4.4.2 The Alpha-Beta Cutoff Procedure

The MINIMAX algorithm, presented above, requires expanding the entire state-space. This is a severe limitation, especially for problems with a large state-space. To handle this difficulty, an alternative approach is to evaluate heuristically the status of the next ply move of the player, to select a current move by the same player. We will demonstrate the process of computation of the heuristic measures of a node below with the well-known tic-tac-toe game.

Consider a heuristic function e(n) [3], [5] at node n that evaluates the difference of possible winning lines of the player and his opponent. Formally,

$$e(n) = M(n) - O(n)$$

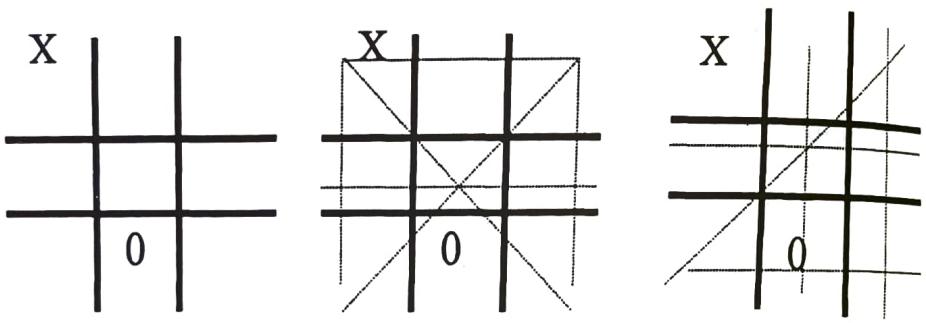
where M (n) = number of my possible winning lines
and O (n) = number of opponent's winning lines.

For example, in fig. 4.13 M (n) =6 , O (n) = 5 and hence e(n) = 1.

Now, we will discuss a new type of algorithm, which does not require expansion of the entire space exhaustively. This algorithm is referred to as alpha-beta cutoff algorithm. In this algorithm, two extra ply of movements are considered to select the current move from alternatives. Alpha and beta denote two cutoff levels associated with MAX and MIN nodes. The alpha value of MAX node cannot decrease, whereas the beta value of the MIN nodes cannot increase. But how can we compute the alpha and beta values? They are the backed up values up to the root like MINIMAX. There are a few interesting points that may be explored at this stage. Prior to the process of computing MAX / MIN of the backed up values of the children, the alpha-beta cutoff algorithm estimates e(n) at all fringe nodes n. Now, the values are estimated following the MINIMAX algorithm. Now, to prune the unnecessary paths below a node, check whether

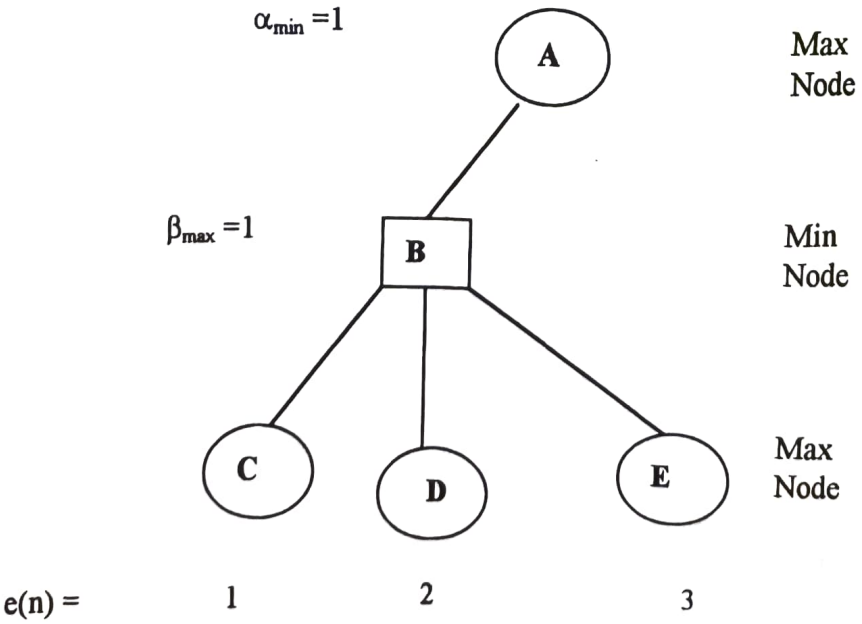
i) the beta value of any MIN node below a MAX node is less than or equal to its alpha value. If yes, prune that path below the MIN node.

ii) the alpha value of any MAX node below a MIN node is greater than or equal to the beta value of the MIN node. If yes prune the nodes below the MAX node.



