Modern developments of Shannon's Chess

Dmitry Batenkov

March 1, 2006

Abstract

We investigate some improvements and recent developments in the game-tree search techniques, which have influenced development of modern computer chess programs.

We shall present an overview of various search heuristics to the basic Alpha-Beta (AB) pruning algorithm in order to achieve good move ordering (Iterative Deepening, Killer Heuristic, History Heuristic, Transposition Tables and Aspiration Windows), as well the improvements to the search algorithm itself (Scout, NegaScout, SSS*, MTD(f)).

Finally, we take a brief glimpse into other directions of research (Conspiracy Numbers, Minmax approximations).

1 Introduction

1.1 Computer Chess Brief History

The interest in devising a computer program that would play chess well against human opponents originated almost simultaneously with the invent of digital computers. In his paper ([9]) Shannon suggested the game-theoretical approach to programming a computer to play chess: the program would traverse the minmax tree up to some depth (which is chosen to be computationally feasible) and choose the best move. The leafs of the tree are evaluated according to some static evaluation function which holds the "knowledge" of the position. This approach can be called "brute-force".

Even Shannon realized that the brute-force approach ("type A strategy") would not be enough for a real-world play. Thus he suggested that a better program would

1. Examine forceful variations beyond the search depth
2. Choose "intelligently" which moves to investigate. In effect, only an approximate result is obtained, contrasted with the exact minmax value of "type A strategy".

The most immediate enhancement to minmax search that was quickly discovered (in 1956) is the Alpha-Beta (AB) pruning, which reduces unnecessary evaluation of tree branches.

Up to the middle 70's, various programs have been built that incorporated the AB with various *ad-hoc* methods including knowledge-based (see on this later), as suggested by the point 2 above.

The 1975-1985 period, called by Simon and Shaeffer the *technology era* in [10], was characterized by research concentrating various improvements and enhancements to AB in order to achieve the minimal tree (see more on this in Section 2).

Since 1985, it could be argued that the AB area was exhausted, as experiments suggested that the AB with various improvements was approaching its optimal limit. So new directions of research emerged (see Section 3).

1.2 Search vs. Knowledge

Two major factors are supposed to affect performance of a chess program:

- the quality of *knowledge* it posesses regarding a given position or the feasibility of a move. This means that a program will choose which move to make on the basis of some domain-specific information and not just the node's minmax value. This approach resembles the way humans play chess, applying extensive pattern recognition and learning abilities.
- its *search* capability - the number of positions (variations) it can consider or the maximal depth of the game-tree the program traverses.

In this article we concentrate on the "search" and not the "knowledge". It is a fact that most of the chess playing programs are concerned with "search" and have very little or no "knowledge" (besides the evaluation function at the leaf nodes, of course).

While seemingly at the opposite sides of the spectrum, the exact relationship between "search" and "knowledge" is a complex one. Deeper search may make a chess program look more "knowledgeable" than it really is.

1.3 Search and performance

It was found by experiments that there exists a strong correlation between the search capability and performance.

"Performance" in this context means "self-play". For example, given a program which searches the tree up to depth d, it is calculated how often it outperforms (i.e. makes a better move) itself when searching the tree just to depth $d-1$. *Diminishing returns* occur whenever the ratio $\frac{\text{Performance}(d)}{\text{Performance(d-1)}}$ decreases as d increases. It has not been conclusively demonstrated that the game of chess exhibits diminishing returns.[1]

Entirely different picture, however, emerges when "performance" is considered as the strength against human opponents. It was erroneously suggested that the program's rating increase is linearly dependent on the amount of search.

[1]It should be noted that for games other than chess (checkers, Othello), it has been shown that there exist diminishing returns.

In the middle 70's, when the best programs were approximately at the 1800 level, every additional ply was estimated to be worth as much as 250 points. However, as the programs began to appoach Master (2200) level, each ply is only worth 100 points and the rate is expected to decrease even further as the programs approach and surpass the Grandmaster (2500) level. [2] (See [10, 1] for further details.)

