Search Control Methods in Deep Blue

Murray S. Campbell
A. Joseph Hoane, Jr.
Feng-hsiung Hsu

IBM Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598
mcam@us.ibm.com, hoane@us.ibm.com, fhh@us.ibm.com

Abstract

The Deep Blue chess-playing system explores a huge search tree when selecting a move to play, typically examining on the order of 20 to 40 billion positions. Although a relatively complex evaluation function is applied to the leaf positions of this search tree, there is significant possibility of error in the evaluation function values. Incorrect position evaluation can lead to a poor move choice at the root of the search tree. We have developed methods to reduce the susceptibility of Deep Blue to evaluation function errors through the use of selective extensions in alpha-beta search. This paper briefly reviews some of the standard methods employed in game-playing programs to cope with evaluation function errors, and describes some of the techniques used in Deep Blue.

Introduction

The values returned by evaluation functions in game-playing programs are generally uncertain. One can attempt to directly model this uncertainty by representing leaf node evaluations as ranges or distributions and devising methods to estimate and manipulate these quantities (Berliner 1979; Palay 1985; Russell and Wefald 1991; Berliner and McConnell 1995; Baum and Smith 1997). This can lead to principled ways to explore a game tree, both in where to look and how much effort to expend. In spite of the great intuitive appeal of these methods, the overwhelming majority of high-performance game-playing programs, Deep Blue included, use some form of the alpha-beta algorithm (Knuth and Moore 1975). This is in part due to the simplicity, efficiency and empirical effectiveness of alpha-beta. One key feature of alpha-beta is the use of a scalar evaluation function, which allows no easy way to model evaluation uncertainty. Alpha-beta must therefore manage the uncertainty indirectly, through search. The general idea is to focus searching effort on parts of the tree where some sort of "instability" exists, resulting in a non-uniform-depth tree.

This paper will describe some of the mechanisms used in the Deep Blue chess machine to enhance the selectivity of alpha-beta search. Although there are numerous domain-specific ways to detect local instabilities in chess positions, our focus is on ways of measuring the likelihood of an evaluation error propagating to the root of a search tree, thereby affecting the decision at the root. This is directly related to the "forcedness" of the moves leading to a given position.

We begin by reviewing some of the previous work in alpha-beta search that, directly or indirectly, addresses the issue of evaluation function uncertainty. This is followed by an overview of the algorithms used in Deep Blue for this purpose. We conclude by summarizing our work and pointing out some of the unresolved questions.

Previous Work in Alpha-Beta

The most common (and best understood) way to improve an alpha-beta searcher for games like chess is to search as deeply as possible. A long series of experiments by numerous researchers (e.g. (Thompson 1982; Berliner et al. 1990; Mysliwietz 1994)) has clearly established the effectiveness of deeper search. Modern chess programs have adopted a set of enhancements to alpha-beta that produce remarkably deep search depths. Iterative deepening, transposition tables, dynamic move reordering, negascout and recursive null move pruning are just a few of the refinements that are used. Nonetheless, at some point one expects to see diminishing returns for each additional ply of search. There is some debate whether or not this effect has been observed in chess (Junghanns et al. 1997; Hyatt and Newborn 1997; Heinz 1998). Given the exponential work required to achieve ever-greater search depth, it seems clear that the general solution is to use

non-uniform search to attack and solve these problems to whatever degree possible.

The earliest example of a non-uniform search technique is the quiescence search in chess game trees. The need for this was pointed out by in (Shannon 1950) and (Turing 1953) describe a procedure to carry this out. Suppose a position is to be evaluated in which a capture/recapture sequence has been initiated. Evaluating in the middle of the sequence, before the recapture has been played, will yield a grossly inaccurate value. Allowing these sorts of instabilities to resolve by means of a capture search has been common practice in chess programs for decades.

In general, the quiescence search gains its benefit from addressing the arbitrary search depth limitation and making this "stopping point" less arbitrary. Modern chess programs employ much more complex quiescence searches than simple capture searches, and these do a good job of allowing the programs to see beyond their nominal search depth. However, the quiescence search does not address the issue of how likely the resultant value is to propagate up the tree toward the root. This is primarily due to the forcedness of the path to a node, and techniques like check evasion extensions and recapture extensions are chess-specific examples of methods to measure this forcedness.

Deep Blue Search Methods

The Deep Blue search can be logically divided into two distinct parts, which we call the *hardware* search and the *software* search.

The hardware search is implemented on a special-purpose chess accelerator chip (Hsu 1999), and is based on minimal-window alpha-beta. The hardware search does not use a transposition table or iterative deepening, but does include a sophisticated quiescence search, pruning mechanisms and some domain-specific search extensions. Many techniques that might not be cost-effective on a general purpose processor become practical if hardware support is available, and this fact was used extensively on the chip. The hardware search is intended to handle the lower levels of the search tree, and typically will carry out a 4 or 5 ply search at about 2.5 million positions per second. The hardware search will not be discussed in detail in this paper.

