UNIT-2 Game Playing in Artificial Intelligence

Game Playing is an important domain of artificial intelligence. Games don't require much knowledge; the only knowledge we need to provide is the rules, legal moves and the conditions of winning or losing the game.

Both players try to win the game. So, both of them try to make the best move possible at each turn. Searching techniques like BFS(Breadth First Search) are not accurate for this as the branching factor is very high, so searching will take a lot of time. So, we need another search procedures that improve –

- **Generate procedure** so that only good moves are generated.
- **Test procedure** so that the best move can be explored first.

The most common search technique in game playing is **Minimax search procedure**. It is depth-first depth-limited search procedure. It is used for games like chess and tic-tac-toe.

2.1 MIN-MAX Search

Games have always been an important application area for heuristic algorithms. In playing games whose state space may be exhaustively delineated, the primary difficulty is in accounting for the actions of the opponent. This can be handled easily by assuming that the opponent uses the same knowledge of the state space as us and applies that knowledge in a consistent effort to win the game. Minmax implements game search under referred to as MIN and MAX.

The min max search procedure is a depth first, depth limited search procedure. The idea is to start at the current position and use the plausible move generator to generate the set of possible successor positions. To decide one move, it explores the possibilities of winning by looking ahead to more than one step. This is called a ply. Thus in a two ply search, to decide the current move, game tree would be explored two levels farther.

Consider the below example

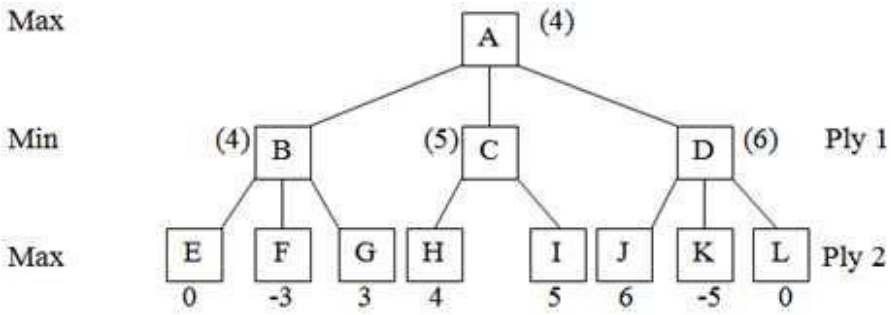


Figure Tree showing two ply search

In this tree, node A represents current state of any game and nodes B, C and D represent three possible valid moves from state A. similarly E, F, G represents possible moves from B, H, I from C and J, K, L, from D. to decide which move to be taken from A, the different possibilities are explored to two next steps. 0, -3, 3, 4, 5, 6, -5, 0 represent the utility values of respective move. They indicate goodness of a move. The utility value is back propagated to ancestor node, according to situation whether it is max ply or min ply. As it is a two player game, the utility value is alternatively maximized and minimized. Here as

the second player's move is maximizing, so maximum value of all children of one node will be back propagated to node. Thus, the nodes B, C, D, get the values 4, 5, 6 respectively. Again as ply 1 is minimizing, so the minimum value out of these i.e. 4 is propagated to A. then from A move will be taken to B.

MIN MAX procedure is straightforward recursive procedure that relies on two auxiliary procedures that are specific to the game being played.

1. MOVEGEN (position, player): the move generator which returns a list of nodes representing the moves that can be made by player in position. We may have 2 players namely PLAYER-TWO in a chess problem.

2. STATIC (position, player): the static evaluation function, which returns a number representing the goodness of position from the standpoint of player.

We assume that MIN MAX returns a structure containing both results and that we have two functions, VALUE and PATH that extract the separate components. A function LAST PLY is taken which is assumed to evaluate all of the factors and to return TRUE if the search should be stopped at the current level and FALSE otherwise.

MIN MAX procedure takes three parameters like a board position, a current depth of the search and the players to move. So the initial call to compute the best move from the position CURRENT should be

MIN MAX (CURRENT, 0, PLAYER-ONE)

(If player is to move)

Or

MIN MAX (CURRENT, 0, PLAYER-TWO)
(If player two is to move)
Let us follow the algorithm of MIN MAX

Algorithm: MINMAX (position, depth, player)

1. If LAST PLY (position, depth)

   Then RETURN VALUE =

   STATIC (position, player)

   PATH = nil.

2. Else, generate one more ply of the tree by calling the function MOVE_GEN (position, player) and set SUCCESORS to the list it returns.

