THE PRINCIPLE OF PRESSURE IN CHESS *

Deniz Yuret
MIT Artificial Intelligence Laboratory
545 Technology Square, Rm:825
Cambridge, MA 02139, USA
email: deniz@mit.edu

Abstract

This paper presents a new algorithm, "Pressure Search," for growing min-max game trees. The algorithm is based on the idea of best-first search. The goal of the search is to find a strategy which will change the estimated value of the current position. The amount of pressure, defined as inversely proportional to the number of options available to the opponent, is used as a heuristic measure of the remaining distance to the goal. Pressure Search has the potential to selectively extend parts of the search tree and thus discover deep combinations faster than standard alpha-beta search.

1 INTRODUCTION

The game of chess has often been called the "drosophila" of the artificial intelligence research. The history of computer chess dates back to the dawn of artificial intelligence in the 1950's and it has been an active research area since then [5, 4, 2]. The interest in computer chess has been partly due to the popularity of the game, and partly due to its fame as a representative of intelligent activity - at least until computers started to beat the grandmasters. Nevertheless, the heuristics from computer chess research and the ideas from other areas of artificial intelligence built upon each other. This paper presents one such fruitful combination of ideas.

Claude E. Shannon, considered the father of the field of computer chess, provided the basis for much of the future research in his seminal paper, "Programming a Computer for Playing Chess", dated back to 1949 [10, 4]. In spite of 45 years of work, most of his ideas are still valid today. Most significant is his definition of two basic strategies for the development of chess programs. The *type A* strategy uses what we call brute force search. Shannon claimed

*This paper describes research done at the Artificial Intelligence laboratory of the Massachusetts Institute of Technology. Support for the laboratory's artificial intelligence research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-91-J-4038.

pos.	program	hardware	nodes/sec
1.	Socrates	IBM PC 486 50Mhz	12K
2.	Cray Blitz	Cray YMP-C90	500-1000K
3.	StarTech	Connection Machine CM5	150-200K
4.	Hitech B*	Sun 4 + special	120K
5.	Zarkov	HP9000/735	15K
6.	Schroder	Laptop + special	8K
7.	Kallisto	IBM PC 486	10-25K
8.	BP	IBM PC 486	2.4K
9.	NOW	IBM PC 486	5-8K
10.	M-Chess Pro	IBM PC 486	6K
11.	Bebe	special	40K
12.	Innovation	Macintosh Quadra 700	2K

Table 1: The competitors of the 23rd ACM International Computer Chess Championship

that *type A* programs would be slow and weak players due to the exponential growth of the search tree. He quotes De Groot's experimental work on chess masters [1] to argue for the *type B* strategy, which corresponds to selective search.

In spite of Shannon's great insight, the work on competitive chess machines has been dominated by programs closer to *type A*. During the period since Shannon's paper, the speed of computers has doubled approximately every two years. The success of the programs has usually been due to the speed of hardware rather than the superiority of algorithms - an unfortunate fact for artificial intelligence research.

Selectivity, ignoring moves that look hopeless, has not been considered a good idea for chess playing programs. Experience has shown that programs that depend heavily on selectivity often fall into blunders. The argument is that no matter how good the criteria for hopeless moves is, it will sometimes eliminate moves that would lead to good results further down the search tree. Whether this failure is the fault of the idea of selectivity or its implementations is controversial.

This controversy was brought into daylight with the recent success of a program based on selectivity. In the 23rd ACM International Computer Chess Championship held in 1993, the program Socrates, running on a PC, at the speed of 12,000 nodes per second, finished in front of Cray Blitz, running on a Cray super computer at a million nodes per second, and StarTech running on a Connection Machine at 200,000 nodes per second [Table 1][3]. A PC beating two powerful super computers raised questions about the brute force approach.

In the following section we will present a game independent methodology that captures the essence of heuristics like selectivity and extensions. The results of an experimental program based on these ideas will follow.

2 THE DISTANCE TO GOAL HEURISTIC

In classical AI search literature, a very well known technique to improve performance is to use some heuristic measure of the remaining distance to the goal [12]. If such a measure is available, one can order potential continuations accordingly, and proceed with the best one.

