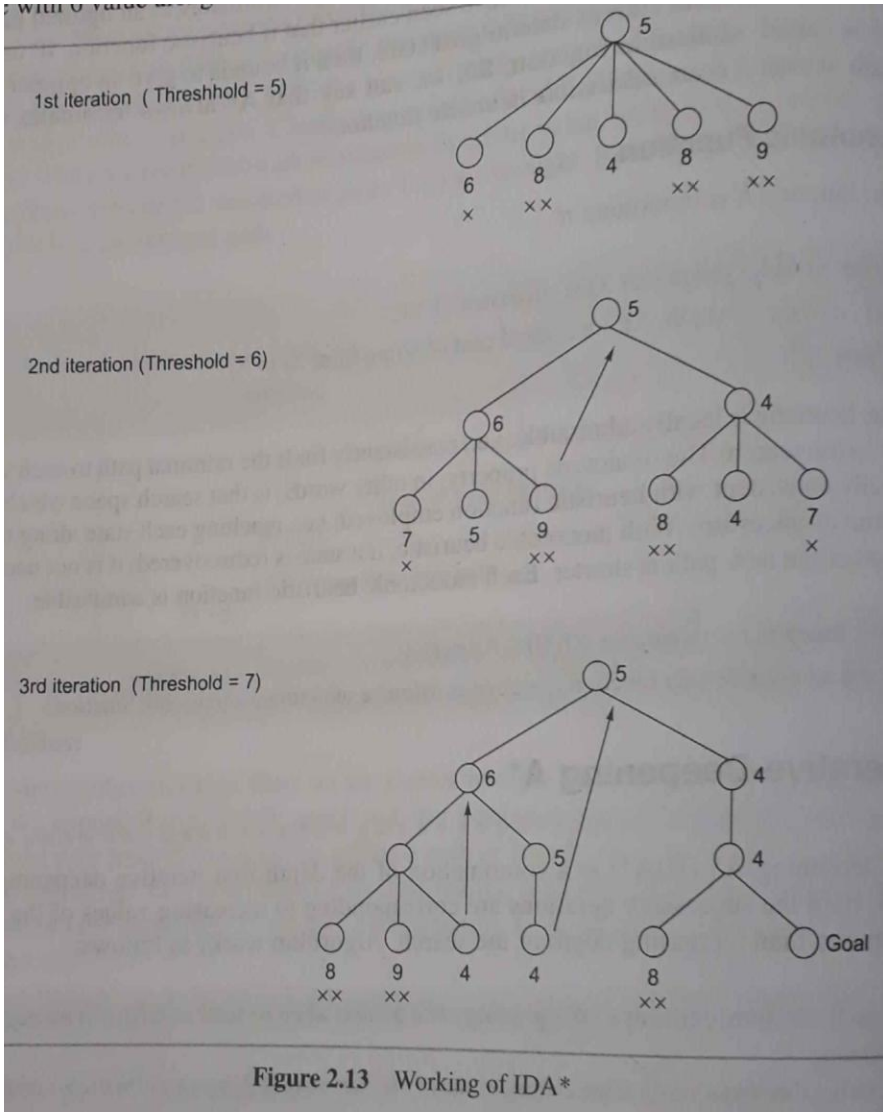

1st iteration (Threshhold = 5)
5
6
×
8
××
4
8
××
9
××
2nd iteration (Threshold = 6)
5
6
4
7
×
5
9
××
8
××
4
7
×
3rd iteration (Threshold = 7)
5
6
4
5
4
8
××
9
××
4
4
8
××
Goal

Figure 2.13 Working of IDA*

IDA* algorithm as shown in Fig. 2.13. Initially, the threshold value is the estimated cost of the

start node. In the first iteration, Threshold = 5' Now we generate all the successors of start node and compute their estimated values as 6, 8, 4' 8, and 9. The successors having values greater than 5 are to be pruned. Now for next iteration,we consider the threshold to be the minimum of the pruned nodes value, that is, threshold=6 and the node wit 6 value along With node With value 4 are retained for further expansion.

The IDA* will find a solution of least cost or optimal solution (if one exists), if an admissible monotonic cost function is used. IDA* not only finds cheapest path to a solution but uses far space than A* , and it expands approximately the same number of nodes as that of A* in a search. An additional benefit of IDA* over A* is that it is simpler to implement, as there are open and closed lists to be maintained. A simple recursion performs DFS inside an outer loop to handle iterations.

Constraint Satisfaction

Constraint Satisfaction Many A1 problems can be viewed as problems of constraint satisfaction in which the goal is to solve some problem state that satisfies a

given set of constraints instead of finding optimal path to the solution. Such problems are called Constraint Satisfaction (CS) Problems. For example, some of the simple constraint satisfaction problems are cryptography, the n-Queen problem, map coloring. Crossword puzzle, etc.

A cryptography problem: A number puzzle in which a group of arithmetical operations has some or all of its digits replaced by letters and the original digits must be found. In such a puzzle, each letter represents a unique digit. Let us consider the following problem in which we have to replace each letter by a distinct digit (0—9) so that the resulting sum is correct.

	B	A	S	E
+	B	A	L	L
G	A	M	E	S

The n-Queen problem: The condition is that on the same row, or column, or diagonal no two queens attack each other.

A map coloring problem: Given a map, color regions of map using three colors, blue, red, and black such that no two neighboring countries have the same color.

In general, we can define a Constraint Satisfaction Problem as follows:

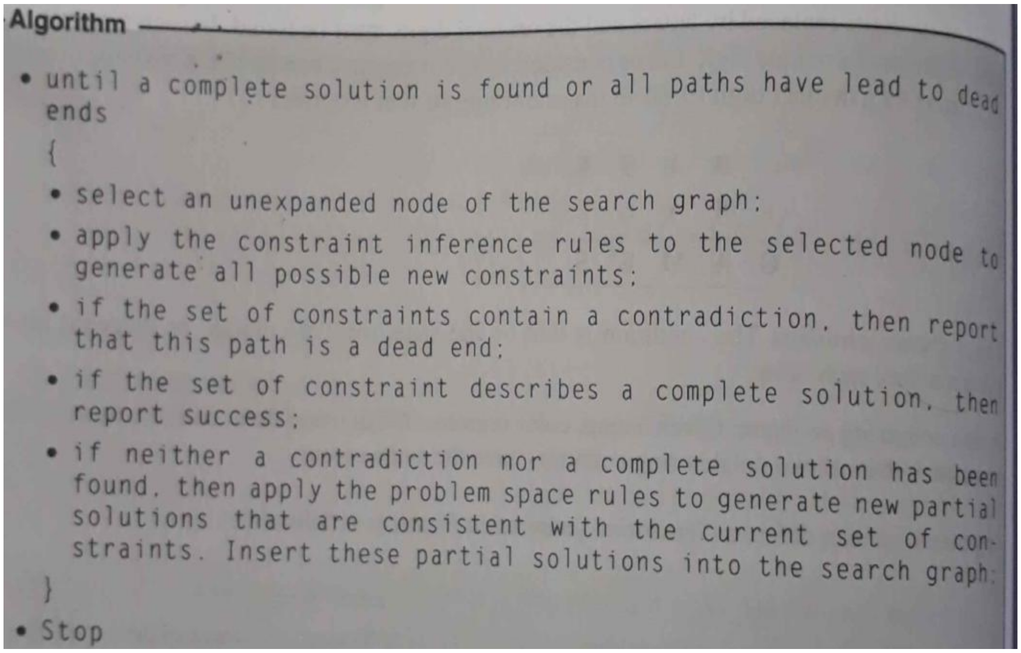
- a set of variables {xl, x2, xn), with each xi e Di with possible values and
- a set of constraints, i.e. relations, that are assumed to hold between the values of the variables.

The problem is to find, for each i, 1<=i<=n, a value of xi ∈ Di , so that all constraints are satisfied. A CS problem is usually represented as an undirected graph, called Constraint Graph in which the nodes are

the variables and the edges are the binary constraints.

- Start state: The empty assignments, i.e. all variables, are unassigned.
- Goal state: all the variables are assigned values which satisfy constraints.

- Operator: assigns value to any unassigned variable, provided that it does not conflict with previously assigned variables.