The software search is a program that runs on a general-purpose computer. In the case of Deep Blue, the software search runs on an IBM RS/6000 SP. The software search handles the upper levels of the game tree, and calls the hardware search as a subroutine. Many of the standard features of high-performance alpha-beta programs are included in the software search: iterative deepening, transposition tables and move reordering mechanisms. Note that a quiescence search is not necessary, since the hardware search returns a quiesced evaluation.

The software search in Deep Blue is based on a depth-limited variant of alpha-beta. Positions at the depth limit are evaluated by the hardware search. In a vanilla version of depth-limited alpha-beta, each move made increments a depth counter, resulting in a uniform depth search. To add selectivity, the increment of the depth counter is suppressed for certain specified moves. This results in a non-uniform tree, hopefully deeper in the "important" branches and shallower in the less relevant branches. A well-known example of this sort of extension is the check evasion extension, where responses to check do not count toward the depth limit.

In 1991 a new search program was developed with the goal of making better use of the large increase in search speed of Deep Thought 2 over Deep Thought (Anantharaman 1990; Hsu et al. 1990). In particular, there was an attempt made to add additional selectivity to the search while at the same time preventing the search from exploding due to excess focus on particular sub-trees. This was in part motivated by two games played between International Master Mike Valvo and Deep Thought. These games were played under correspondence conditions, i.e., days to make a move rather than minutes. It was clear to us that under these conditions Valvo was stronger than Deep Thought tactically.

The resultant search was highly selective, often reducing the full-width search depth by two ply or more (in Deep Blue) and concentrating this effort on the "important" lines of play. The search speed of the full Deep Blue system allows this kind of depth reduction while still providing "insurance" (Junghanns 1998) that nothing simple is missed. The nature of this insurance is that all lines are searched to a certain minimal depth. This minimal depth is the iteration number

In addition to the depth counter, two values were maintained, one for each side. These values, called white credit and black credit, measure the "forcedness" of the white moves and black moves that lead to the current position. A "forced" white move adds to the white credit if the move does not "fail low". Similarly, a "forced" black move adds to the black credit if the move does not "fail low". A fail-low move is, in negamax terminology (Knuth and Moore 1975), a move whose backed up value is below the current alpha-beta window. In other words, it is a move that has a refutation. There are different degrees of

forcedness, which add varying amounts to the white or black credit. This has the effect of allowing fractional extensions. In the Deep Blue implementation, a credit of ¼ ply is possible. In addition, the software search can go up to four times the nominal search depth.

To the best of our knowledge, the idea of separating the white and black depth computation was first suggested by David McAllester. A later paper (McAllester and Yuret 1993) derived an algorithm, ABC, from conspiracy theory (McAllester 1988). Apparently the ideas as described by McAllester have proven successful in improving tactical play, but not self-play performance (McAllester 1995).

There is a large set of mechanisms to identify nodes that should receive credit.

- Singular, binary, trinary, etc.: A singular move is one that is significantly better than all the alternatives [Anantharman 1990]. One can generalize this to the case where there are two, three or more good moves. Of course the more reasonable moves that are available, the less credit that should be given. It is clear that a large part of what a strong human chess player would define as forcing is covered by singularity. It is in just these kinds of positions that Grandmasters are able to calculate very deeply.
- Absolute singular: When there is only one legal move a large credit can be given with very little risk. The reason is that, if the absolute singular move ends up failing low, there are no alternatives to examine so the cost is contained. It is reasonable in many cases to give a full two ply extension.
- Threat, mate threat: It is relatively simple using a "null move search" to detect if there is a threat in the current position [Anantharaman 1991]. A null move search is a search conducted after one side passes. The intuition here is that if one side passes, then loses quickly, that side is deemed to have a pending threat against it which the other side is close to exploiting. Positions where a threat exists tend to be constrained in the number of reasonable alternatives. If a large threat exists, such as a threat to checkmate, a higher level of credit can be given.
- Influence: This mechanism gives credit for moves which are enabled by previous moves. For example, credit may be given for an apparently good white response to a black move which was not available the previous time black moved. The idea here is that we assume black is developing a combination even if we don't quite see what the combination is.

Many of these methods require auxiliary computation in order to gather the information necessary to make extension decisions. This was in line with our philosophy of using the tremendous raw searching power of Deep Blue to enable a more selective search.

There are significant implementation issues related to these search methods, especially involving interactions with the transposition table. The use of values stored in the transposition table becomes more problematic and more subject to "graph history" problems than a vanilla alpha-beta algorithm. The graph history problem arises when a node score is taken from the transposition table. The score may then be applied to a current node. This heuristic is very effective, but can fail catastrophically if the result of the sub-tree is dependent on the history leading to the current node. There is a significant role for the transposition table in preserving the envelope of the search, i.e., ensuring that previously examined lines of play are not lost as the search progresses.

We have also observed serious effects on the parallel search efficiency in some cases where a large number of extensions are possible. The parallel efficiency is inherently lowered as a result of using search extensions. The variance in execution time of a given size of sub-tree is increased due to the possibility of a search extension. This greatly complicates the task of load balancing.