3. If SUCESSORS is empty,

   THEN no moves to be made

   RETURN the same structure that would have been returned if LAST_PLY had returned TRUE.

4. If SUCCESORS is not empty,

   THEN examine each element in turn and keep track of the best one.

5. After examining all the nodes,

   RETURN VALUE = BEST- SCORE

   PATH = BEST- PATH

When the initial call to MIN MAX returns, the best move from CURRENT is the first element in the PATH.

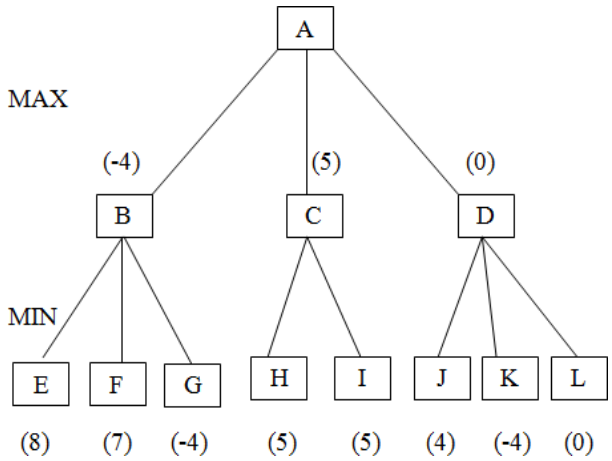
2.2 Alpha- Beta (α-β) Pruning

When a number of states of a game increase and it cannot be predicted about the states, then we can use the method pruning. Pruning is a method which is used to reduce the no. of states in a game. Alpha- beta is one such pruning technique. The problem with minmax search is that the number of game states it has to examine is exponential in the number of moves. Unfortunately we cannot eliminate the exponent, but we can effectively cut it in half. Alpha-beta pruning is one of the solutions to the problem of minmax search tree. When α-β pruning is applied to a standard minmax tree, it returns the same move as minmax would, but prunes away branches that cannot possibly influence the final decision.

The idea of alpha beta pruning is very simple. Alpha beta search proceeds in a depth first fashion rather than searching the entire space. Generally two values, called alpha and beta, are created during the search. The alpha value is associated with MAX nodes and the beta value is with MIN values. The value of alpha can never decrease; on the other hand the value of beta never increases. Suppose the alpha value of A MAX node is 5. The MAX node then need not consider any transmitted value less than or equal to 5 which is associated with any MIN node below it. Alpha is the worst that MAX can score given that MIN will also do its best. Similarly, if a MIN has a beta value of 5, it need not further consider any MAX node below it that has a value of 6 or more.

The general principal is that: consider a node η somewhere in the search tree, such that player has a choice of moving to that node. If player has a better choice K either at the parent node of η or at any choice point further up, then η will never be reached in actual play. So once we have found out enough about η (by examining some of its descendents) to reach this conclusion, we can prune it.

We can also say that "α" is the value of the best choice we have found so far at any choice point along the path for MAX. Similarly "β" is the value of the best choice we have found so far at any choice point along the path for MIN. Consider the following example



Figure

Here at MIN ply, the best value from three nodes is - 4, 5, 0. These will be back propagated towards root and a maximizing move 5 will be taken. Now the node E has the value 8 is far more, then accepted as it is minimizing ply. So, further node E will not be explored. In the situation when more plies are considered, whole sub tree below E will be pruned. Similarly if $\alpha=0$, $\beta=7$, all the nodes and related sub trees having value less than 0 at maximizing ply and more than 7 at minimizing ply will be pruned.

Alpha beta search updates the value of α and β as it goes along and prunes the remaining branches at a node as soon as the value of the current node is known to be worse than the current α and β value for MAX or MIN respectively. The effectiveness of alpha- beta pruning is highly dependent on the order in which the successors are examined suppose in a search tree the branching factor is x and depth d. the α-β search needs examining only $x^{d/2}$ nodes to pick up best move, instead of x^{d} for MINMAX.