Algorithm

```
• until a complete solution is found or all paths have lead to dead
  ends
  {
    • select an unexpanded node of the search graph:
    • apply the constraint inference rules to the selected node to
      generate all possible new constraints:
    • if the set of constraints contain a contradiction, then report
      that this path is a dead end:
    • if the set of constraint describes a complete solution, then
      report success:
    • if neither a contradiction nor a complete solution has been
      found, then apply the problem space rules to generate new partial
      solutions that are consistent with the current set of constraints.
      Insert these partial solutions into the search graph:
  }
• Stop
```

Crypt-Arithmetic Puzzle Problem

example 1

Statement: Solve the following puzzle by assigning numeral (0—9) in such a way that each letter is assigned unique digit which satisfy the following addition:

	B	A	S	E
+	B	A	L	L
G	A	M	E	S

- Constraints: No two letters have the same value (the constraints of arithmetic).
- Initial Problem State

  G=? ;A=?;M=?;E=?;S=?;B=?;L=?;
- Apply constraint inference rules to generate the relevant new constraints.
- Apply the letter assignment rules to perform all assignments required by the current set of constraints. Then choose other rules to generate an additional assignment, which will. in turn, generate new constrains at the next cycle.
- At each cycle, there may be several choices of rules to apply.
- A useful heuristics can help to select the best rule to apply first

for example ,if letter that has only two possible values and another with six possible values ,then there is a better chance of guessing right on the first than on the second.

C4	C3	C2	C1		<-carrier
	B	A	S	E	
+	B	S	L	L	
G	A	M	E	S	

Constraint equation are

E+L=S

S+L+C1=E

A+A+C2=M

B+B+C3=A

G=C4

We can easily see that G is non zero value value ,so the value of carry C4 should be 1
hence G=1

now consider eq1 where E+L =S ,which is a carrying c1 then it can re-written as

E+L=S+10

E=S-L+10--→1

As S+L=E which is a carrying c2 then it can re-written BY Substitute the eq 1 in above equation

S+L=S-L+10

2L=10

L=5

CHARACTER	VALUE
G	1
B	
A	
M	
S	
L	5
E	

From E+L=S

=>S-E=L

now consider the possible range of vales of S and E i.e. in the pair

(S,E)=(5,0) -> NOT POSSIBLE AS L=5

(6,1) ->NOT POSSIBLE AS G=1

(7,2) -> POSSIBLE

(8,3) -> POSSIBLE

(9,4) -> POSSIBLE

So, now consider B+B=A then B>5 i.e, B=6,7,8,9

now consider S=7,E=2,B=6

C3+B+B=A

6+6=10+2

A=2 which is conflicting with E .So consider A=3

-----→ c2+A+A=M

1+3+3=7 which is conflicting with S?

So, consider different values of S,E and B

i.e, S=8,E=3 and B=7

C3+B+B=A

0+7+7=10+4 (C3=0)

So A=4

c2+A+A=M (c2=1)

1+4+4=9

whose values are not conflicting with any alphabet ,then the final state of alphabets are

CHARACTER	VALUE
G	1
B	7
A	4
M	9
S	8
L	5
E	3

1		1			<-carrier
	7	4	8	3	
+	7	4	5	5	
1	4	9	3	8	

example 2

C3	C2	C1		<-carrier
	T	W	O	
+	T	W	O	
F	O	U	R	

3 Game Playing

Since the beginning of AI Paradigin, game playing has been considered to he a major topic of AI as, it requires intelligence and has certain well-defined states and rules. a game is s defined as a 'sequence of choices where each choice is made from a number of discrete alternatives. Each sequence ends in a certain outcome and every outcome has a definite value for the opening player. Playing games by human experts against computers has always been a great fascination. We will consider only two-player games in which both the players have exactly opposite goals. Games can classify into two types: perfect information games and imperfect information games. Perfect information games are those in which both the players have access to the same information about the in progress; for example, Checker, Tic—Tac—Toe, Chess, Go, etc. On the other hand, in imperfect information games. Players do not have access to complete information about the game: for e\ample, games involving the use of cards (such as Bridge) and dice. We will restrict our study to discrete and perfect information games. A game is said to be discrete if it contains a finite number of states or configurations.

Game Problem versus State Space Problem

It should be noted that there is a natural correspondence between games and state space probie For example, in state space problems; we have a start state, intermediate states, rules or operators and a goal state. In game problems also we have a start state, legal moves, and winning positions, (goals). To further clarify the correspondence between the two problems the Comparisons are show in below table

State space problem	Game problem
State	Legal board position
Rule	Legal position
Goal	Winning position

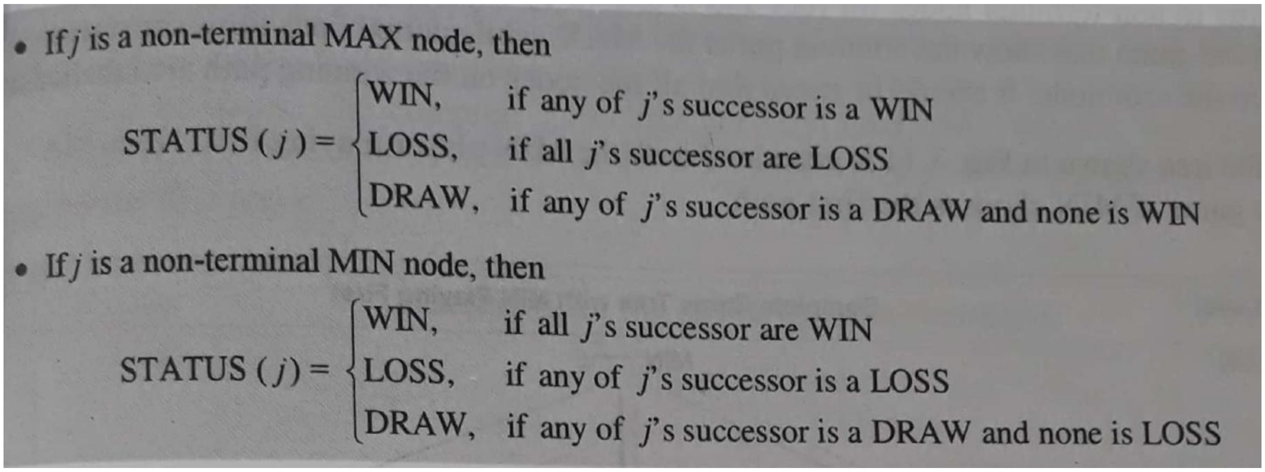
A game begins from a specified initial state and ends in a position that can be declared a win for one a loss for the other, or possibly a draw. A game tree is an explicit representation of all possible of the game. The root node is an initial position of the game. Its successors are the positions that the first player can reach in one move; their successors are the positions resulting from the second player's moves and so on. Terminal or leaf nodes are represented by WIN, LOSS, or DRAW. Each path from the root to a terminal node represents a different complete play of the game.

There is an obvious correspondence between a game tree and an AND—OR tree. The moves avail-able to one player from a given position can be represented by the OR nodes, whereas the move available to his opponent are the AND nodes. Therefore, game tree, one level is treated in the an OR node level and other as AND node level from one player's Point of view. On the other hand. in a general AND—OR tree, both types of nodes may be on the same level.

Game theory is based on the philosophy of minimizing the maximum possible loss and maximizing the minimum gain. In game playing involving computers, one player is assumed to be the computer, while the other is a human. During a game, two types of nodes are encountered, namely MAX and MIN. The MAX node will try to maximize its own game, while minimizing the opponent (MIN) game. Either of the two players, MAX and MIN, can play as the first player. We will assign the computer to be the MAX player and the opponent to be the MIN player. Our aim is t0 make the computer win the game by always making the best possible move at its turn. For this we have to look ahead at all possible moves in the game by generating the complete game tree and then decide which move is the best for MAX. As a part of game playing, game trees labeled ' MAX level and MIN level are generated alternately.