1.4 Theory and practice - experiments

In the end, effectiveness of one or another game playing algorithm is assesed by performing experiments with either simulated data or under tournament conditions, where severe real-time performance constraints are imposed. While this might be considered as having nothing to do with "Computer Science" per se, Shaeffer argues [6] that this might be the only viable option:

> In theory, given sufficient resources, all minimax search algorithms (such as alpha-beta, SSS*, conspiracy numbers, etc.) are equivalent. These search algorithms have different costs and search strategies which means that they can produce different answers given the same real-time constraints. There isn't a usable theory of optimization that allows one to scientifically analyze the myriad of possibilities and select the "best" one. Instead, extensive experimentation is the only solution.

2 Alpha-Beta dominance

The most basic enhancement of a minimax depth-first search algorithm is the Alpha-Beta pruning, first discovered in 1956. Since then, it has remained the main technique to traverse minimax game-trees. Since it is very easy to understand and implement, it has been incorporated into all major chess programs.

2.1 Optimal Minimax Tree

In 1975 a paper describing the performance of optimal Alpha-Beta is published [2]. It is shown that given constant branching factor w and a depth d, any algorithm that finds the minimax value of the tree must traverse at least $N_{\min}$ leaf nodes, with

$$N_{\min} \stackrel{\text{def}}{=} w^{\lfloor \frac{d}{2} \rfloor} + w^{\lceil \frac{d}{2} \rceil} - 1 \tag{1}$$

However, in order to achieve this theoretical minimum, it is imperative that the nodes be ordered from best to worst, i.e. the algorithm should evaluate the best move first.

[2] The ratings are usually measured in ELO scale. Best human players have ratings near 2700 points.

This result has been used extensively by various authors in order to compare the performance of their algorithms to the "optimal" case, calculating the number of nodes their algorithm expanded relative to $N_{\min}$.

2.2 Alpha-Beta Enhancements

During the years, various enhancements to the basic AB algorithm have been proposed. Following is a brief overview of the major and mostly used ones. For futher information, see [11, 7].

2.2.1 Iterative deepening

This is one of the most important heuristics. Basically, it says the following: in order to perform depth-first d-ply search, first do a $d-1$-ply search and then use its results to obtain a good move ordering for the additional ply. Although it may seem like a wasted effort (why evaluate the intermediate nodes of ply $d-1$ if we are not going to use them in the final minimax estimation?), a careful analysis of (1) shows that the time spent expanding levels $1, 2, \ldots, d-1$ is insignificant compared to the time spent expanding level d. Most important advantages:

- When starting to expand nodes at ply d, there is a good estimate of which move to begin with - just take the best move of ply $d-1$. As stated earlier, good move ordering is essential for the performance of AB.
- At any point there is a reasonable estimate for the best move. This is important for example under tournament conditions, when there is a time constraint on each move.

Iterative deepening has been widely recognized as an essential component of any competative chess program.

2.2.2 Killer heuristic

The idea here is to remember the "killer moves", or the ones which produced a cutoff, and try these moves as potential candidates for cutoff in other positions - at the same level, or even at an earlier level (which in fact causes a *dynamic move reordering*). The rationale behind this heuristic is the assumption that if a move is a good one in some position, it will be as good in a slightly different position.

The usefulness of the Killer Heuristic is not undisputed (as, for example, is the Iterative Deepening), therefore actual implementations of chess programs may or may not use it.

2.2.3 History heuristic

A generalization of the Killer Heuristic is the so-called "History Heuristic", which employs more elaborate scheme for "remembering good moves". While

the Killer Heuristic remebers just several best moves, the History Heuristic keeps track of the successful moves on all depths (a successful move is either the one causing the cutoff or the one leading to the current minimax value of the tree). Various scores can be assigned to moves according to various factors, for example its depth (see [7] for details). It is reported that the heuristic has proven useful for parallel implementations of AB.

