“A Review Of Game-Tree Pruning” T.A. Marsland (1986)

AI for Games

Jessica Löhr
Matrikelnummer: 3467703
jessica-loehr@web.de

May 2, 2019

Contents

1 Introduction

Back in the 1950s when alpha-beta pruning was invented it was a huge breakthrough for machine playing. Since it greatly decreases the number of positions in a game tree that needed to be evaluated in order to find the best possible move, engines became way faster. Especially in games where the players have just limited time to make a move, alpha-beta pruning greatly increased the performance of machine play. Even today, 50 years later, engines like the currently best chess engine Stockfish [3] use variations of alpha-beta-pruning or methods that are based on its ideas.
In 1986, computer scientist Thomas Anthony Marsland released an article "A Review of Game-Tree Pruning" [4] where he covers and evaluates different enhancements to standard alpha-beta pruning. One of the main methods is the so called "Principal Variation Search", short PVS, for what he was a co-creator. PVS, along with some other minor enhancements, belong to the methods that contribute to the strength of todays chess engines. In this report i will explain these methods in more detail as well as the problems that they still face. Before that i will give a short explanation of the basics that are of relevance for the understanding of this report.
Since chess is a large area for these algorithms and the paper refers to chess as an example, i will also cover chess specific examples.
Most of the pseudocodes are taken from the paper [4] and rewritten for todays standards. Only quiescence search was taken from the page "https://www.chessprogramming.org/" [2]

2 Prior Knowledge

In this chapter i will shortly cover some background that is needed to understand the enhancements. I assume that the reader knows about the minimax approach since there already were several talks about it. If you need more detailed information about it, i advise you to go read the reports of the corresponding talks or read the paper "Artificial intelligence: a modern approach" [6] .

2.1 Negamax

The pseudo code of all algorithms that are covered in the original paper and this report follow the so calles "Negamax" approach. It is a way to model the minimax search without having to program the same code two times, one for the minimizing, one for the maximizing player [1]. Instead it uses the mathematical condition, that maximizing a function is equal to minimizing the negative function:

$$max(a, b) = -min(-a, -b)$$

All of the following code snippets use the negamax approach.

2.2 Alpha-Beta Pruning

Alpha-beta pruning is a simple recursive function to decrease the size of game trees. It was invented in the 1950s and variations of it are still used even at strong chess engines today.

2.2.1 Idea

The idea behind the algorithm is quite close to how humans actually think. The two parameters α and β represent the values of the moves that are currently, for α, the worst or, for β, the best the algorithm found. Humans would understand α as a lower bound, what means that if he calculates a move that leads to a line with a result worse than α, he does not need to examine that line any further. Otherwise, assuming perfect play of the opponent, the game tree would result in a score under α what is obviously not the best result the player could reach. It behaves similar with β. β can be understood as the lower bound for the opponent, what means that moves above β lead to results that are too good so that the opponent will not choose that line. Hence the player would not need to examine that line. Of course this again assumes perfect play of the opponent.
So to keep it short, the alpha-beta algorithm looks for the best possible α without paying attention to sub-trees that lead to a score that exceeds β.

2.2.2 Algorithm

The algorithm takes the start position, α and β bounds and the current depth as input and returns the score for the input position.

Algorithm 1: AlphaBeta

```
   Input: position, α, β, depth
   Output: score
 1 int score, j, value;
 2 postion[] posn;
 3 if depth == 0 then
 4     return evaluate(p) ;
 5 posn = generate(p);
 6 if posn.isempty() then
 7     return evaluate(p);
 8 score = −∞ ;
 9 for j = 0 to posn.size()-1 do
10     Make(posn[j]);
11     value = -AlphaBeta(posn[j], -β, -max(α, score), depth-1);
12     if value > score then
13         score = value;
14     Undo(posn[j]) ;
15     if score ≥ β then
16         break ;
17 return score ;
```

Before the actual alpha-beta part starts, it is checked weather the current node is a leaf node, since the result of evaluation function of that node needs to be returned then. Also those positions are generated that result from all legal moves. If there are none, the game is finished and also the result of the evaluation function returned. Then the actual alpha-beta algorithm starts. The algorithm iterates over all possible following positions and stores the value of the best continuation in a variable *score*. In line eleven the negamax approach is used to start the alpha-beta algorithm for the child node in order to get the score of it. Line twelve checks if ithat value exceeds α, since we want to keep that value then. Line fifteen checks for a β cutoff. The functions *Make()* and *Undo()* are self-explanatory and refer to ḿaking a move and taking it back.