2.3 The water jug problem :

There are two jugs called four and three ; four holds a maximum of four
gallons and three a maximum of three gallons. How can we get 2 gallons in the jug four. The state space is a set of ordered pairs giving the number of gallons in the pair of jugs at any time ie (four, three) where four = 0, 1, 2, 3, 4 and three = 0, 1, 2, 3. The start state is (0,0) and the goal state is (2,n) where n is a don't care but is limited to three holding from 0 to 3 gallons. The major production rules for solving this problem are shown below:

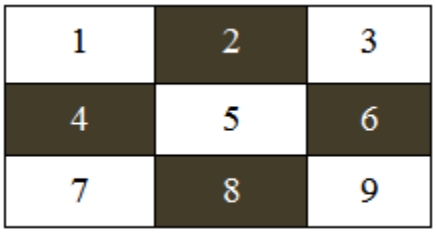
Initial condition goal comment
1 (four,three) if four < 4 (4,three) fill four from tap
2 (four,three) if three< 3 (four,3) fill three from tap
3 (four,three) If four > 0 (0,three) empty four into drain
4 (four,three) if three > 0 (four,0) empty three into drain
5 (four,three) if four+three<4 (four+three,0) empty three into four
6 (four,three) if four+three<3 (0,four+three) empty four into three
7 (0,three) If three>0 (three,0) empty three into four
8 (four,0) if four>0 (0,four) empty four into three
9 (0,2) (2,0) empty three into four
10 (2,0) (0,2) empty four into three
11 (four,three) if four<4 (4,three-diff) pour diff, 4-four, into four from three
12 (three,four) if three<3 (four-diff,3) pour diff, 3-three, into three from four and
a solution is given below Jug four, jug three rule applied
0 0
0 3 2
3 0 7
3 3 2
4 2 11
0 2 3
2 0 10

2.4 Chess Problem

Definition:

It is a normal chess game. In a chess problem, the start is the initial configuration of chessboard. The final state is the any board configuration, which is a winning position for any player. There may be multiple final positions and each board configuration can be thought of as representing a state of the game. Whenever any player moves any piece, it leads to different state of game.

Procedure:



A 3x3 Chess board

Figure

The above figure shows a 3x3 chessboard with each square labeled with integers 1 to 9. We simply enumerate the alternative moves rather than developing a general move operator because of the reduced size of the problem. Using a predicate called move in predicate calculus, whose parameters are the starting and ending squares, we have described the legal moves on the board. For example, move (1, 8) takes the knight from the upper left-hand corner to the middle of the bottom row. While playing Chess, a knight can move two squares either horizontally or vertically followed by one square in an orthogonal direction as long as it does not move off the board.

The all possible moves of figure are as follows.
Move (1, 8) move (6, 1)
Move (1, 6) move (6, 7)
Move (2, 9) move (7, 2)
Move (2, 7) move (7, 6)
Move (3, 4) move (8, 3)
Move (3, 8) move (8, 1)
Move (4, 1) move (9, 2)
Move (4, 3) move (9, 4)

The above predicates of the Chess Problem form the knowledge base for this problem. An unification algorithm is used to access the knowledge base. Suppose we need to find the positions to which the knight can move from a particular location, square 2. The goal move (z, x) unifies with two different predicates in the knowledge base, with the substitutions {7/x} and {9/x}. Given the goal move (2, 3), the responsible is failure, because no move (2, 3) exists in the knowledge base.

Comments:

_ In this game a lots of production rules are applied for each move of the square on the chessboard.
_ A lots of searching are required in this game.
_ Implementation of algorithm in the knowledge base is very important.

Puzzles(Tiles) Problem

Definition:
"It has set off a 3x3 board having 9 block spaces out of which 8 blocks having tiles bearing number from 1 to 8. One space is left blank. The tile adjacent to blank space can move into it. We have to arrange the tiles in a sequence for getting the goal state".

Procedure:
The 8-puzzle problem belongs to the category of "sliding block puzzle" type of problem. The 8-puzzle is a square tray in which eight square tiles are placed. The remaining ninth square is uncovered. Each tile in the tray has a number on it. A tile that is adjacent to blank space can be slide into that space. The game consists of a starting position and a specified goal position. The goal is to transform the starting position into the goal position by sliding the tiles around. The control mechanisms for an 8-puzzle solver must keep track of the order in which operations are performed, so that the operations can be undone one at a time if necessary. The objective of the puzzles is to find a sequence of tile movements that leads from a starting configuration to a goal configuration such as two situations given below.

Figure (Starting State)

1	2	3
4	5	6
	7	8

(Goal State)

1	2	3
4	5	6
7	8	-

The state of 8-puzzle is the different permutation of tiles within the frame. The operations are the permissible moves up, down, left, right. Here at each step of the problem a function f(x) will be defined which is the combination of g(x) and h(x).
i.e. F(x)=g(x) + h (x)
Where g _x_: how many steps in the problem you have already done or the current state from the initial state.
h _x_: Number of ways through which you can reach at the goal state from the current state or
Or
F(x)=g(x) + h (x)
h _x_is the heuristic estimator that compares the current state with the goal state note down how many states are displaced from the initial or the current state. After calculating the f (x) value at each step finally take the smallest f (x) value at every step and choose that as the next current state to get the goal state.