The following gives a sample of the behavior of Deep Blue in two quite different positions. The first position is before White's move 37 in the game Deep Blue-G. Kasparov, Match game 2, New York, 1997, and contains some very forcing tactical lines. The second position is before Black's move 11 in the fifth game of the same match, and would not normally be considered a tactical position. For better observability, this experiment was run on Deep Blue Jr., a version of Deep Blue that runs on a single node of an IBM RS/6000 SP computer. For a given iteration *i*, the software is assigned *i*-4 ply, which represents the minimum depth search in software. The maximum depth reached in software is given in the third column. In these two positions, the maximum software depth is approximately three times the minimum depth. The last column estimates the maximum depth reached in hardware and software combined. It is not possible to directly measure this number, but the estimate is based on results of simulating the hardware search. When hardware search extensions and quiescence search are taken into account, we typically see searches of 6 to 16 ply. Thus we can see iteration 12 searches can reach as deep as 40 ply in positions of this type, which suggests that position 2 is rather tactical after all.

Position 1			
Iteration	Minimum software depth	Maximum software depth	Estimated maximum combined depth
6	2	5	11-21
7	3	6	12-22
8	4	11	17-27
9	5	15	21-31
10	6	17	23-33
11	7	20	26-36
12	8	23	29-39

Position 2			
Iteration	Minimum software depth	Maximum software depth	Estimated maximum combined depth
6	2	5	11-21
7	3	6	12-22
8	4	10	16-26
9	5	16	22-32
10	6	19	25-35
11	7	20	26-36
12	8	24	30-40

Conclusion

The methods for controlling the search in the Deep Blue chess machine have proven quite successful in actual play against Grandmasters. A number of issues remain to be explored, however. A careful and systematic evaluation of the effectiveness of these methods still needs to be carried out. Deciding on the amount of credit to allocate for various types of extensions was done on a rather ad hoc basis. It is possible to try and systematically tune these values so as to maximize performance in practical play.

References

Anantharaman, T.S. 1991. Extension heuristics. *ICCA Journal* 14(2):47-65.

Anantharaman, T.; Campbell, M.; and Hsu. F. 1990. Singular extensions: Adding selectivity to brute-force searching. *Artificial Intelligence* 43:99-109.

Baum, E.B. and Smith, W.D. 1997. A Bayesian approach to relevance in game playing. *Artificial Intelligence* 97(1-2):195-242.

Berliner, H.J. 1979. The B* tree search algorithm: A best-first proof procedure. *Artificial Intelligence* 12(1):23-40.

Berliner, H.J. and McConnell, C. 1996. B* probability-based search. *Artificial Intelligence* 86(1):97-156.

Berliner, H.J.; Goetsch, G.; Campbell, M.S.; and Ebeling, C. 1990. Measuring the performance potential of chess programs. *Artificial Intelligence* 43(1):347-367.

Heinz, E.A. 1998. DarkThought goes deep. *ICCA Journal* 21(4):228-244.

Hsu, F.; Anantharaman, T.S.; Campbell, M.S.; and Nowatzyk, A. 1990. Deep Thought. In Marsland, T.A. And Schaeffer, J. eds. *Computers, Chess and Cognition.* New York: Springer-Verlag. 55-78.

Hsu, F. 1999. The Chess Grandmaster Chips in the IBM Deep Blue Supercomputer. Forthcoming in *IEEE Micro.*

Hyatt, R.M. and Newborn, M.M. Crafty goes deep. *ICCA Journal* 20(2): 79-86.

Junghanns, A.; Schaeffer, J; Brockington, M; Bjornsson, Y; and Marsland, T.A. 1997. Diminishing returns for additional search in chess. In van den Herik, H.J. and Uiterwijk, J.W.H.M., eds., *Advances in Computer Chess 8.* University of Maastricht. 53-67.

Junghanns, A. 1998. Are There Practical Alternatives to Alpha-Beta in Computer Chess? *ICCA Journal* 21(1):14-32.

Knuth, D.E. and Moore, R. 1975. An analysis of alpha-beta pruning. *Artificial Intelligence* 6:293-326.

McAllester, D.A. 1988. Conspiracy numbers for min-max search. *Artificial Intelligence* 35:287-310.

McAllester, D.A. and Yuret, D. 1993. Alpha-Beta-Conspiracy Search. Unpublished, http://www.research.att.com/~dmac/abc.html.

McAllester, D.A. 1995. Game Search and Computer Chess. Unpublished comments, http://www.research.att.com/~dmac/games.html.

Mysliwietz, P. 1994. *Konstruktion und Optimierung von Bewertungsfunktionen beim Schach.* Ph.D. Thesis, University of Paderborn.

Palay, A.J. 1985. *Searching with Probabilities.* London: Pitman.

Russell, S.J. and Wefald, E. 1991. *Do the Right Thing.* Cambridge, MA:MIT Press.

Shannon, C. E. 1950. Programming a computer for playing chess. *Philosophical Magazine* 41: 256-275.

Thompson, K. 1982. Computer chess strength. In Clarke, M.R.B., ed., *Advances in Computer Chess 3.* Pergamon Press. 55-66.

Turing, A. M. 1953. Chess. In Bowden, B.V., ed. *Faster than thought*. London: Pitman. 286-295.