The Significance of the Low-Level Heuristics in Chess Searching Algorithms

Vladan Vučković [1]

Abstract - **This paper is concern with the low-level heuristics implementation in the author's *Geniss Axon XP* chess application. The heuristics represent the expert knowledge and could be used with a great efficiency in searching algorithms in cooperation with standard Alfa-Beta pruning technique. The empirical tests generated on two grandmaster games proves the high potentials of this combination of pruning techniques.**

Keywords - **Computer chess, heuristics, chess searching algorithms, programming.**

I. Introduction

According to Shannon underlying works **[1],[2]** the computer chess is one of the basic directions of the artificial intelligence. The principles of machine "thought" in chess are radically different from the human ones. Namely, the very first works in this array proved that it is very difficult to mimic the thought processes of the strong chess player with the computer. The idea was to observe and analyze the steps which human chess player must go through when decide what move is the best to play. If one could define these steps into the algorithm manner, the computer could be easily programmed to follow them. But unfortunately this approach did not shunt to the chess computer playing with grandmaster strength. The problem is that a large amount of chess knowledge, sometimes essential parts, stay beyond the horizon. For instance, the chess intuition and position "feel" are something possessed by the strong chess players and could not be defined and transferred to machine in simple way.

The real theory of computer chess is based on different methods of calculations **[6]**. First, computer orientation is rather "brute-force" than using the smart, selective searching **[15],[16]**. The first successful brute-force program was *Chess 4.0* **[3]**. The program logic is simple, machine calculates all legal variants (generated by all moves in every position) to the fixed depth. Every tactical or strategic move below search horizon will be found. This is the simplest way to make a chess program which strength is in function only of the searching horizon. With the fast hardware the computers could achieve master level of play.

[1] Vladan Vučković is from the Faculty of Electronic Engineering, University of Ni{, 18000 Ni{, Yugoslavia, Email: vld@elfak.ni.ac.yu

II. Alfa-Beta Pruning Technique

The main problem with the brute-force approach is combinatory explosion which is avoidable **[3],[8],[10]**. For instance, if the primary chess position has 20 legal moves by one side, each of them generate 20 moves by the other side which is 400 positions in total. With horizon which depth is only 5 (beginner level of play) machine has to calculate $20^5 =$ 3.2 million positions. This enormous number of positions could not be suppressed even with the fastest hardware. That is the main reason why researchers in this array try to reduce the number of nodes and branches in chess game tree as much as possible **[3],[4],[7],[11]**.

The basic method for pruning the chess tree is the Alfa-Beta pruning technique **[5],[9]**. This method enables to skip over a large parts of the tree which are irrelevant to the final decision. The method is absolutely safe and gives the same best moves in all cases compared to the standard full-width searcher. The logic of the pruning technique could be described on the following sample:

The presumption is that the somewhere into the decision tree we have the node with white side on the move. If one tracks the variant from that node to the route it is possible to determinate the best evaluation of the opponent side on the move (black). This value is called *Beta* value. Now, if the searching process in white node found the continuation which achieves and overcomes the *Beta* value the procedure could immediately stop the further computing because it is irrelevant. The new value could never be perforated to the root because the better value for blacks (Beta) has already found. The analogue method is used for black nodes opposite to white ones to get the Alfa value. The Alfa-Beta prunes could achieve maximum of efficiency when the best successor position is searched first, because this position will either be part of the principal variation or it will cause a cutoff to be as early as possible. If the W is average number of moves per node and D is depth the Alpha-Beta still has to search $W^{(D+1)/2} + W^{D/2} - 1$ positions which compared to W^D gives a double horizon within the same amount of time.

III. The "Intelligent Pieces" Conception

In this chapter the author will present briefly his new theoretical concept of "intelligent pieces" enabling the framework for inducting the low-level heuristic pruning techniques. After the broadly inspection of the Alfa-Beta pruning the limitations of the technique is occurred. It is interesting fact that the Alfa-Beta has general approach and could be applied on all logic games, not only for chess game. This universality limits the technique. If one wants to go further in the process of tree pruning, the next step must be induction of the specific chess knowledge into the searcher. The main question is where and at what level is the best to implement the knowledge. The rules implemented into the program in the purpose for handling the chess knowledge are called *heuristics* **[3],[12]**. The *Geniss Axon XP* **[13],[14]** application has implemented author's innovative conception called "intelligent pieces". Namely, the majority of chess programs concentrate on move generators with a moves as the base of the searching process. The authors conception is that basics of the searching algorithm could be chess pieces. With simplification, the procedure is:

- Scan the chess board,
- Locate the positions of all pieces on the board,
- Generate the machine call instruction for each of the piece,
- Execute the code associated with each piece.