Status Labeling Procedure in Game Tree

We label each level in the game tree according to the player who makes the move at that point in the game. The leaf nodes are labeled as WIN, LOSS, or DRAW depending on whether they represent a win, loss or draw from MAX's point of view. Once the leaf nodes assigned their WIN—LOSS—DRAW status, each non-terminal node in the game tree can be labeled as WIN, LOSS, or DRAW by using the bottom-up process; this process is similar to the status labeling procedure used in case of the AND—OR graph. Status labeling procedure for a node with WIN, LOSS, or DRAW in case of game tree is given as follows:



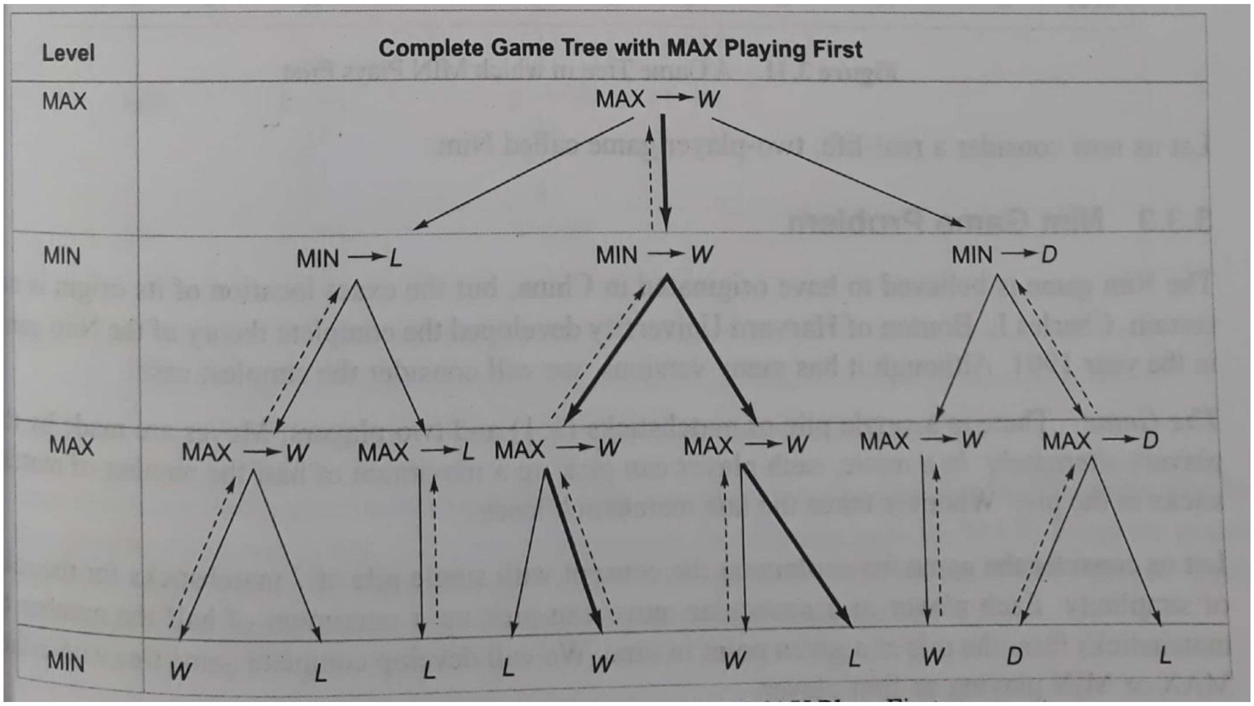
- If j is a non-terminal MAX node, then

$$\text{STATUS}(j) = \begin{cases} \text{WIN}, & \text{if any of } j\text{'s successor is a WIN} \\ \text{LOSS}, & \text{if all } j\text{'s successor are LOSS} \\ \text{DRAW}, & \text{if any of } j\text{'s successor is a DRAW and none is WIN} \end{cases}$$

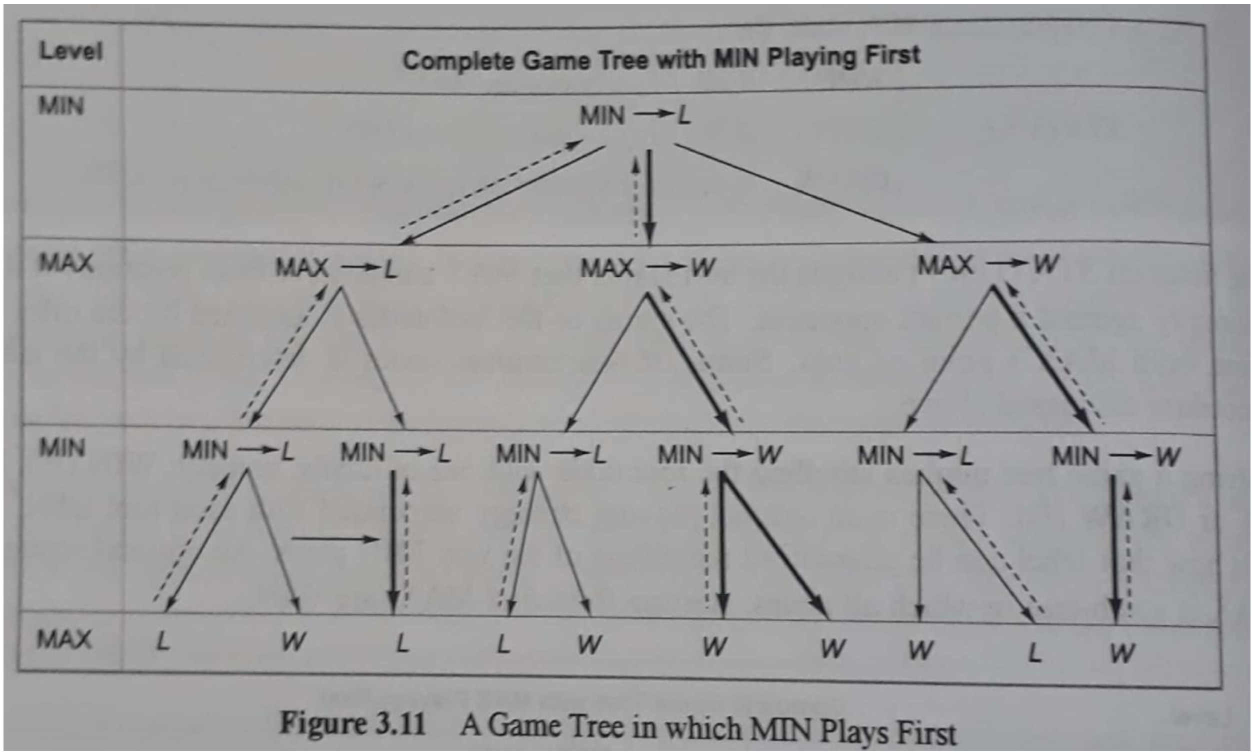
- If j is a non-terminal MIN node, then

$$\text{STATUS}(j) = \begin{cases} \text{WIN}, & \text{if all } j\text{'s successor are WIN} \\ \text{LOSS}, & \text{if any of } j\text{'s successor is a LOSS} \\ \text{DRAW}, & \text{if any of } j\text{'s successor is a DRAW and none is LOSS} \end{cases}$$

Solving a game tree implies labeling the root node with one of labels, namely. WIN (W), LOSS (L), or DRAW (D). There is an optimal playing strategy associated with each root label, which tells how that label can be guaranteed regardless of the way MIN plays. An optimal strategy for MAX is a sub-tree in which all nodes, starting from first MAX, or WIN.



Complete Game Tree with MAX Playing First



Complete Game Tree with MIN Playing First

Figure 3.11 A Game Tree in which MIN Plays First

Nim Game Problem

The Game There is a single pile of matchsticks (> l) and two players. Moves are made by the players alternately. In a move, each player can pick up a maximum of half the number of match sticks in the pile. Whoever takes the last matchstick loses.

Let us consider the game for explaining the concept with single pile of 7 matchsticks for the sake of simplicity. Each player in a particular move can pick up a maximum of half the number matchsticks from the pile at a given point in time. We will develop complete game tree with either MAX or MIN playing as first player.