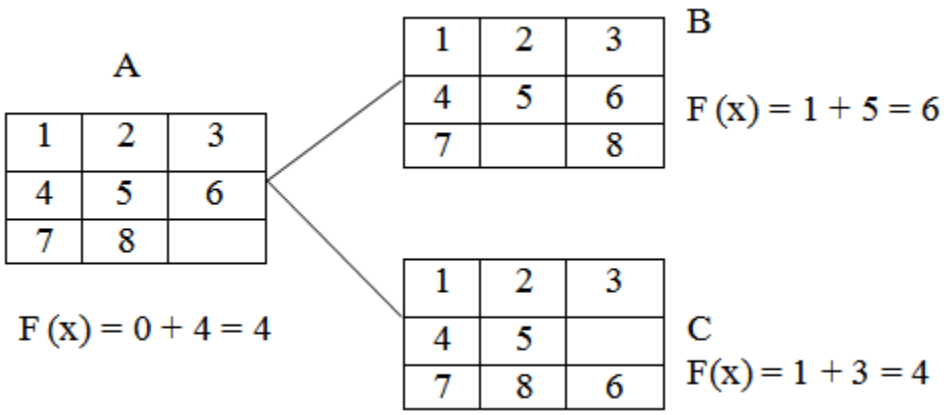
Let us take an example.

Figure (Initial State)

1	2	3
4	5	6
7	8	

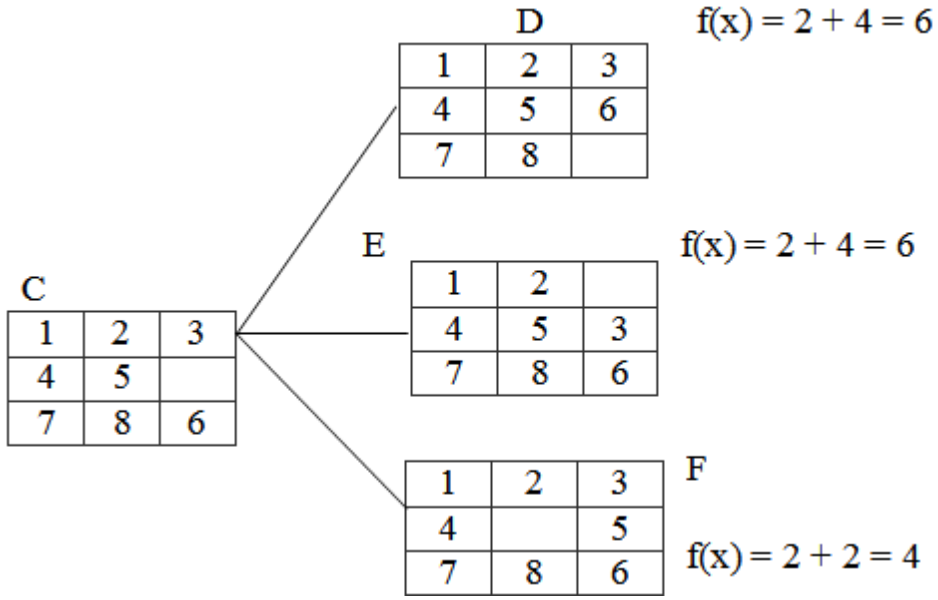
(Goal State)