This conception automatically solve the problem where to implement the chess knowledge - the subroutines connected with each piece are the ideal place. There are 12 different subroutine connected with each piece, 6 for white and 6 for black. Each subroutine encapsulates machine code enabling the piece to recognize different structures on the current position and to generate move and weight in function of them. Of course, the recognizing procedure is simplified because the source piece is precisely defined. The analogue logic is used also for the realization of the evaluation function.

IV. Heuristic Pruning in Geniss Axon XP

The heuristics, embedded in chess piece subroutines, recognize some structures and generate corresponding bits into the machine weight word. The format of the word is 16-bits with following meanings:

- Special move bit,
- Safe bit,
- Check bit,
- Capture bit,
- Null move attack bit,
- Double attack bit,
- Escape bit,
- Force move bit,
- Attack/Defend bit,
- Queen/Rook/Knight/Pion piece identification bit,
- Null move initiator bit,
- "Touch piece" bit,
- Strategic move bit.

The bits are ordered according to their weights. The numeric values (binary representation) directly guide the searching process, because they generate the move ordering at the same time. In *Geniss Axon XP* the heuristic pruning mechanisms are complex and the detail recapitulation of them will surely significantly overcome the format of this paper. The two of them are the most important and they will be mentioned. On Fig. 1. one position is diagrammed with the heuristic bits associated with it:

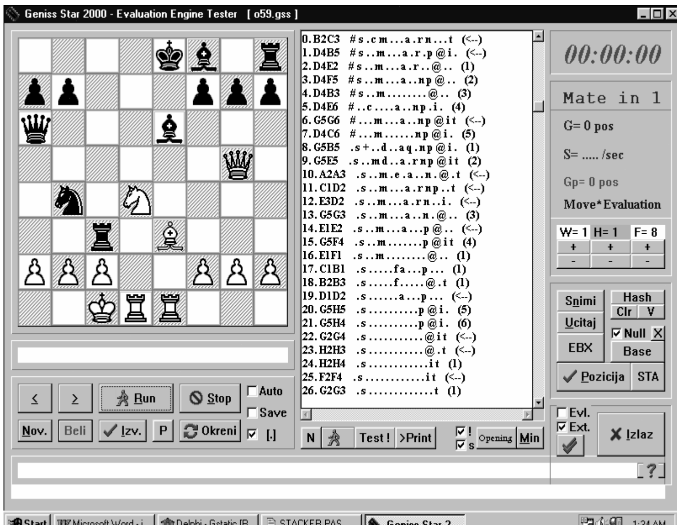
```
0.B2C3  #s.cm...a.rn...t  (<--)
1.D4B5  #s..m...a.r.p@i.  (<--)
2.D4E2  #s..m...a.r..@..  (1)
3.D4F5  #s..m...a..np@..  (2)
4.D4B3  #s..m........@..  (3)
5.D4E6  #..c....a..np.i.  (4)
6.G5G6  #...m...a..np@it  (<--)
7.D4C6  #...m......np@i.  (5)
8.G5B5  .s+..d..aq.np@i.  (1)
9.G5E5  .s..md..a.rnp@it  (2)
10.A2A3  .s..m.e.a..n.@.t  (<--)
11.C1D2  .s..m...a.rnp..t  (<--)
12.E3D2  .s..m...a.rn..i.  (<--)
13.G5G3  .s..m...a..n.@..  (3)
14.E1E2  .s..m...a...p@..  (<--)
15.G5F4  .s..m.......p@it  (4)
16.E1F1  .s..m........@..  (1)
17.C1B1  .s.....fa...p...  (1)
18.B2B3  .s.....f.....@.t  (1)
19.D1D2  .s......a...p...  (<--)
20.G5H5  .s..........p@i.  (5)
21.G5H4  .s..........p@i.  (6)
22.G2G4  .s............@it  (<--)
23.H2H3  .s............@.t  (<--)
24.H2H4  .s.............it  (1)
25.F2F4  .s.............it  (<--)
26.G2G3  .s..............t  (1)
```

Fig 1. Position with heuristic bits generated.

One could notice that the heuristic bits determine the move ordering process. The two major heuristic pruning mechanisms are:

- ***horizontal heuristic pruning*** - if the search procedure does found the good continuation at the current node, its positive value overpowers the further moves and prune them in function of their heuristic bits. For example, if the searcher found the queen capture as the good move it is irrelevant to explore the capture moves on pions with exception if they lead to mate treats or promotions.

- ***vertical heuristic pruning*** - the analogue with previous technique but works with the good move found by tracking vertically to the tree root.

The heuristic pruning techniques in cooperation with the standard Avlfa-Beta serves much better tree searching results. The next chapter shows some empirical data affirming that conclusion.