(a) A state M (n) =6 O (n) =5

Fig. 4.13: Evaluation of e (n) at a particular state.



Pass - I

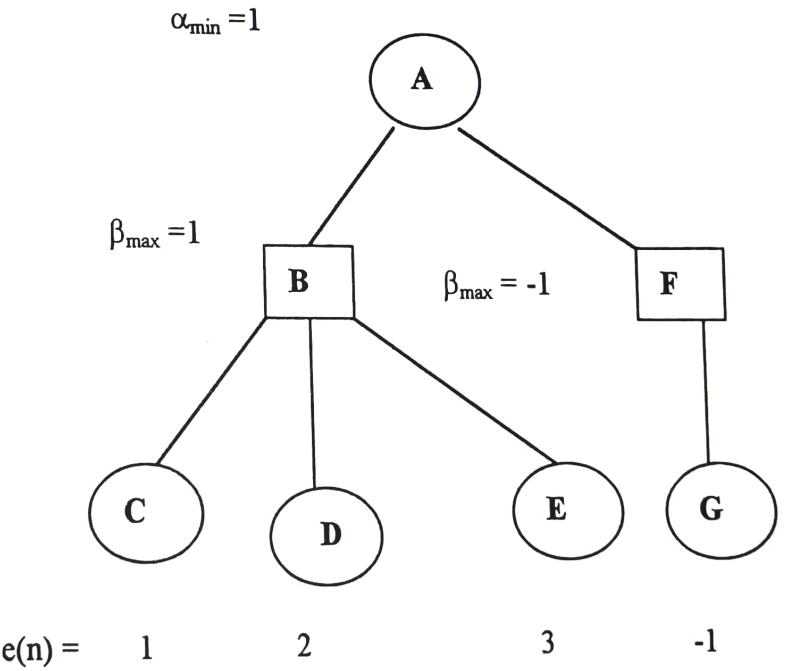
Fig.4.14: Two-ply move of the MAXIMIZER with computed e(n) at the fringe nodes: C, D, E; backed up values at node B and A and setting of α_{max} and β_{min} values at nodes A and B, respectively.

Based on the above discussion, we now present the main steps in the α-β search algorithm.

i) Create a new node, if it is the beginning move, else expand the existing tree by depth first manner. To make a decision about the selection of a move at depth d, the tree should be expanded at least up to a depth (d + 2).

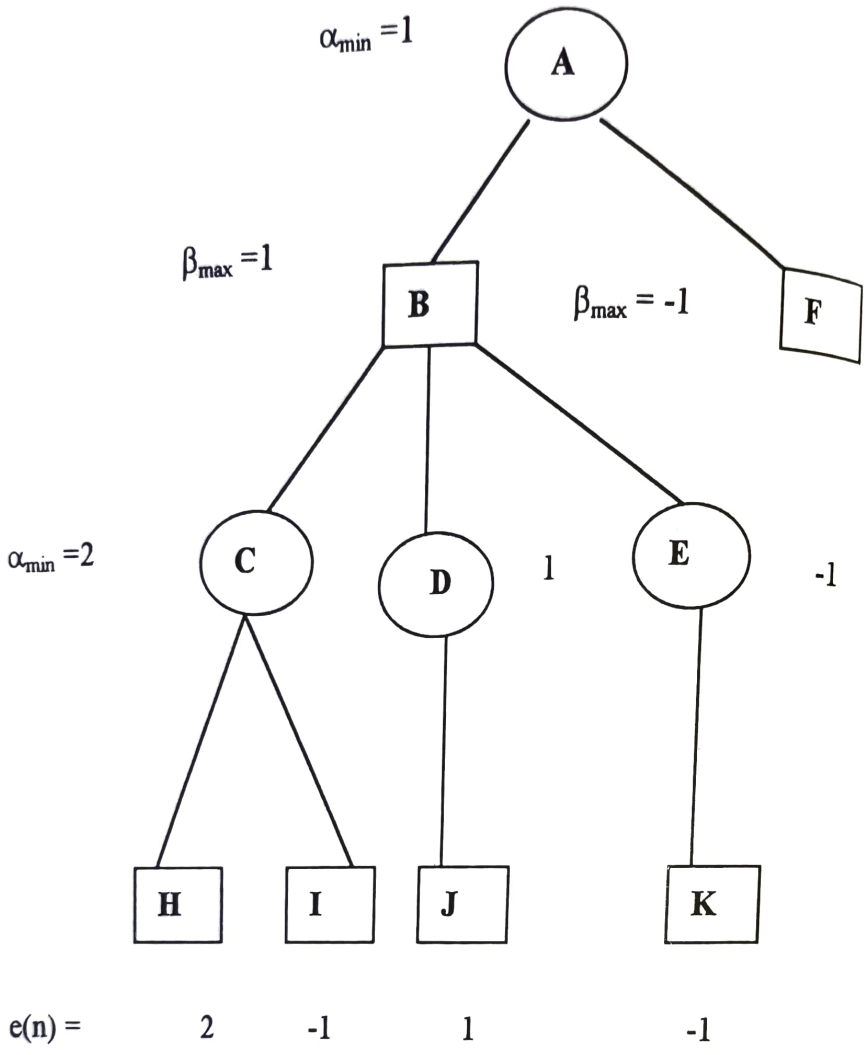
ii) Compute e(n) for all leave (fringe) nodes n in the tree.