It is not obvious however, how to apply this technique to game search. If the goal is *winning*, the solution to the whole search problem is required to calculate a good estimate of the distance to the goal. Thus this estimate cannot be used to guide the search.

The real goal of game search, we claim, is to discover combinations that will change the value of a position. The initial guess of the value of a position is determined by a static formula called the evaluation function. Search can be viewed as part of this evaluation function that discovers the weak points of this guess by looking forward in the tree. Search will find the combinations that transpose the position into another that is significantly better or worse for one side. Thus it will correct the initial errors of evaluation.

Having defined the goal of search, the distance to the goal becomes “How hard it is to find a strategy that will change the value of the current position.” A strategy for one side is defined as a subtree of moves that typically contain “the right moves” for that side for all of the opponent’s responses. If the subtree were missing some moves of the opponent, it would not be a sound strategy.

A good estimate for this distance is the number of options available to the opponent. The definition of option we will use here is a move that at least restores the balance for the side that is making the move. This definition can be improved upon, but it will serve the purpose for our present discussion.

Suppose you are considering two threatening moves. Your opponent has two options after the first threat, and four options after the second. In some sense the first move is putting more pressure on him. If we assume that it would take the same amount of work to defeat each of those sensible responses, the first move will lead to victory with half as much work.

Note that this strategy will tend to discover the most concise combinations first. The standard alpha beta procedure is blind to this “expected work” heuristic. Thus it will spend as much energy on a promising line of play with a lot of pressure, as on another with lots of open ends.

This pressure heuristic is not to be confused with the “mobility” component typical chess programs use in their evaluation function. The “mobility” component actually modifies the value of certain positions and permanently changes the evaluation of the program. In contrast, the pressure heuristic just directs the search into more promising lines of play, without altering the program’s assessment of different positions. Since the time it takes to justify the soundness of a strategy is very much related to which moves are considered first, the pressure heuristic is able to improve efficiency substantially.

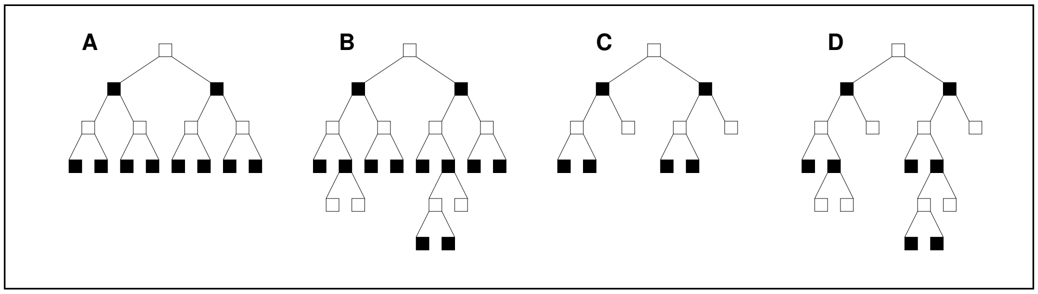

Figure 1: The four figures illustrate the shape of the trees grown by various procedures. (A) Standard min-max procedure. (B) Extensions. (C) Selectivity. (D) Pressure search generalizes non-uniform tree growth.

3 THE PRESSURE SEARCH ALGORITHM

There are two classes of non-uniform tree growth heuristics commonly used in the computer chess literature, extensions and selectivity [Figure 1]. Standard alpha beta search grows the tree to a constant specified depth down all the considered branches. An extension heuristic chooses some forced lines and extends them beyond that constant depth. A selective heuristic cuts off hopeless lines of play before they reach that depth. As illustrated in the figure, they are really two faces of the same coin. If no constant depth is specified at all, they add up to saying: "If there is a promising line with a lot of pressure, extend, otherwise if the position is passive and looks stable, then stop searching."

In fact both selectivity and extensions can be performed at the same time by using the heuristic described in the preceding section. Each time a new node is to be extended, the one that exerts highest pressure is selected. This will push the high pressure lines deeper, and will not spend much effort on stable positions.