2.3 Horizon Effect

The horizon effect is a phenomenon that occurs due to limited search depth. It appears if the line that is calculated as the best line will be refuted some moves later. The algorithm could not detect the refutation since the refutation lies beyond the chosen search depth. On the other hand the search depth can not just be set to the total

number of moves in a game because the algorithm would then take too much time to determine. Even if that calculation power was available, in some cases it would still not determine, for example when there is a "Dauerschach" in chess. So the horizon effect can not be avoided when using standard alpha-beta pruning or any algorithm that uses a fixed search depth in general. Methods that avoid the appearance of the horizon effect, such as quiescence search, will be covered in chapter three.

3 Enhancements

In this chapter, the enhancements of alpha-beta pruning will be explained. The several sections for the algorithms are split into subsections, where i will explain the main idea first, followed by a pseudocode with explanation and a short conclusion at the end.

3.1 Quiescence Search

Quiescence Search is an approach that avoids the horizon effect by omitting a parameter for the search depth.

3.1.1 Idea

Core idea is that instead of using a fixed search depth, those moves will be examined that lead to a heavy change of the value of the sub-tree. The algorithm in general terminates when positions are reached where the values will not show immense changes when traversing deeper through the tree, in other words when the position becomes "quiet". For chess this means that all capture moves will be examined. Since the opening of a game might only consist of quiet positions, pure quiescence search is of course not suitable for being used alone and should thus only be used as an enhancement for other search algorithms. It is usually called instead of the evaluation function at the end of another algorithm after it performed the main search to avoid the horizon effect [5].

3.1.2 Algorithm

As mentioned before, quiescence search lacks the depth parameter. Instead it traverses the tree deeper until the position becomes "quiet". In this case for chess, capture moves are most likely to refute a line and thus examined in this algorithm. The algorithm terminates when there are no capture moves in the position available.

Algorithm 2: Quiescence

```
1  [2] Input: position, α, β
   Output: score
2  value = evaluate(p) ;
3  if value ≥ β then
4      return β ;
5  if α <value then
6      α = value ;
7  capt = generateCaptures(p);
8  for i = 0 to capt.size() do
9      Make(capt[i]) ;
10     score = -Quiescence(−β, −α);
11     Undo();
12     if score ≥ β then
13         return β ;
14     if score > α then
15         α = score ;
16     return α ;
```

3.2 Aspiration Window

3.2.1 Idea

In alpha-beta search, ∞ and $-\infty$ are used as initial bounds. Aspiration search tries to estimate those bounds by trying to estimate the value of the position that needs to be evaluated. In chess this could be done by using a window around the material balance, which is calculated by subtracting the sum of the values of the material of player A from those of player B. The table on the right shows the value of pieces in chess. Of course this just a rough estimation of the position and does not a cheap replacement of the evaluation function, since the value of a position is way more complex than just the material balance. That is why an error rate **e** has to be estimated.

Piece	Value of piece in pawns
Pawn	1
Knight	3
Bishop	3
Rook	5
Queen	9

3.2.2 Algorithm

This function calls the alpha-beta function with optimized bounds:

Algorithm 3: AspirationSearch

```
1 α = V - e ;
2 β = V + e ;
3 V = AlphaBeta(p, V, +∞, depth);
4 if V ≥ β then
5 |   V = AlphaBeta(p, V, +∞, depth);
6 else
7 |   if V ≤ α then
8 |   |   V = AlphaBeta(p, −∞, V, depth);
```

3.3 Principal Variation Search

Its increase in performance comes from the assumption that the best move is the first to be evaluated. That also means that PVS is able to prune the most amount of branches if the moves are ordered.

3.3.1 Idea

Core idea behind PVS is that is it easier to determine if a subtree will lead to a cutoff instead of calculating its exact value and the assumption that the first examined move is also the best. Thus, before traversing through the tree further, a null-window search of the correspoding sub-tree is performed first. Null-window search means that the α and β bounds are set to $\alpha = \alpha$ and $\beta = \alpha + 1$. Even if that means that the search will always fail, we will be able to determine if the search will cause a β cutoff or lies under α instead. If it fails high, the sub tree does not need to be searches any further, otherwise if the result is also about α of the node that started the null-window search, the sub-tree will be searched again with the result of the null-window search as α and β as β bound.