iii) Compute α_{min} (for max nodes) and β'_{max} values (for min nodes) at the ancestors of the fringe nodes by the following guidelines. Estimate the minimum of the values (e or α) possessed by the children of a MINIMIZER node N and assign it its β'_{max} value. Similarly, estimate the maximum of the values (e or β) possessed by the children of a MAXIMIZER node N and assign it its α'_{min} value.



Pass - II

Fig. 4.15: A thinks of the alternative move F, and also mentally generates the next ply move G; e(G) = -1; so β_{max} at F is –1. Now, β_{max} at F is less than α_{min} of A. Thus there is no need to search below F. G may be pruned from the search space.



Pass – III

Fig. 4.16: The node G has been pruned; The nodes C, D and E have been expanded;The e(n) is estimated at n = H, I, J and K and the α_{min} values are evaluated at nodes C, D and E. Since the α_{min} value of C is greater than the β_{max} value of B and α_{min} value of D = β_{max} value of B, there is no need to search below nodes C and D.

iv) If the MAXIMIZER nodes already possess α_{min} values, then their current α_{min} value = Max (α_{min} value, α'_{min}); on the other hand, if the MINIMIZER nodes already possess β_{max} values, then their current β_{max} value = Min (β_{max} value, β'_{max}).

v) If the estimated β_{max} value of a MINIMIZER node N is less than or equal to the α_{min} value of its parent MAXIMIZER node N' then there is no need to search below the node MINIMIZER node N. Similarly, if the α_{min} value of a MAXIMIZER node N is more than the β_{max} value of its parent node N' then there is no need to search below node N.

The above steps are continued until the game is over. If we call these five steps together a pass, then the first three passes are shown in fig. 4.14-4.16. The interested reader may on his own work out the tic-tac-toe game using the definition of e (n) and the last 5 steps. We could not present it here because of the page size of this book, which cannot accommodate large trees.

4.5 Conclusions

We presented a large number of search algorithms in this chapter. We started with the breadth first and the depth first search algorithms and analysed their complexities. It is clear from the analysis that the breadth first search is not appropriate for large state space as the memory requirement is excessively high. The depth first search algorithm works well for most of the typical AI problems. Sometimes, we may not be interested to explore below a given depth. The iterative deepening search is useful under this context. Recently, the iterative deepening algorithm has been formulated in A* fashion and is called the IDA* algorithm. The IDA* algorithm has a great future, as it has been seriously studied by many researchers for realization on parallel architecture [4]. We shall take up these issues in chapter 22.

Among the heuristic search algorithms presented in the chapter, the most popular are A* and AO* algorithm. A* is used on a OR graph, while AO* is employed in an AND-OR graph. The A* is applied in problems to find the goal and its optimal path from the starting state in a state space, while the AO* determines the optimal paths to realize the goal. Heuristic search, since its inception, has remained an interesting toolbox for the researchers working in AI. Recently, Sarkar et al. [8] extended the A* algorithm for machine learning by strengthening the heuristic information at the nodes of the state-space. We, however, do not have much scope to discuss their work here.

Besides the search algorithms presented in the chapter, there exist a few more problem solving techniques we introduced in chapter 1. These are constraint satisfaction techniques, means and ends analysis and problem

reductions. Constraint satisfaction techniques, being an emerging research area, will be presented in detail in chapter 19. Means and ends analysis and problem reduction techniques, on the other hand, are available in most text books [7] and we omit these for lack of space.