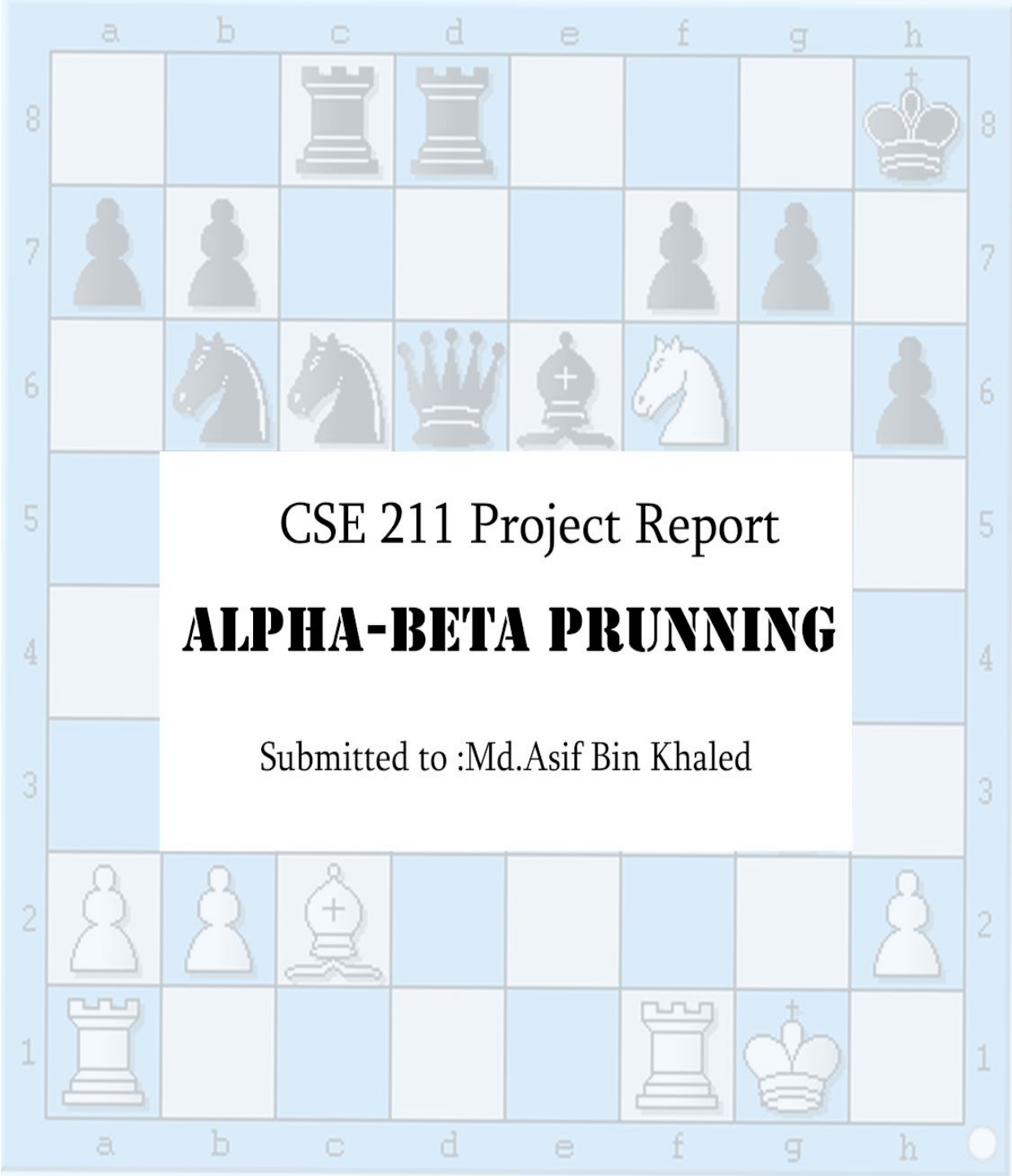
Group 21: PRUNE AND SEARCH PARADIGM

Section : 03



CSE 211 Project Report

ALPHA-BETA PRUNNING

Submitted to :Md.Asif Bin Khaled

Date : 10/01/2021

Submitted By : Nabila Islam_1721662
Ishtiaq Ahmed _1720943
Sabrina Masum Meem _1720694

Introduction of the Paradigm

Prune- and- search paradigm was invented by a mathematician and computer scientist , Nimrod Megiddo, in 1983.
This paradigm is used for solving various optimization problems.
The basic feature of prune- and- search is to reduce the search space by pruning a fraction of the input elements and recurse on the remaining valid input elements.