V. Comparation Results

In this chapter two tests will be performed to prove a great profit by using the chess heuristics. The examples are from the meridian grandmaster games. Tables I and II present the data generated only with pure Alfa-Beta technique, while Tables II and IV present results when heuristics are included. The tables show that, with combination of Alfa-Beta, heuristics and search extensions key moves could be found relatively fast, at lower depths. The *Geniss Axon XP* application running on

Pentium I 200 MHz machine (64 Mb RAM) is used to perform the tests.

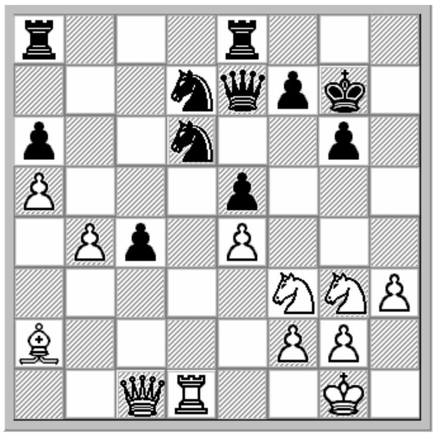

Fig 2. The position from the Karpov-Spassky game (1973). The wining move for white is Qg5!!.

TABLE I
TABLE SHOWS COMPARATION RESULTS OF THE POSITION (FIG. 1) ANALYZING WITHOUT HEURISTIC PRUNING

De-pth	Positions	Time (sec)	Move	Evaluation	T.G.F
1	407	<< 1s.	C1D2	+1.30	-
2	5439	1 s.	C1D2	+1.30	13.36
3	19084	3 s.	C1D2	+1.30	3.508
4	244872	32 s.	C1G5!	+1.38	12.83
5	687547	1:27 s.	C1G5!	+1.38	2.808
6	6360400	13:18 s.	C1G5!	+1.44	9.251

TABLE II
TABLE SHOWS COMPARATION RESULTS OF THE POSITION (FIG. 1) ANALYZING WITH HEURISTIC PRUNING

De-pth	Positions	Time (sec)	Move	Evaluation	T.G.F
1	407	<< 1 s.	C1D2	+1.30	-
2	1169	1 s.	C1D2	+1.30	2.872
3	3391	1 s.	C1D2	+1.30	2.900
4	10196	2 s.	C1D2	+0.59	3.006
5	40985	5 s.	C1G5!	+1.38	4.019
6	91550	12 s.	C1G5!	+1.40	2.233
7	221544	28 s.	C1G5!	+1.30	2.430
8	539533	1:06 s.	C1G5!	+1.18	2.435
9	2518123	4:55 s.	C1G5!	+1.18	4.667

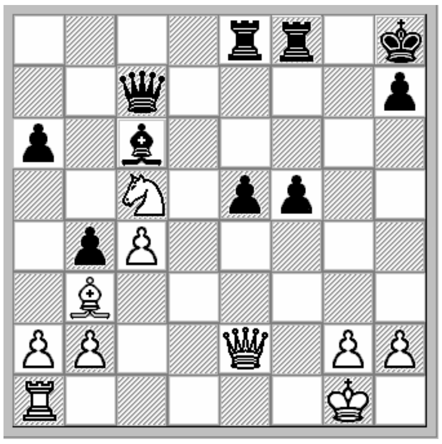

Fig 3. The position from the Fischer-Tal game (the tournament of candidates, 1959.). The winning move for black is Bg2!!.

TABLE III
TABLE SHOWS COMPARATION RESULTS OF THE POSITION (FIG. 2) ANALYZING WITHOUT HEURISTIC PRUNING

De-pth	Positions	Time (sec)	Move	Evaluation	T.G.F
1	756	<< 1 s.	C6G2!	+2.29	-
2	6678	1 s.	C6G2!	+1.79	8.833
3	27567	3 s.	A6A5	+1.73	4.128
4	228819	26 s.	A6A5	+1.86	8.300
5	766494	1:26 s.	C6G2!	+1.88	3.349
6	7362057	13:35 s.	F8G8!	+2.10	9.604

TABLE IV
TABLE SHOWS COMPARATION RESULTS OF THE POSITION (FIG. 2) ANALYZING WITH HEURISTIC PRUNING

De-pth	Positions	Time (sec)	Move	Evaluation	T.G.F
1	756	<< 1 s.	C6G2!	+2.29	-
2	2281	<< 1 s.	C6G2!	+1.79	3.017
3	6278	1 s.	F8G8!	+1.68	2.752
4	17975	2 s.	C6G2!	+1.82	2.863
5	105336	12 s.	C6G2!	+1.88	5.860
6	247685	29 s.	C6G2!	+1.82	2.351
7	936609	1:48 s.	C6G2!	+1.71	3.781
8	1528003	2:55 s.	C6G2!	+1.99	1.631
9	6091125	11:28 s.	F8G8!	+1.99	3.986