2.2.4 Transposition tables

This is the most immediate improvement that always comes to mind - store the results of previous searches and reuse them if necessary. The actual game-tree is really a game-graph, with the possibility that a position can be reached via different paths (or, in other words, a node may have more than one parent). So it makes sense to store previously calculated subtree values and use them whenever the same position is reached afterwards. For every such position we will store it in a table, including the value of the node, the move that leads to that value and the depth to which the node was searched. On a subsequent search, if the same position (node) is encountered, either of the two occurs:

1. The position has been previously searched to the desired depth. In this case, just retrieve the value from the table and return.

2. The position has been previously searched to a smaller depth than desired. In this case, we must expand the node to the desired depth. However, we may use the best move of the shallow search as a guess for our best move.

The transposition tables tend to be large, since we want to reuse as many previous positions as possible.

2.2.5 Enhanced Transposition Cutoffs (ETC)

As an improvement to Transposition Tables the following enhancement have been suggested in [8]: consider some interior node N (as in Figure 1) with children B and C. Suppose that the transposition table suggests that B should be expanded. If B produces (at some later stage) a cutoff for N, then the node C would never be searched. However, C may transpose to some previously encountered branch A, causing an immediate cutoff at N. So the enhancement would be as follows: before searching deeper at a node N, first perform a lookup in the transposition table for all children of N - maybe one of them will cause an immediate cutoff.

2.2.6 Aspiration windows

Recall that in every stage of the basic AB, a *search window* is maintained, which is the interval (alpha,beta). The search begins with the window of $(-\infty, +\infty)$ and is narrowed to the final minimax value. However, if we estimate the value of the root to be within some interval *before the search*, we can run the AB with our estimate of the initial window, thus causing more cutoffs along the

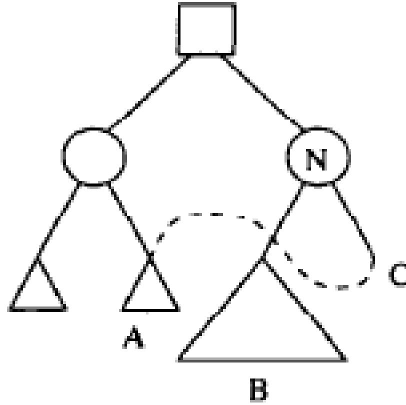

Figure 1: Enhanced Transposition Cutoff example

way. If the final root minimax value falls within our guessed interval, we're done. Otherwise we have either a *fail high* or a *fail low*, in which case our initial estimate was incorrect and we have to restart the search with an updated initial window.

2.3 Modifications to Alpha-Beta - minimal windows

The last enhancement in the previous section used a narrower initial search window, which caused more cutoffs. The idea of varying the search window can be taken to its extreme - the minimal window, which will cause most cutoffs. This leads to a whole family of refinements of the basic AB, which we consider next.

2.3.1 Scout, NegaScout (PVS) [5]

The first refinement is the so-called Principal Variation Search (PVS), or NegaScout (which is just Negamax equivalent of Scout). The idea is this: after finding the exact value of the first (left-most) child, we assume that this will eventually be the value of the tree, because the move ordering is considered to be good (i.e. the first move is the best move). In order to test this assumption, we perform the AB search on all the siblings of the first node *with a minimal window*. If our assumption was correct, then a *fail low* will occur (if we are at a maximizing node). If, on the other hand, a *fail high* occured, we will have to restart the search with an updated window, until the true value is found. The penalty of a restart can be minimized if we use some kind of cache of previous results (for example, Transposition Tables). Thus the only nodes that will have to be recomputed are the ones that caused the fail high.