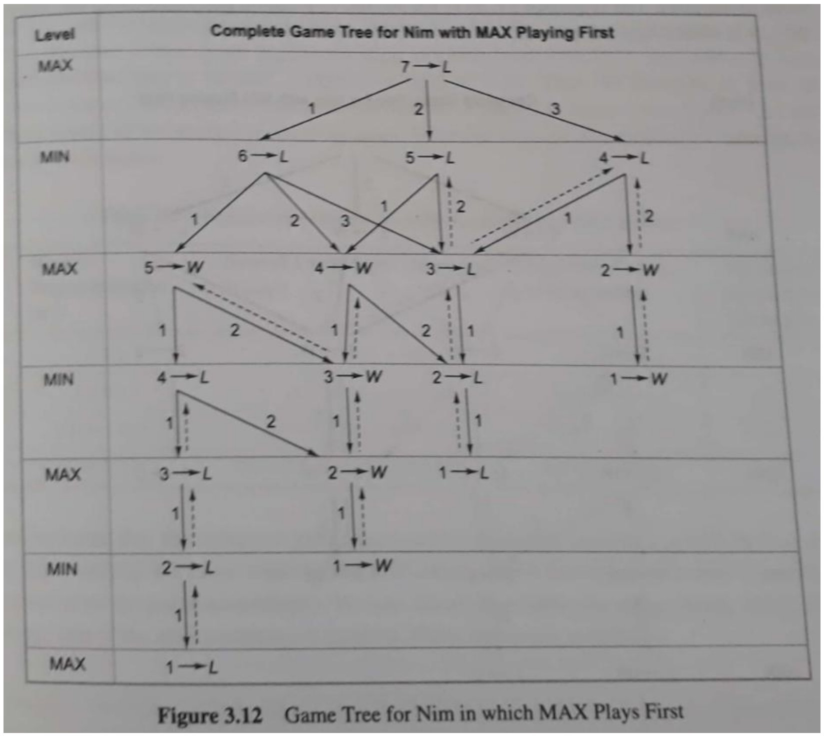


Figure 3.12 Game Tree for Nim in which MAX Plays First

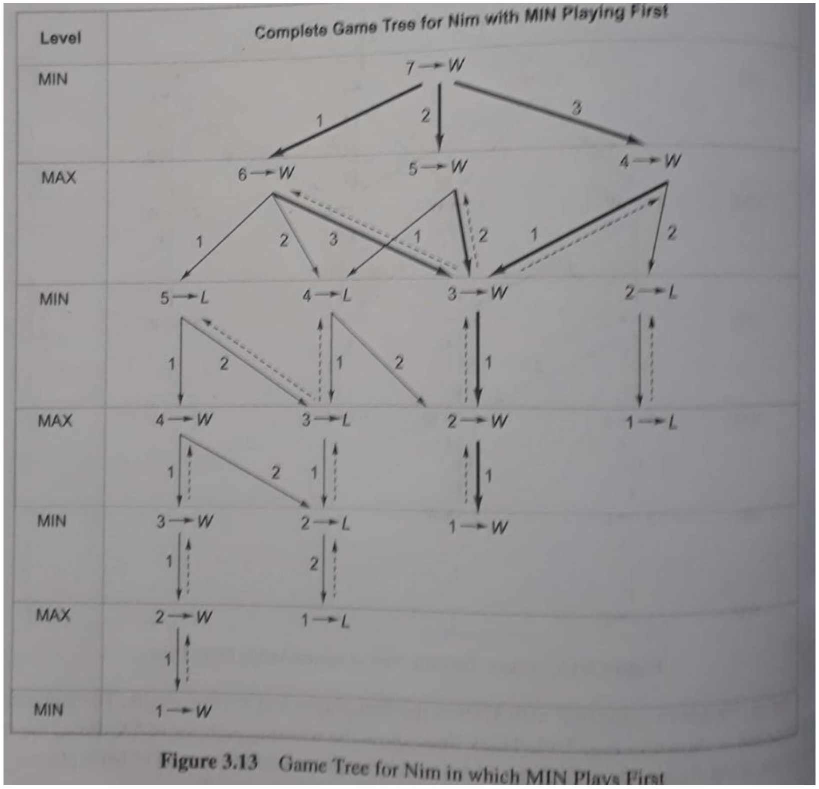


Figure 3.13 Game Tree for Nim in which MIN Plays First

STRATEGY

if at the time of MAX players turn their N matchsticks in a pile ,then MAX can force a win by leaving M matchsticks for the MIN player to play, where M ∈ {1,3,7,15,31,63….} using the rule of game .The sequence {1,3,7,15,31,63….} can be generated using the formula

$$2x_{i-1} + 1$$

Where $x_0 = 1$ for i>0

now we will formulate a method which will determine the number of matchsticks that have to be picked up by MAX player. There are 2 ways of finding the number.

1. the first method is to look up from the sequence {1,3,7,15,31,..} and figure out the closest number less than the given number N of match sticks in the pile .The difference between N and that number gives the desired number of sticks that have to be picked up.
   for example if N=45 the closest number to 45 in the sequence is 31,soer obtain the desired number of match sticks to be picked up as 14 on subtracting 31 from 45.In this case we have to maintain a sequence {1,3,7,15,31,..}.
2. The desired number is obtained by removing the most significant digit from the binary representation of N and adding it to the least significant digit position.

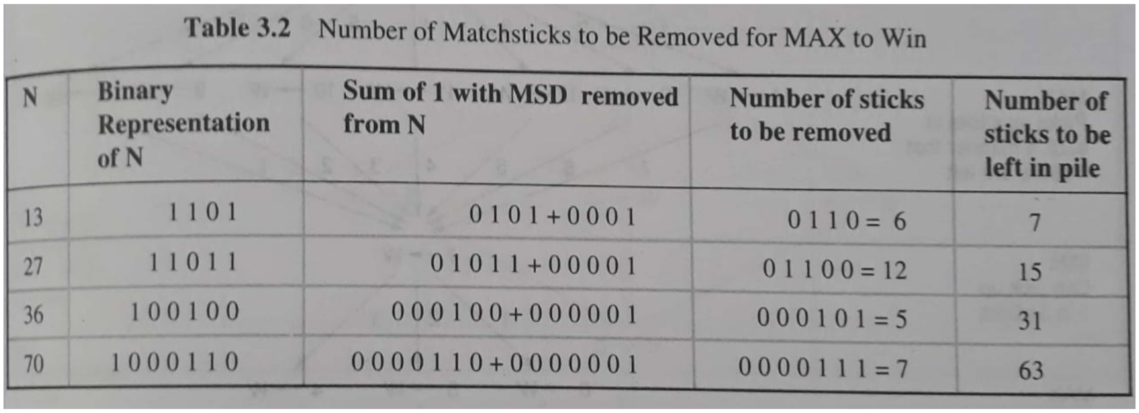


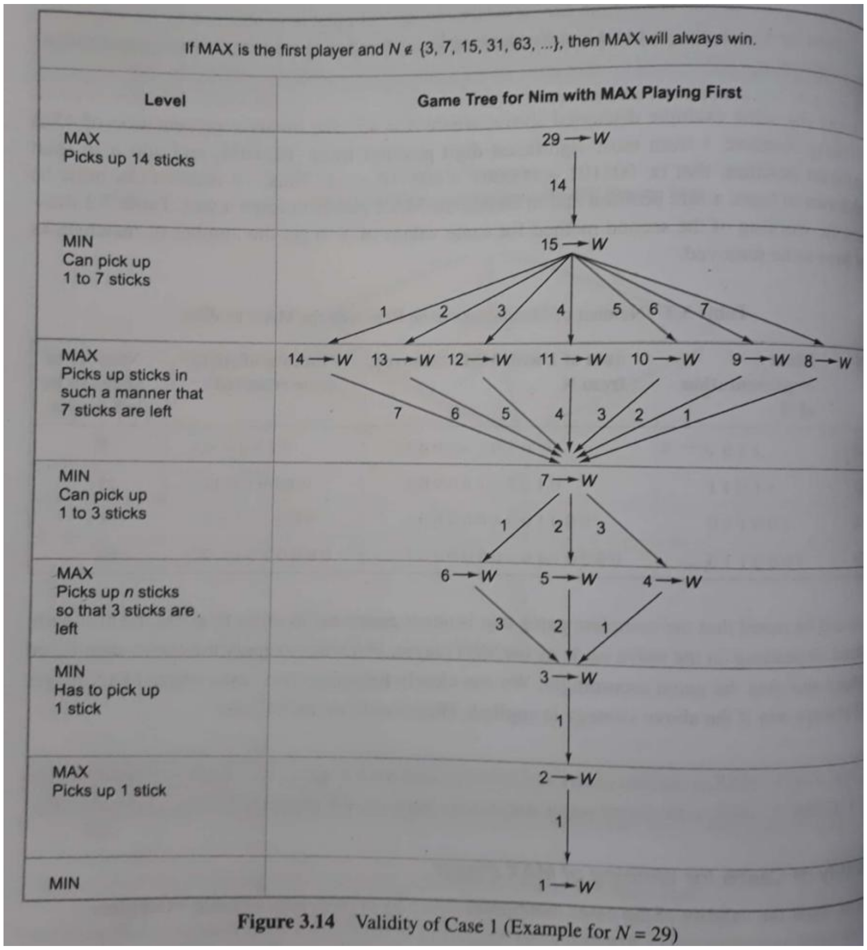
Table 3.2 Number of Matchsticks to be Removed for MAX to Win

N	Binary Representation of N	Sum of 1 with MSD removed from N	Number of sticks to be removed	Number of sticks to be left in pile
13	1101	0101+0001	0110= 6	7
27	11011	01011+00001	01100=12	15
36	100100	000100+000001	000101=5	31
70	1000110	0000110+0000001	0000111=7	63

example :consider N=45 ,the binary representation of 45 is (101101),remove 1 from most significant digit position and add it to the least significant position ie, 001101+000001=001110=14.Thus 14 match sticks must be removed to be in a safe position and enable to win MAX player to force a win.

case 1

if MAX is the first player and NØ {1,3,7,15,31,‥}. consider a pile of 29 sticks and let MAX be the first player and wins.