The analyses of the empirical data show that heuristic pruning implemented in chess application could significantly reduce the number of positions computed in game tree. If one compares the results among the tests with and without heuristics it is obvious that if the heuristics are included the same results of searching (key moves) are generated in the much less amount of time. Also, the greater depths could be achieved. The next conclusions could be postulated:

- At the same search depth, heuristic pruning save enormous number of positions. For example, at depth 6, (Tables I and II) if no heuristic are included program must generate 6360400 positions but with heuristics only 91550 which is only 1.4% of the primary number of pos. The key moves are correct in both cases (c1g5!). The similar situation is with Fig. 3 (Tables III and IV), where the pruned tree is about 3.3% of the source one.

- The Tree Growth Factor, which is the indicator of the combinatory explosion is much less when heuristics are included (Fig 2., Tables I and II , at depth 6, 2.233 compared to 9.251 and with Fig 3., Tables III and IV, 2.351 compared to 9.604).

- The searching are more stabile when heuristics are included. The key moves are found earlier, and remain unchangeable in next search iterations.

VI. Conclusion

The main intention of this paper is to present some new concepts connected with heuristic induction into the chess searching process. The authors concept of "intelligent pieces" could be very solid framework for integrating the expert knowledge with the searching algorithm and programming it at low-level machine language. The results in Chapter V show that the heuristics could provide a great savings in generated positions. The extra time obtained in that way could be used to achieve the greater searching depth and to increase the power of machine play. On the contrast of the Alfa-Beta pruning technique, which is absolutely safe, the heuristic pruning could make false cut and reject some good move which is not within the current heuristic. But if the heuristic is applied on the instant position, the tree pruning could be enormous. Of course, the theory and practice of the chess heuristic pruning are at the beginning but their developing will be very benefit for the increase of power of the computer chess play.

References

[1] Claude E. Shannon “Programming a computer for playing chess”, first presented at the *National IRE Convention*, March 9, 1949. New York, U.S.A

[2] Claude E. Shannon “A chess-playing machine”, reprinted with permission, *Scientific American*, 1950, USA

[3] Peter W. Fray *Chess Skill in Man and Machine*, texts and monographs in computer science, Springer-Verlag, 1977,1978, New York, USA

[4] *Search Algorithms,* http://fbi001.onformatik.fh-hamburg.de/~owsnicki/search.html, 2000.

[5] *Analysis of Alpha-Beta Pruning,* http://www.npac.syr.edu/copywrite/pcw/node351.html, 2000.

[6] T.A.Marsland, *The Anathomy of Chess Programs*, ICCA homepage, http://www.icca.com, 1999

[7] Allis, L.V. Ingo Althöfer: “On pathology in game tree and other recursion tree models” *ICCA Journal*, Vol. 15, No. 2, p. 80. (1992).

[8] Donskoy, M.V. “Fundamental concepts in search” *ICCA Journal*, Vol. 13, No. 3, pp. 133-137. (1990).

[9] Junghanns, A. “Are there practical alternatives to Alpha-Beta?” *ICCA Journal*, Vol. 21, No. 1, pp. 14-32. (1998).

[10] Kaindl, H., Horacek, H. and Wagner, M. “Selective search versus brute force” *ICCA Journal*, Vol. 9, No. 3, pp. 140-145. (1986).

[11] Marsland, T.A. “A review of game-tree pruning” *ICCA Journal*, Vol. 9, No. 1, pp. 3-19. (1986).

[12] Schaeffer, J. “The history heuristic” *ICCA Journal*, Vol. 6, No. 3, pp. 16-19. (1983).

[13] Vuckovic, Vladan "The Basic Elements of the Chess Program Implementation", *Proceedings of a Workshop on Computational Intelligence and Information Technologies*, pg. 17-23, June 20-21. 2001.

[14] Vuckovic, Vladan "Decision Trees and Search Strategies in Chess Problem Solving Applications", *Proceedings of a Workshop on Computational Intelligence and Information Technologies*, pg. 141-159, February 27. 2001.

[15] Bruin, A. de, Pijls, W. and Plaat, A. “Solution trees as a basis for game-tree search” *ICCA Journal*, Vol. 17, No. 4, pp. 207-219. (1994).

[16] Anantharaman, T.S., Campbell, M.S. and Hsu, F. “Singular extensions: adding selectivity to brute-force searching” *ICCA Journal*, Vol. 11, No. 4, pp. 135-143. (1988).