In this paradigm an optimal value is found by iteratively dividing the search space into two parts:
Part A : which contains the optimal value and is recursively searched
Part B : which does not contain the optimal value later pruned away (eliminated)

Pruning is a technique where without checking each node of the tree we can compute the correct minimax decision.

Prune and search is a form of decrease and conquer algorithm, At each iteration it prunes away a fraction of the input data and then applies the same algorithm recursively to solve the problem for the remaining data.

Notable algorithms in Prune and Search Paradigm includes :
1.Negamax
2.Expectiminimax
3.AlphaBeta Pruning
4.Minimax algorithm
5.Geometric Algorithm
6.Selection Problem
7.Fixed Dimensional Linear Problems

Our selected algorithm is **AlphaBeta Pruning.**

The name Alpha Beta comes from the two parameters in the algorithm.
Alpha : Is the maximum value that can be obtained at the current level or above.
Beta: Is the minimum value that can be obtained at the current level or above.
AlphaBeta Pruning is an adversarial search algorithm.

Adversarial search is where we examine the problem, which arises when we try to plan ahead of the world and the opposition party is planning against it.

This algorithm uses tree pruning to improve the minimax search of data tree structures. AlphaBeta allows the program to search thoroughly all potential options for moves and choose the best one.
Each node keeps its Alpha (Maximum) and Beta (Minimum)value that has been found along the branch . When it reaches a branch that guarantees it can't beat it with other values, prunes (eliminates)the remaining nodes , saving calculation time.

AlphaBeta Pruning is an optimization technique for MiniMax Algorithm.

MiniMax is a recursive algorithm: It performs a depth-first search algorithm for the iteration of the complete tree and then proceeds all the way down to the leaf node of the tree then backtracks the tree with recursion.

Minimax algorithm works with two decision rules to minimize the loss of the worst possible case. The decision alternates at every current level for a maximum player and minimum player. Each player will choose the best output for themselves.
Nodes are each given a value for their position with higher being better for the current player. The algorithm searches through the entire tree . Node values are taken within the depth of the tree to evaluate which path to take returning the optimal branch with highest guaranteed value, without prior knowledge of the opponents move.

Pseudocode :

```
Function AlphaBeta (node, depth, alpha, beta,MaxPlayer )
If depth is null or is a Leaf node
Then
Return optimal value of node
If MaxPlayer then
Value = -infinity
For each child of node
Execute
AlphaBeta =( child, depth - 1, Alpha , Beta)
Value = max (value,AlphaBeta , False)
Alpha= max (Alpha, Value)
If
Alpha>= Beta
Then
Break
Return Value
Else
Value = +infinity
For each child of node
```

```
Do
AlphaBeta =(child , depth-1, Alpha, Beta)
Value = min ( value, AlphaBeta, True)
Beta=min (Beta, Value)
If
Beta <= Alpha
Then
Break
Return Value.
```

Implementation :

```
// C++ program to demonstrate
// working of Alpha-Beta Pruning

#include<bits/stdc++.h>
using namespace std;
const int MAX_NODE = 9999;
const int MIN_NODE = -9999;

 int MinMax(int depth, int nodeIndex,bool maximizingPlayer,int values[], int alpha,int beta){
          if (depth == 3)
          return values[nodeIndex];
          if (maximizingPlayer)
          {
          int BEST_CASE_NODE = MIN_NODE;
         for (int i = 0; i < 2; i++)
          {
         int VALUES = MinMax(depth + 1, nodeIndex * 2 + i,false, values, alpha, beta);
         BEST_CASE_NODE = max(BEST_CASE_NODE, VALUES);
         alpha = max(alpha, BEST_CASE_NODE);
         if (beta <= alpha)
                   break;
          }
          return BEST_CASE_NODE;
          }
          else
          {
          int BEST_CASE_NODE = MAX_NODE;
```

```
        for (int i = 0; i < 2; i++)
        {
        int val = MinMax(depth + 1, nodeIndex * 2 + i,
                        true, values, alpha, beta);
        BEST_CASE_NODE = min(BEST_CASE_NODE, val);
        beta = min(beta, BEST_CASE_NODE);
        if (beta <= alpha)
                break;
        }
        return BEST_CASE_NODE;
        }
}
int main()
{
        int elements[8] = { 2, 10, 7, 9, 3, 6, 4, -6 };
        cout <<"OUTPUT: "<< MinMax(0, 0, true, elements, MIN_NODE, MAX_NODE);
        return 0;
}
```