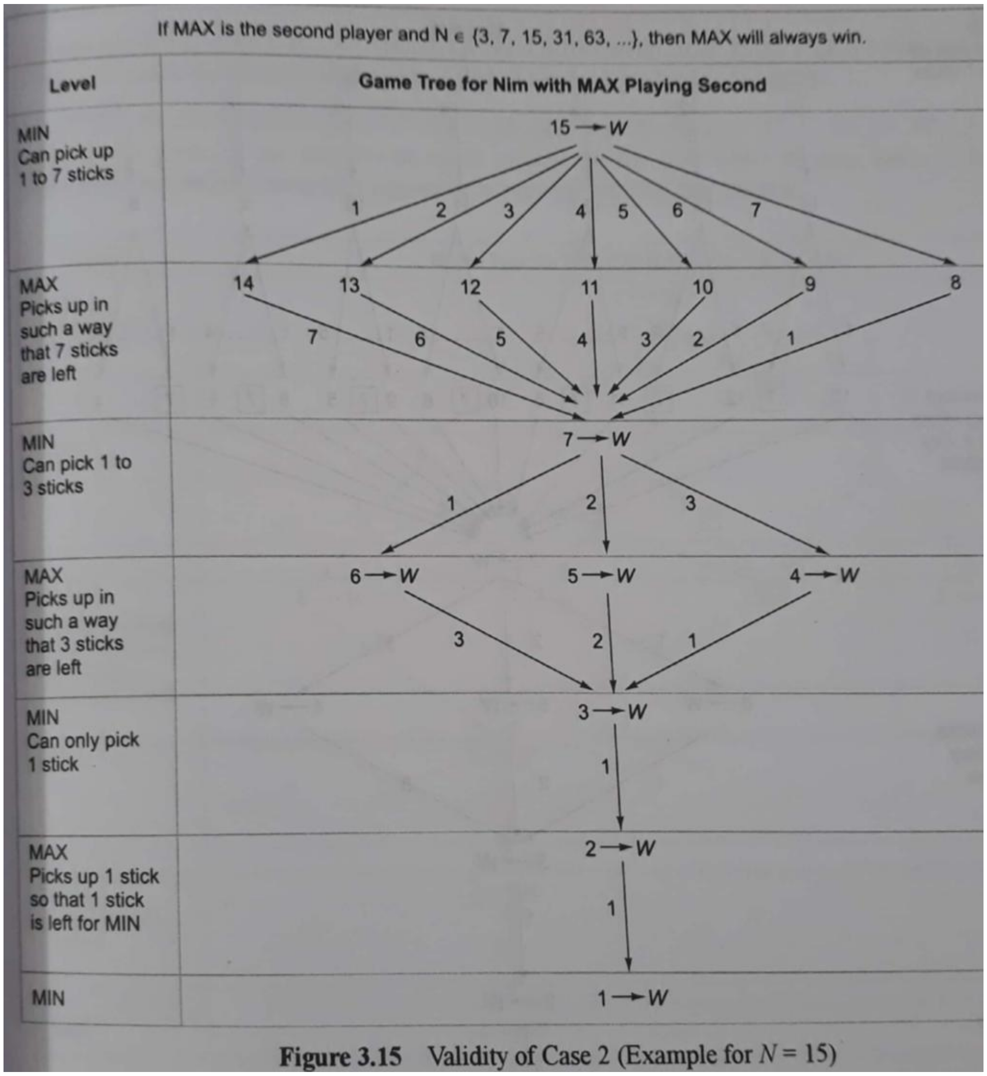
If MAX is the first player and $N \notin \{3, 7, 15, 31, 63, ...\}$, then MAX will always win.

Level	Game Tree for Nim with MAX Playing First
MAX Picks up 14 sticks	
MIN Can pick up 1 to 7 sticks	
MAX Picks up sticks in such a manner that 7 sticks are left	
MIN Can pick up 1 to 3 sticks	
MAX Picks up *n* sticks so that 3 sticks are left	
MIN Has to pick up 1 stick	
MAX Picks up 1 stick	
MIN	

Figure 3.14 Validity of Case 1 (Example for $N = 29$)

case 2

if MAX is the second player and N∈ {1,3,7,15,31,‥}. consider a pile of 15 sticks and MAX wins.


If MAX is the second player and N ∈ {3, 7, 15, 31, 63, ...}, then MAX will always win.

Level
Game Tree for Nim with MAX Playing Second
MIN Can pick up 1 to 7 sticks
15 → W
1
2
3
4
5
6
7
MAX Picks up in such a way that 7 sticks are left
14
13
12
11
10
9
8
7
6
5
4
3
2
1
MIN Can pick 1 to 3 sticks
7 → W
1
2
3
MAX Picks up in such a way that 3 sticks are left
6 → W
5 → W
4 → W
3
2
1
MIN Can only pick 1 stick
3 → W
1
MAX Picks up 1 stick so that 1 stick is left for MIN
2 → W
1
MIN
1 → W

Figure 3.15 Validity of Case 2 (Example for $N = 15$)

If MAX is the first player and N∈ {1,3,7,15,31,‥}. at the root of the game ,then MAX can fore to win,

If MAX can leave any player number out of the sequence (3,7,15,31,63,…}for MIN ,MAX might lose only in the case of getting 7

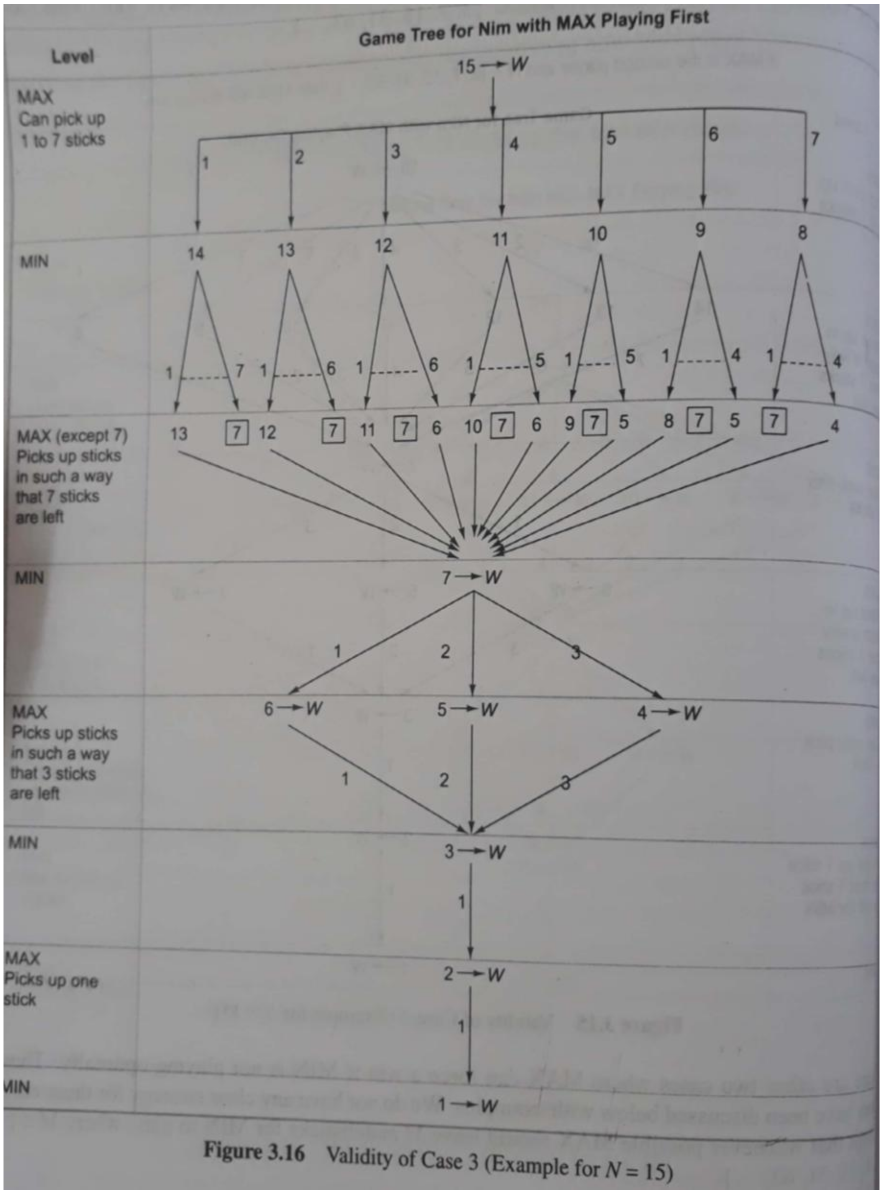


Figure 3.16 Validity of Case 3 (Example for $N = 15$)

Case 3 if Max is the first player and N∈ {1,3,7,15,31, . . }. at root of the game ,then MAX can force a win using the strategy mentioned above in all cases except when MAX gets a number from the sequence {1,3,7,15,31,..}. as its turns.