> *Basic Principle:* The side that wants to exert pressure on his opponent should first consider the moves that leaves the opponent with the minimum number of options. [Figure 2 (a)]

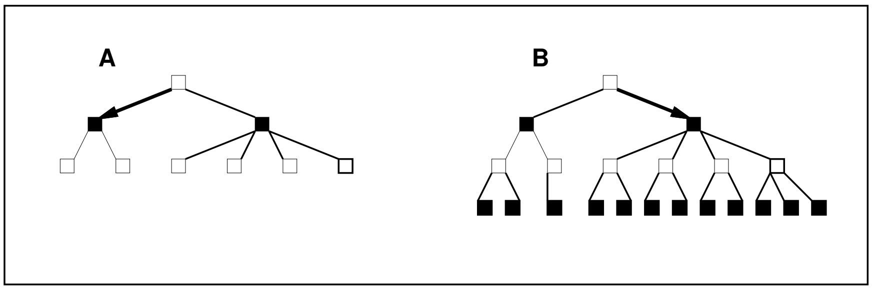

Figure 2: The figures show in which order the nodes are extended. In the first figure, the attacking player (in this case white) extends the leftmost move first, since it offers its opponent the minimum number of options. In the second figure, although the situation is similar, white extends the rightmost move as the defensive player.

It is important to understand the asymmetry of this process. One side has to be thought of as the attacker and the other side as the defender. This corresponds to the offensive and defensive thinking of real chess players. When planning an attack some lines of play are considered, when thinking defensively other lines are considered. So, in pressure search, the defensive side considers the move that would free him the most rather than trying to constrain the attacker.

> *Asymmetry:* The defensive side first considers the move that would give him the maximum number of sensible moves on his next move. [Figure 2 (b)]

To generalize this process to higher levels in the tree, we use a simple scoring mechanism: [Figure 3]

- The score at the nodes near the leaves of the tree is equal to the number of options available to the defender.
- The score at a defender node is equal to the sum of the scores of its children.
- The score at an attacker node is equal to the minimum of the scores of its children.

If we work this up the tree, each node's score will represent the number of options of the defender down the subtree starting from that node. Once we have these numbers we can use the following procedure to decide which node to extend next:

- At an attacker node extend the child with the minimum score.
- At a defender node extend the child with the maximum score.
- If the scores of the children are not available, extend all of them.
- If there are no possible extensions, mark the move as a winner.

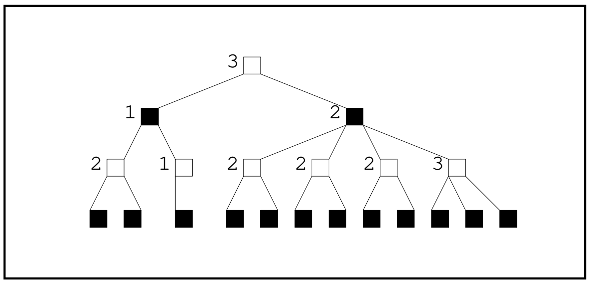

Figure 3: The scoring mechanism to calculate the number of options available at the higher nodes of the tree.

To summarize, the pressure heuristic is used to decide how deep to search on a particular line and which node to extend next in the search tree. It is not in any way combined with the static evaluation function. Thus it sometimes cuts the search shorter than where standard alpha-beta would reach, and sometimes dives deeper on promising lines of play. It is neither a purely selective algorithm or an extension algorithm, it is the generalization of both. Most selectivity and extension heuristics are special cases of the pressure idea. By analyzing the idea directly, we have developed a game independent principle that improves the performance of min-max search, and helps us understand how some of the popular techniques actually work.

4 RESULTS AND RELATED WORK

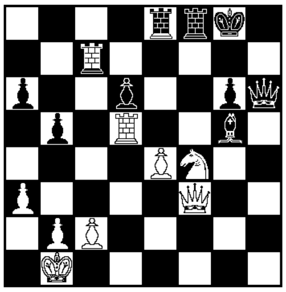

Figure 4: This problem, taken from one of Fischer's games, was solved by the experimental program in about a minute on a Sparc 10.

4.1 Initial Results

An experimental chess program was built based on these ideas. The evaluation function consisted of the material values of pieces. The program had surprisingly successful results on some test problems [7, 8, 11]. In most cases it had no problem extending beyond 10 ply down some variations within the first minute of its search. The example given in [Figure 4] show a particular case taken from a game of Bobby Fischer [7, game 14], where some lines extend to 15 ply. Although the program is at an early stage and is inefficient, it takes approximately one minute to search 50000 nodes and uncover this combination.