2.3.2 MTD(f) [4]

The (already extreme) idea of a minimal window can also be taken to its extreme, which result in a whole class of algorithms that perform series of AB searches with minimal windows only. As before, the efficiency of these algorithms relies heavily upon the ability to cache previous results. In this case, it can be shown that these algorithms will never expand more nodes than a regular AB. The difference between the algorithms in this family lies with the initial guess. We can either

1. $MTD(+\infty)$ Begin with $+\infty$ and continuously lower the upper bound, until it converges to the true value
2. $MTD(-\infty)$ Similarly, but starting with $-\infty$ and raising the lower bound until convergence
3. $MTD(f)$ Starting with some finite interval and "squeezing" it until convergence

It is interesting to note that the algorithm 1 above has been known in the literature as SSS* since 1979, and only recently it was shown to be equivalent to $MTD(+\infty)$

3 Other directions of research

It appears that the Alpha-Beta family has been thoroughly investigated by the community, and new directions of research begin to emerge. Here we shall only mention some of them briefly.

3.1 Conspiracy numbers [3]

Conspiracy Numbers algorithm is a minimax search procedure that builds game trees to variable depths without application-dependent knowledge. The algorithm gathers information to determine how likely it is that the search of a sub-tree will produce a useful result. "Likeliness" is measured by the conspiracy numbers, the minimum number of leaf nodes that must change their value (by being searched deeper) to cause the minimax value of a sub-tree to change. The search is controlled by the conspiracy threshold (CT), the minimum number of conspiracy numbers beyond which it is considered unlikely that a sub-tree's value can be changed.

3.2 Minmax approximations [10]

Min/max approximation uses mean value computations to replace the standard minimum and maximum operations of AB. The advantage of using mean values is that they have continuous derivatives. For each terminal node, a derivative is computed that measures the sensitivity of the root value to a change in value

of that node. The terminal node that has the most influence on the root is selected for deeper searching. The search terminates when the influence on the root of all terminal nodes falls below a set minimum. Note that when min/max approximation and conspiracy numbers are used, alpha-beta cut-offs are not possible.

4 Future of Computer Chess

The computer chess challenge is considered largely "solved" and it is expected that in a decade, the computer will surpass the human best player in an ordinary game.

Computer hardware improvements (according to Moore's Law) will be sufficient to achieve significant search depths, so conventional Alpha-Beta pruning with enhancements will be enough. The knowledge factor will be largely ignored.

In contrast, games like Go (where the branching factor is much larger than in chess) will not be "solved" by mere increase in speed, and new directions of research, knowledge-based, will have to be developed.

References

[1] Andreas Junghanns, Jonathan Schaeffer, Mark Brockington, Yngvi Bjornsson, and Tony Marsland. Diminishing returns for additional search in chess.

[2] Donald E. Knuth and Ronald W. Moore. An analysis of alpha-beta pruning. *Artificial Intelligence*, 6(4):293–326, 1975.

[3] Lisa Lister and Jonathan Shaeffer. An analysis of the conspiracy numbers algorithm.

[4] Aske Plaat, Jonathan Schaeffer, Wim Pijls, and Arie de Bruin. Best-first fixed-depth game-tree search in practice. In Chris S. Mellish, editor, *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 273–281, San Mateo, August 20–25 1995. Morgan Kaufmann.

[5] Alexander Reinefeld. An improvement of the scout tree search algorithm. *ICCA Journal*, 6(4):4–14, 1983.

[6] Jonathan Schaeffer. Experimental computer science in game playing.

[7] Jonathan Schaeffer. The history heuristic and alpha-beta search enhancements in practice. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-11(11):1203–1212, 1989.

[8] Jonathan Schaeffer and Aske Plaat. New advances in alpha-beta searching. In *ACM Conference on Computer Science*, pages 124–130, 1996.

[9] Claude E. Shannon. Programming a computer for playing chess. *Philosophical Magazine*, 41:256–275, 1950.

[10] H. Simon and J. Schaeffer. The game of chess, 1992.

[11] T.A.Marsland and M.Campbell. A survey of enhancements to the alpha-beta algorithm. *ACM '81*, pages 109–114, November 9–11 1981.