1	2	3
4	8	5
	7	6



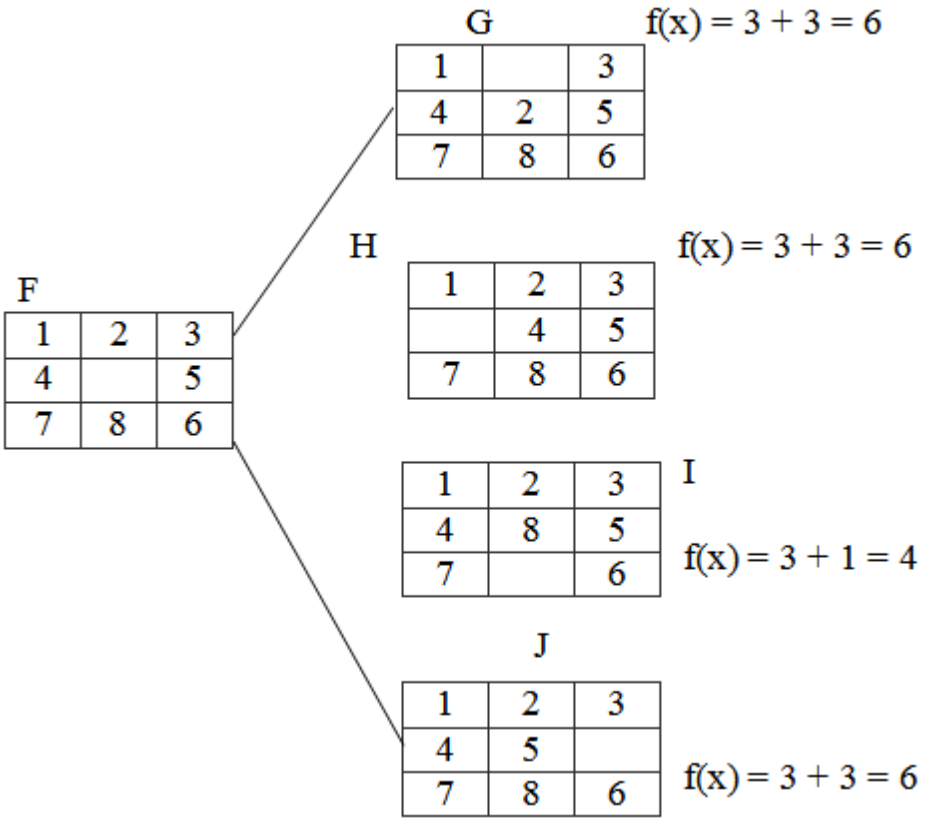
Step1:
f _x_is the step required to reach at the goal state from the initial state. So in the tray either 6 or 8 can change their portions to fill the empty position. So there will be two possible current states namely B and C. The f (x) value of B is 6 and that of C is 4. As 4 is the minimum, so take C as the current state to the next state.

Step 2:
In this step, from the tray C three states can be drawn. The empty position will contain either 5 or 3 or 6. So for three different values three different states can be obtained. Then calculate each of their f (x) and

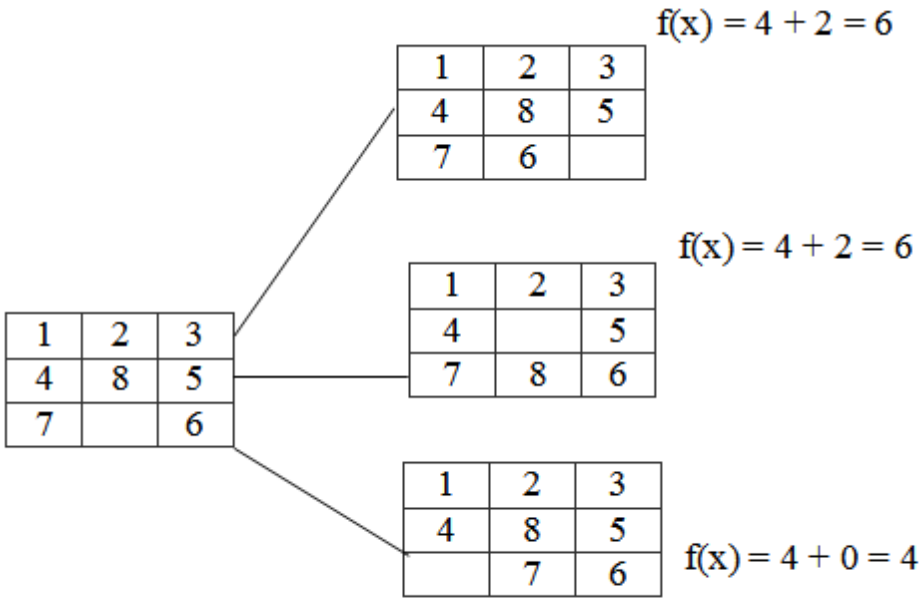


take the minimum one.
Here the state F has the minimum value i.e. 4 and hence take that as the next current state.

Step 3:
The tray F can have 4 different states as the empty positions can be filled with b4 values i.e.2, 4, 5, 8.



Step 4:
In the step-3 the tray I has the smallest f (n) value. The tray I can be implemented in 3 different states because the empty position can be filled by the members like 7, 8, 6.



Hence, we reached at the goal state after few changes of tiles in different positions of the trays.
Comments:
□□This problem requires a lot of space for saving the different trays.
□□Time complexity is more than that of other problems.
□□The user has to be very careful about the shifting of tiles in the trays.
□□Very complex puzzle games can be solved by this technique.