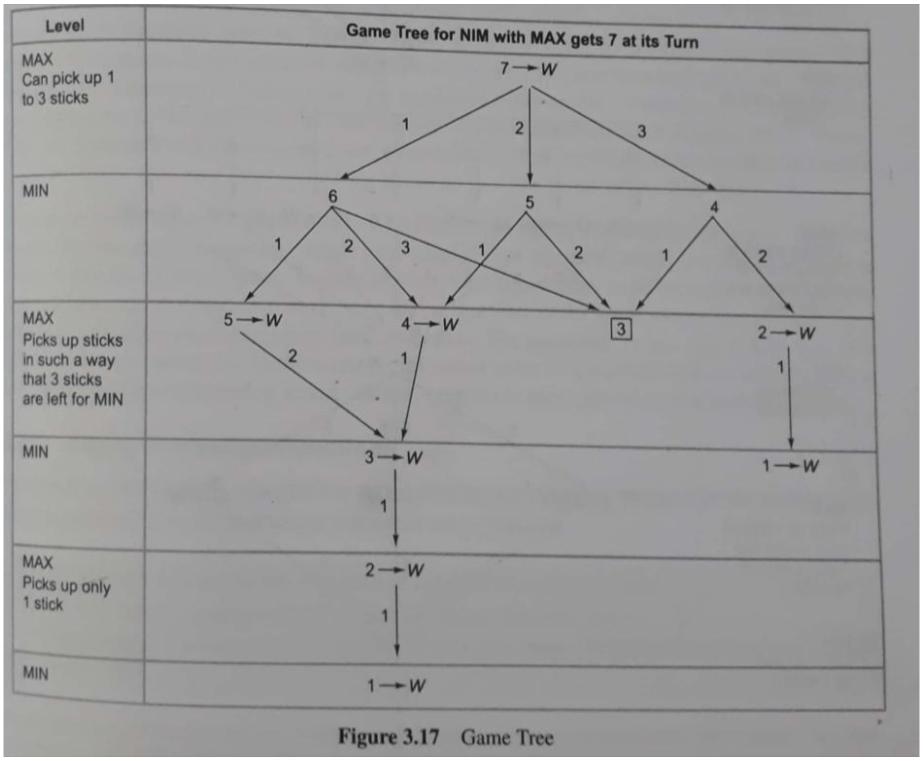
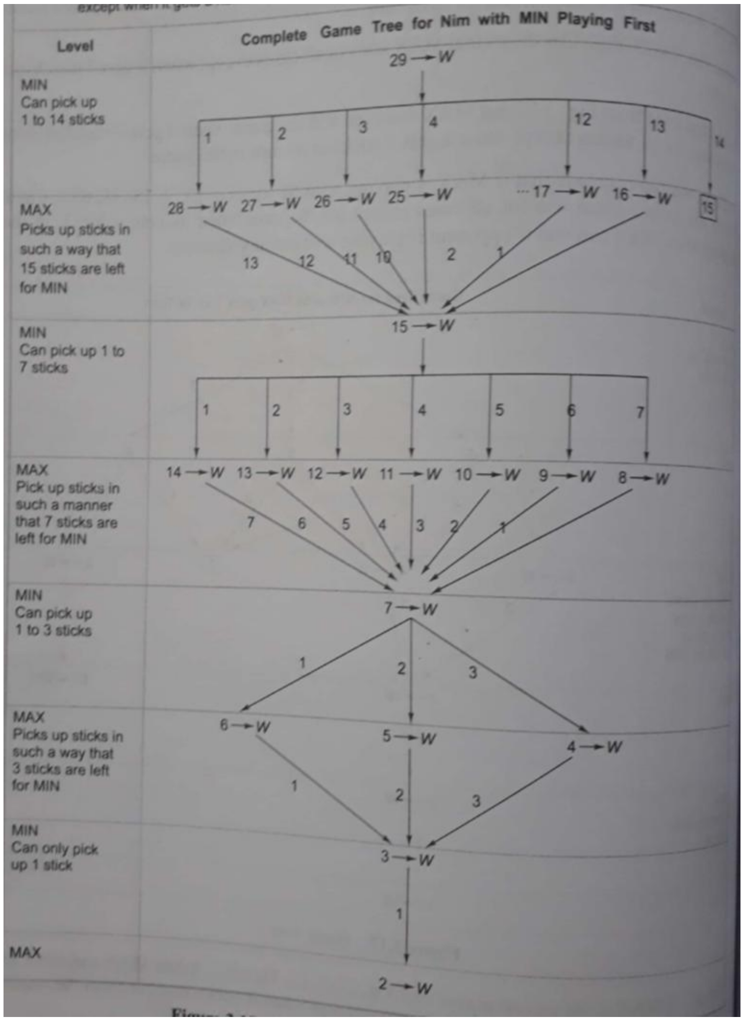


Figure 3.17 Game Tree

Case 4

if Max is the second player and NØ {1,3,7,15,31,..}..,then MAX can force a win using the strategy mentioned above in all cases except when MAX gets a number from the sequence {1,3,7,15,31,..}. as its turns.



Bounded Look-Ahead Strategy and Use of Evaluation Functions

If a player can develop the game tree to a limited extent before deciding on the move, then this shows that the player is looking ahead; this is called look-ahead strategy. If a player is looking ahead n number of levels before making a move, then the strategy is called n-move look-ahead.

Using Evaluation Functions

The process of evaluation of a game is determined by the structural features of the current state. The steps involved in the evaluation procedure are as follows:

- The first step is to decide which features are of value in a particular game.
- The next step is to provide each feature with a range of possible values.
- The last step is to devise a set of weights in order to combine all the feature values into a single value.

This function provides numerical assessment of how favorable the game state for MAX.

The general strategy for MAX is to play in such a manner that it maximizes its winning chances, while simultaneously minimizing the chances of opponent. The node which offers the best path is then chosen to make a move. for the sake of convince ,let us assume the root node to be MAX.

Consider the one-play and two play game as shown, the score leaf node is assumed to be calculated using evaluation function.