This naive implementation was not able to outperform alpha-beta consistently. The best first approach requires the whole tree to be kept in memory. This makes it impossible to solve problems that would take more than a few minutes. Since the evaluation function is very simple, the algorithm was not able to assess positional advantages. In positional problems where all moves give similar values, it loses its advantage over alpha-beta.

However, the ability to discover some deep combinations after analyzing relatively few moves was promising. Thus at this stage it could be used as a "combination detector" supplement to a general chess program. For a more general application, a better evaluation function and an improved memory management strategy is required.

4.2 Conspiracy Numbers

Conspiracy numbers for game search were introduced by McAllester [6, 9]. The procedure he suggests is based on a measure of the "accuracy" of the root value. The min-max value of the root of a game tree is a function of the static evaluations of the leaf nodes. The root value is more dependent on some of these leaf values than others. When a particular leaf value proves to be wrong it is said to "conspire". The minimum number of leaf values that have to conspire together to bring about a certain change in the root value is a "conspiracy number". Thus a conspiracy number is a measure of the accuracy of the root value. The game tree should be grown in a way as to maximize the conspiracy numbers.

Note the similarities and the differences of conspiracy numbers with the idea of pressure. Conspiracy search tries to grow a search tree as to make the root min-max value as accurate as possible. Pressure search tries to find the smallest combination that can change this root value. Conspiracy search improves the accuracy of the root value by growing the tree to support it. Pressure search tries to undermine the value by finding the weakest support point in the tree and attacking it.

4.3 ABC Search

Although they were discovered independently, the similar insights provided by the algorithms brought the two works together. McAllester and I started looking more closely into what made these algorithms work. We decided to demonstrate their efficacy in competitive chess programs. There were two approaches, modifying an existing program with the guidelines that our algorithms provide, or creating a new program after solving problems about memory and efficiency.

The first route worked quite well. The team of the current world champion chess program Socrates started adding new features to their program under our advice. In particular they implemented the idea of using two different modes of search, the "attacker" mode, and the "defender" mode. A particular node at some level in the tree might be sufficiently deep from one point of view, but may need more exploration from another. They implemented two depth parameters to represent these two modes. Their program was a result of years of careful hill-climbing, and their advantage rested on a carefully optimized selectivity. However this idea brought about a substantial leap in the performance. Socrates participated in the *4th Harvard Cup*, a championship in which the best computer programs compete against the best human masters. It demonstrated its performance by winning three points from six of the best players in the U.S.A.

The second approach proved to be very challenging. To cope with the memory problem, we designed a combination of our algorithms with alpha-beta. This algorithm, named ABC-Search, requires linear amount of memory with the depth. It combines the ideas for tree growth and move ordering with classical alpha-beta. A flexible two depth approach allows non-uniform tree growth. Work is still in progress. We expect to have an algorithm that combines the robustness of alpha-beta with the efficiency of principled non-uniform tree growth.

References

[1] A. D. De Groot. *Thought and choice in chess.* Mouton, the Hauge, 1965 (English translation of the original Dutch edition of 1946).

[2] P. W. Frey, editor. *Chess skill in man and machine.* Springer-Verlag, New York, 1983.

[3] D. Kopec. The 23rd acm international computer chess championship. *International Computer Chess Association Journal*, 16:38–53, March 1993.

[4] D. Levy, editor. *Computer chess compendium.* Springer-Verlag, New York, 1988.

[5] Marsland and Schaeffer, editors. *Computers chess and cognition.* Spriger-Verlag, New York, 1990.

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

[7] B. Pandolfini. *Bobby Fischer's outrageous chess moves.* Simon & Schuster, New York, 1985.

[8] F. Reinfeld. *Win at chess.* Dover, New York, 1958 (originally published in 1945 under the title 'Chess Quiz').

[9] J. Schaeffer. Conspiracy numbers. *Artificial Intelligence*, 43:67–84, 1990.

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

[11] S. Tarrasch. *The game of chess.* Dover, New York, 1987 (originally published in 1935).

[12] P. H. Winston. *Artificial intelligence, 3rd ed.* Addison-Wesley, Massachusetts, 1992.