Pruning Condition:
Alpha =< N =< Beta

The algorithm works as follows :

1. Depth First traversal from root to end of the tree.
2. Declaring parameters:
   >setting Alpha as INT _MIN (- infinity)
   >setting Beta as INT_MAX (+infinity)
3. Traversing the tree in DFS order to get the highest value possible for maximizer player and lowest value possible for minimizer player.
4. Values are for Alpha and Beta are updated accordingly and passed to the child notes.
5. Backtracking the tree Node values will be updated. Values will be passed to upper nodes .

Complexity Analysis :

Minimax has a search time of O(bm)
It is the sum of all levels of the tree: b0+b1_b2+....bm, b =maximum branching factor o and m is the maximum depth of the tree

Best Case:	Best case can be described as while expanding a node , the algorithm will choose

	the best order . Let us take a scenario where the greatest to least is MAX_NODE and the least to greatest is the MIN_NODE . Let, b=branching factor d=depth v=optimal score Here the condition for pruning is : α□β Procedure: For the current player , the algorithm expands with “v” . Let current player be MAX_PLAYER · At first the MAX_NODE score will be less than “v”. · At first the MIN_NODE score will be less than “v”(Here all MIN_NODE meets the condition above) So now : FOR MAX_NODE : · All the child has to be searched · The condition is not met since no score is greater than v FOR MIN_NODE : · The condition is met · Pruned again Now after every two levels the node “b” is expanded and a sqrt b branching factor is met.
Average Case :	Here, Branching factor = b Depth = d Now, MAX_LEAF_NODE position = O (b*b..*b) =O(b^b) FOR ODD DEPTH: LEAF_NODE = O(b*1*b*1*...*1) FOR EVEN DEPTH:LEAF_NODE = O(b^(d/2))

	Here b*1*b*1* describes that the moves by the first player determines the best move but the best move by the second player can refute all the moves Average case implies that Alpha-beta pruning is the best algorithm for two player based games which involves score keeping
Worst Case:	The whole tree is examined since no nodes can be pruned since all the branches are ordered Here, depth = d The grand children is found to be b^2 by examining the node Now we find the algorithm to be O(b^(d/2)) nodes i.e the worst case WORST CASE COMPLEXITY: O(|V|) = O(b * d)

Optimization :

The efficiency of alpha beta pruning is dependent on the ordering in which the nodes are traversed. Move order plays an important role in alpha beta pruning, it works in two ways:

Worst order : Cases where the algorithm does not prune any leaves of the tree. It consumes more time because of alpha and beta factors. In this scenario the best choice occurs on the right branch of the tree , Time complexity : O(b^m)

Idea order : Cases where lots of pruning occurs in the tree. DFS is applied to traverse the whole tree which works more efficiently than a minimax algorithm in the same amount of time. Best move is found at the left side of the tree, Time complexity : O(b^ m/2)

Alpha Beta pruning is commonly implemented for game playing in AI and for a fast searching approach.
It has been used in games like Chess, Checkers, tic-tac-toe, Go and other two player games. This algorithm can also be implemented to build games such as : COUP , Rummikub , Scopa, GoFish etc.

The Expectiminimax algorithm is similar to Alpha-Beta Pruning algorithm. It is a variation of minimax algorithms used in systems of artificial intelligence. A chance node reveals the probabilities of certain moves. In expectiminimax nodes are alternated with max and min nodes and takes a weighted average for which the outcome depends on the users' skills and chances unlike AlphaBeta pruning which chooses from combinations of several alternatives.

Pruning does not impact the final outcome. Assuming the lower bound on each value of the function and upper bound on the sum of all values, shallow alpha beta pruning is valid, saving computational time. But the effectiveness of this algorithm is limited to the optimal directional approach for two player games only.