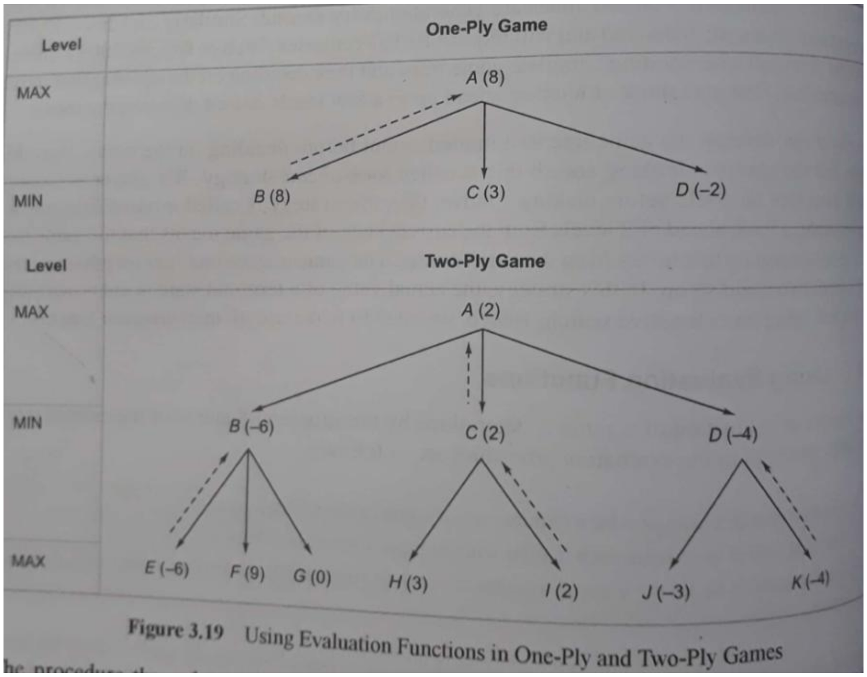
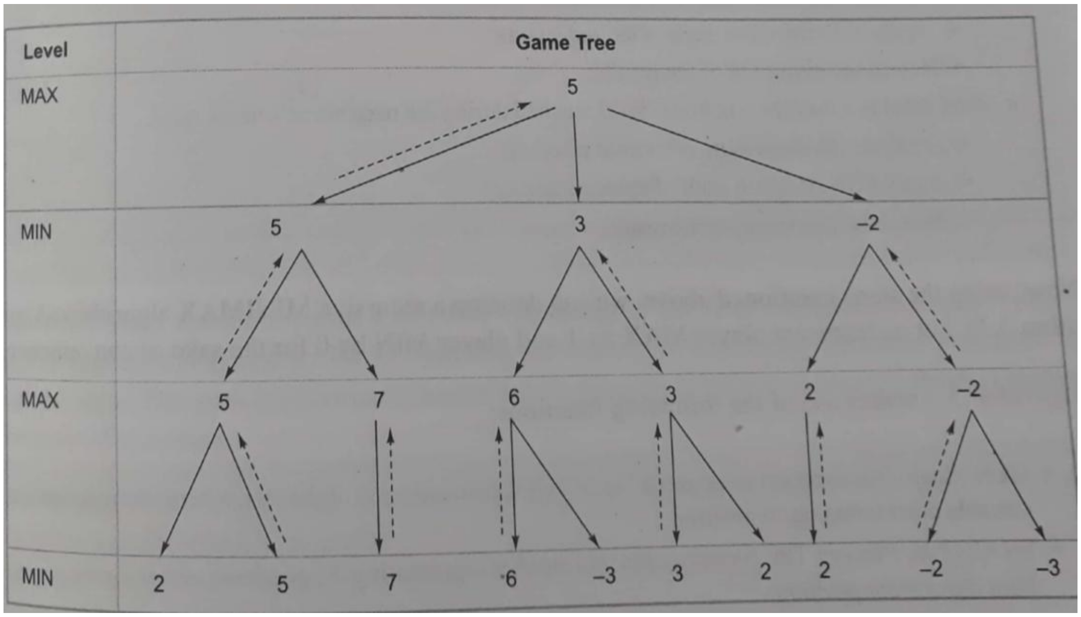


Figure 3.19 Using Evaluation Functions in One-Ply and Two-Ply Games

The procedure through which the scoring information travels up the game tree is called MINIMAX procedure. This procedure represents a recursive algorithm for choosing the next move in two-play game.

MINIMAX Procedure

The player hoping to achieve a positive number is called the maximizing player, while the opponent is called the minimizing player. At each move, the MAX player will try to take a path that leads to a large positive number; on the other hand, the opponent will try to force the game towards situations with strongly negative static evaluations.



MINIMAX Procedure

The algorithmic steps of this procedure may bc written follows:

- Keep on gcnerating tree till the limit, say depth d of the tree, has been
- Compute the static value ofthc leaf nodes at depth d from the current position of the game tree using evaluation function.
- Propagate the values till the current position on the basis of the MINIMAX strategy.

MINIMAX Strategy

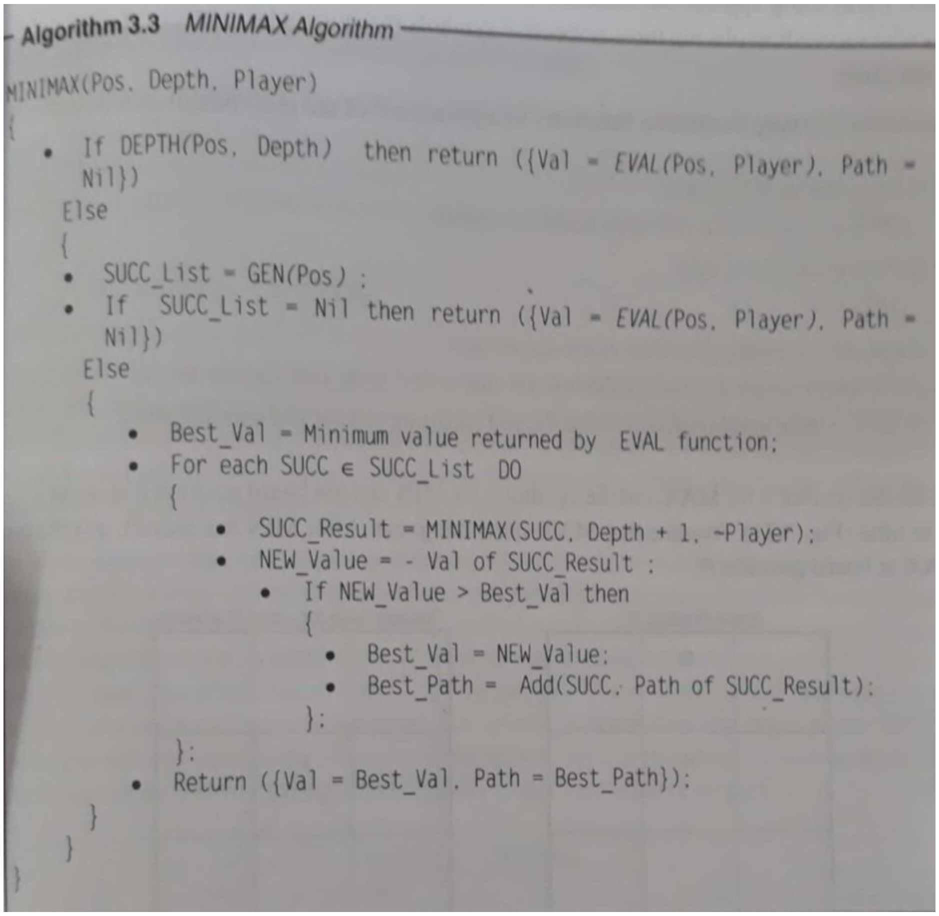
The steps in the MINIMAX strategy arc written as follows:

- If the level is minimizing level (level reached during the minimizer's turn), then
    - Generate the successors of the current position
    - Apply MINIMAX to each of the successors
    - Return the minimum of the results
- If the level is a maximizing level (level reached during the maximizer's turn), then
    - Generate the successors of current position
    - Apply MINIMAX to each of these successors
    - Return the maximum of the results

Algorithm makes use of the following functions:

- **GEN (Pos):** This function generates a list of SUccs (successors) of the Pos, where Pos represent a variable corresponding to position.
- **EVAL (Pos, Player):** This function returns a number representing the goodness of Pos for the player from the current position

•Depth(Pos,Depth): It is a Boolean function that returns true if the search has reached the maximum depth from the current position ,else it returns false.



Algorithm 3.3 *MINIMAX Algorithm*

```
MINIMAX(Pos, Depth, Player)
{
  • If DEPTH(Pos, Depth) then return ({Val = EVAL(Pos, Player), Path =
    Nil})
    Else
    {
    • SUCC_List = GEN(Pos);
    • If SUCC_List = Nil then return ({Val = EVAL(Pos, Player), Path =
      Nil})
      Else
      {
        • Best_Val = Minimum value returned by EVAL function;
        • For each SUCC ∈ SUCC_List DO
          {
            • SUCC_Result = MINIMAX(SUCC, Depth + 1, ~Player);
            • NEW_Value = - Val of SUCC_Result ;
            • If NEW_Value > Best_Val then
              {
                • Best_Val = NEW_Value;
                • Best_Path = Add(SUCC, Path of SUCC_Result);
              };
          };
        • Return ({Val = Best_Val, Path = Best_Path});
      }
    }
}
```

The MINIMAX function returns a structure consisting of Val field containing heuristic value of the current state obtained by EVAL function and Path field containing the entire path from the current state. This path is constructed backwards starting from the last element to the first element because of recursion.

Tic-tac-toe Game

Tic-tac-toe is a two-player game in which the players take turns one by one and mark the in a 3 x 3 grid using appropriate symbols. One player uses 'o' and other uses player succeeds in placing three respective symbols in a horizontal, vertical, or diagonal row wins the game.