3.3.2 Algorithm

Algorithm 4: PVS

```
   Input: position, α, β, depth
   Output: score
 1 int score, j, value;
 2 postion[] posn;
 3 if depth == 0 then
 4 |  return evaluate(p) ;
 5 posn = generate(p);
 6 if posn.isempty() then
 7 |  return evaluate(p);
 8 score = -PVS(posn[0], −β, −α, depth-1);
 9 for j = 1 to posn.size()-1 do
10 |  if score ≥ β then
11 |  |  break ;
12 |  α = max(score, α) ;
13 |  value = -PVS(posn[j], −α-1, −α, depth-1);
14 |  if value > score then
15 |  |  if (α < value) & (value < β) & (depth > 2) then
16 |  |  |  score = -PVS(posn[j], −β, -value, depth-1) ;
17 |  |  else
18 |  |  |  score = value ;
19 return score ;
```

Main difference to alpha-beta pruning lies in line eight and thirteen. In line eight you can see that the first move is considered as the best and all following values will be compared to this score, unlike in the alpha-beta algorithm where *score* is assigned to $-\infty$. In line thirteen the null-window search is performed. Only if the result of this search is within the bounds of the parent node that started the null-window search and still better than *score*, the branch is worth visiting again. This re-search is then done with bounds that allow a successful search.

Remember that even though the code is more complex and branches might be searched twice, the duration for the evaluation of the whole tree will greatly decrease since less evaluation functions of the leaf nodes need to be called and that those functions are the most expensive part of the evaluation.

3.3.3 Move Ordering

As already mentioned, PVS works best if moves are ordered and the first searched move is also the best one. Move ordering in chess makes use of the fact, that good moves are those that refute a line, or cause a cutoff in other words. Technically, often different

types of tables are used, that somehow store how likely a move is to refute a line. As already mentioned for quiescence search, capture moves are most likely to cause such a refutation. The first implementation for this is the so called **killer heuristic** that stores the two moves that are most likely to cause a refutation for every depth of the tree. A method that is more common today is the so called **history heuristic** that keeps track not only of the two moves that are most likely to refute a line, but of the frequency for all legal moves to refute a line. This is implemented by a 64 x 64 table, where the first entry stands for the source and the second one for the target position.

4 Conclusion

The paper introduces several methods to enhance alpha-beta pruning. The algorithm can be enhanced in two ways:

1. improvement so that more branches can be pruned and a saving of time is achieved
2. improvement so that the search depth is higher and possible refutations can be uncovered timely

For the first improvement, several algorithms were introduced that changed the initial bounds of alpha-beta pruning. While alpha-beta pruning starts with infinite bounds, other algorithms such as PVS introduce null windows and the assumption that the first move is the best, to achieve more β-cutoffs. For that, also different move ordering mechanisms were introduced. Others like aspiration search try to estimate the value of the current position by using different heuristics that the game board offers, such as material balance, to select suiting bounds.
To avoid playing a move, whose refutation lies beyond the search depth, a method of iterative deepening was introduced: quiescence search. Since it is commonly used at the leaf nodes of the line we want to play, the horizon effect that was a big deal in alpha-beta pruning can easily be avoided.
To obatin the best result, engines use a mixture of all those methods.

All the methods that were discussed here have a big influence on game engines today. Stockfish, the currently strongest chess engine, for example still uses a mix all methods that were listed here [3]. Of course it also uses several more advanced methods, but the fact that methods that were known about 40 years ago still appear in todays engine is quite impressive.

Bibliography

[1] Hamed Ahmadi. Negamax. http://www.hamedahmadi.com/gametree/.

[2] Chessprogramming.org. Quiescence search. https://www.chessprogramming.org/Quiescence_Search".

[3] Chessprogramming.org. Stockfish. https://www.chessprogramming.org/Stockfish.

[4] T. A. Marsland. A review of game tree pruning, 1986.

[5] Bruce Moreland. Quiescence search. http://web.archive.org/web/20070813042640/www.seanet.com/~brucemo/topics/quiescent.htm.

[6] Stuart J Russell and Peter Norvig. Artificial intelligence: a modern approach, 2016.