Let us define the static evaluation functionfto a position P of the grid (board) as follows.
• If P is a win for MAX. then
f(P) = n. (where n is a very large positive number)
• If P is a for MIN, then

f(P) = -n

- If P is not a winning position for either player, then

f(P) = (total number of rows, columns, and diagonals that are still open for
- (total number of rows, columns, and diagonals that are still open for MIN)

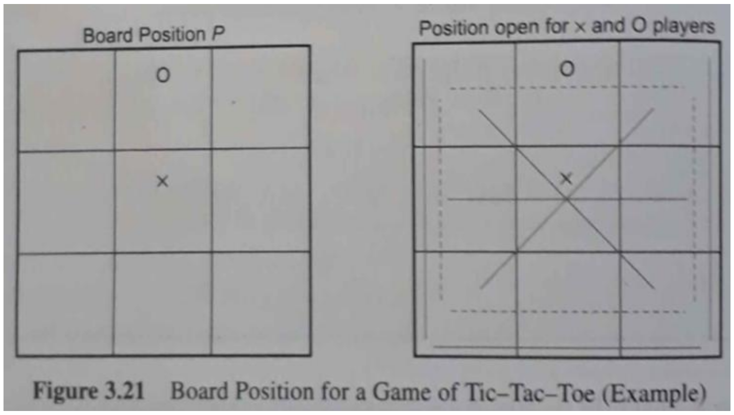


Figure 3.21 Board Position for a Game of Tic–Tac–Toe (Example)

- Total number of rows, columns, and diagonals still open for MAX (thick lines) = 6
- Total number of rows, columns, and diagonals still open for MIN (dotted lines) = 4

Alpha-Beta Pruning

The strategy used to reduce the number of tree branches explored and the number of static evaluation applied is known as alpha—beta pruning. This procedure is also called backward pruning, which is a modified depth-first generation procedure. The purpose of applying this procedure is to reduce the amount of work done in generating useless nodes and is based on common sense or basic logic.

The alpha—beta pruning procedure requires the maintenance of two threshold values: one representing a lower bound (a) on the value that a maximizing node may ultimately be assigned (we call this alpha) and another representing upper bound (β) on the value that a minimizing node may be assigned (we call it beta).

Level	First step
MAX	A
	↓
MIN	(<=2)B
	↑ ↓
MAX	(2)D

Assume that the evaluation function generates a = 2 for state D. At this point, the upper
2 at state B and is shown as <=2.
After the first step, we have to backtrack and generate another state E from B in the SECOND step as shown in Fig. 3.23.

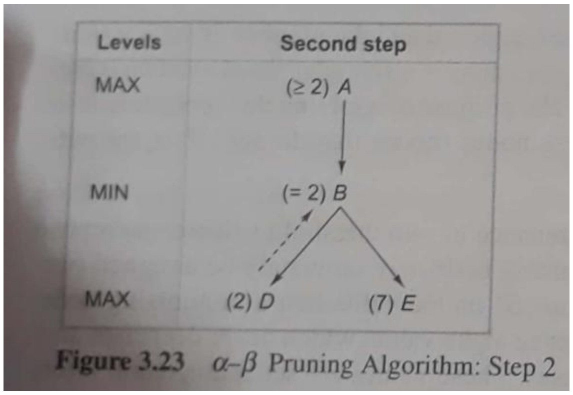


Figure 3.23 α–β Pruning Algorithm: Step 2

The state E gets a = 7 and since there is no further Successor of B (assumed), the β value at state B becomes equal to 2. Once the β value is fixed at MIN level the lower bound a = 2 gets propagated to state A as => 2.

In the third step, expand A to another successor C, and then expand C's successor to F with a= 1From Fig. 3.24 we note that the value at state C is <=1 and the value of a root A cannot be less then 2; the path from A through C is not useful and thus further expansion of C is pruned. Therefore there is no need to explore the right side of the tree fully as that result is not going to alter the move decision. Since there are no further successors of A (assumed), the value of root is fixed as 2 , that is, a= 2

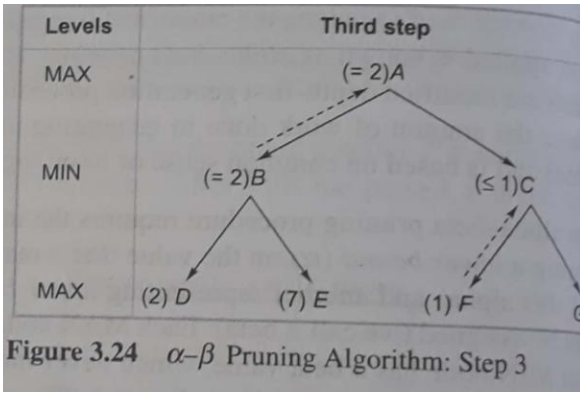


Figure 3.24 α–β Pruning Algorithm: Step 3

The complete generation of $(a - \beta)$pruning algorithm is shown as below

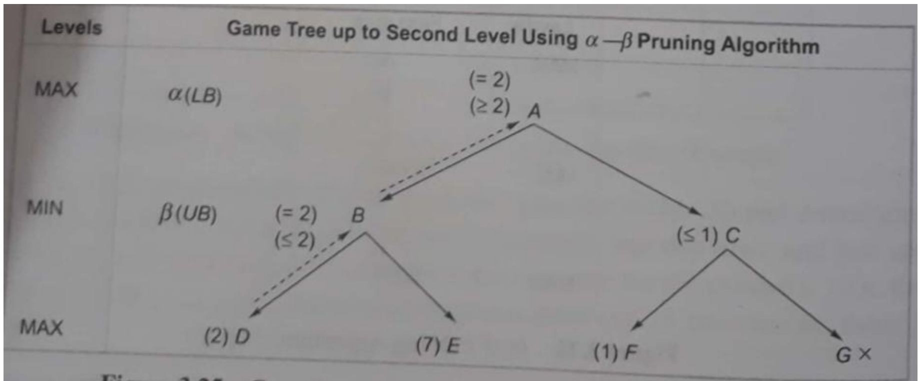


Figure 3.25 Game Tree Generation using α–β Pruning Algorithm

Refinements to(a −Q) Pruning

In addition to (a −β)pruning. a number of modifications can be made to the MINIMAX procedure in order to improve its performance. One of the important factors that need to be considered during a search is when to stop going deeper in the search tree. Further, the idea behind a---P pruning procedure can he extended by cutting-off additional paths that appear to be slight Improvements over paths that have already been explored.

pruning of Slightly Better Paths

in the below figure we see that the value 4.2 is only slightly better than 4, so we may terminate further exploration of node C. Terminating exploration of a sub-tree that offers little possibility for improvement over known paths is called futility cut off.

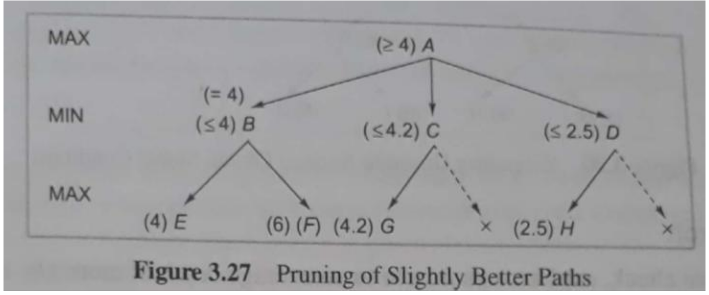


Figure 3.27 Pruning of Slightly Better Paths

Alternative to (a −Q)pruning MINIMAX Procedure

The MINIMAX procedure still has some problems even with all the refinements discussed above; It is based on the assumption that the opponent will always choose an optimal move. In a winning situation, this assumption is acceptable but in a losing situation, one may try other options and gain some benefit in case the opponent makes a mistake (Rich & Knight, 2003). Suppose, we have to choose one move out of two possible moves, both of which may lead to bad situations for us if the opponent plays perfectly. MINIMAX procedure will always choose the bad move out of the two: however, here we can choose an option which is slightly less bad than the other. This is based on the assumption that the less bad move could lead to a good situation for us if the opponent makes a single mistake. Similarly, there might be a situation when one move appears to be only slightly more advantageous than the other. Then, it might be better to choose the less advantageous move. To implement such systems, we should have a model of individual playing styles of opponents.

Iterative Deepening

Rather than searching till a fixed depth in a given game tree, it is advisable to first search ohnley. one-ply, then apply MINIMAX to two-ply, then three-ply till the final goal state is searched . There is a good reason why iterative deepening is popular in cast of games such as chess and others programs. In competitions, there is an average amount of time allowed per move. The idea that enables us to conquer this constraint is to do as much look-ahead as can be done in the available time. we use iterative deepening, we can keep on increasing the look-ahead depth until we run out of time. We can arrange to have a record of the best move for given look-ahead even if we have to interrupt our attempt to go one level deeper. This could not 'X done using (unbounded) depth-first search. With effective ordering, $(a - Q)$ pruning MINIMA algorithm can prune many branches and the total search time can be decreased.

Two player perfect information games

1. Chess
2. Checkers
3. Othello(reverse)