PREPRINT OF AN ARTICLE* IN: ICCA JOURNAL 21(2), PP. 75–83, JUNE 1998.

Extended Futility Pruning

Ernst A. Heinz†
Institute for Program Structures and Data Organization (IPD)
School of Computer Science, University of Karlsruhe
P.O. Box 6980, D-76128 Karlsruhe, F.R. Germany

March 31, 1998

Abstract

This article presents a new selective pruning technique for alpha-beta based game-tree search in computer chess. *Extended futility pruning* builds on ideas of both deep razoring and normal futility pruning at frontier nodes (depth = 1). It cuts complete branches of the search tree at pre-frontier nodes (depth = 2) according to solely static criteria at the respective nodes and thus performs true forward pruning.

Although extended futility pruning is theoretically unsound, extensive experiments with our master-strength chess program DARKTHOUGHT[1] show that it works markedly well in practice. Even at fixed search depths, extended futility pruning exhibits hardly any loss of tactical strength while shrinking the search trees by 10%–20% on average as compared with normal futility pruning.

Furthermore, extended futility pruning combines nicely with a conservatively limited variation of razoring that reduces the search trees at fixed search depths > 10 plies by additional 5%–15% on average. Last but not least, the presented pruning schemes scale very well with search depth and reap ever more benefits at higher depths.

1 Introduction

Until the advent of TECH [9] and CHESS 4.X [23] in 1972–1973 most chess programs were highly selective alpha-beta searchers. They mainly relied on knowledge-intensive plausible move generators that cut off statically "implausible" moves at nodes of all levels and thus pruned large parts of the

*The published version of the article won the international Prof. Salvatore Award 1998.
†Email = <heinze@ira.uka.de>, WWW = <http://wwwipd.ira.uka.de/~heinze/>
[1] WWW = <http://wwwipd.ira.uka.de/Tichy/DarkThought/>

search tree. But as soon as brute-force searchers like CHESS 4.X reached depths ≥ 5 plies in the middlegame, they routinely punished the tactical shortcomings of plausible move generation and started their own reign.

Except for the triggering of variable-depth search extensions, typical brute-force chess programs of this era featured no selectivity at all in the full-width parts of their searches. For avid proponents of the brute-force search paradigm deem any kind of forward cutting too suspicious and unsafe as to employ it in their alpha-beta searchers. These true brute-forcers do not even trust normal futility pruning at frontier nodes (see Section 3).

In 1989–1990 Beal [4] and Goetsch & Campbell [10] reported about their experiments with the null move which initiated a renaissance of selective searching in computer chess. Null-move pruning exhibits only minor tactical weaknesses and scales very well with search depth. The spectacular success of null-move pruning in microcomputer chess made it one of the most popular alpha-beta enhancements today.

State-of-the-art chess programs using null-move pruning easily reach search depths of 10–12 plies in the middlegame on fast microcomputers. Thence, the field of computer chess now needs new techniques that make alpha-beta searching more scalable at high search depths > 10 plies. The prospect of ever more powerful computers further adds to this need.

We believe that selective forward pruning schemes promise good chances for improvement in this area. A prerequisite for the practical success of any such scheme will be to preserve the tactical strength of the search. Both our new pruning techniques *extended futility pruning* (see Section 4) and *limited razoring* (see Section 5) fulfill this requirement and scale nicely.

2 Related Work

Although it enjoys wide-spread use there does not exist much published material about futility pruning at frontier nodes. Schaeffer briefly explains the underlying futility idea in his 1986 Ph.D. thesis [19]. To the best of our knowledge, this is the earliest publication that presents quantitative experimental data illustrating the effectiveness of futility pruning at frontier nodes. Ye and Marsland's 1992 article [24] about forward pruning and search extensions in Chinese chess supports Schaeffer's findings by independent experiments. Section 3 discusses futility pruning at frontier nodes in detail.

Marsland's excellent (yet meanwhile somewhat outdated) overview of computer chess and search from 1992 [14] mentions three other statically selective pruning techniques that gained broad attention in the early days of computer chess. But Newborn's GAMMA algorithm [16], Slagle's marginal forward pruning [22], and Birmingham & Kent's razoring [5] seem to generate only little interest today (if any at all) because their aggressive selectiveness incurs too many tactical risks. Nevertheless, we deem the fundamental ideas of these forward pruning schemes as still very valuable. With the right additions and modifications, static forward pruning following a similar spirit like the above can work great as presented in the remainder of this article.

Null-move pruning [4, 10] and Buro's PROBCUT enhancement [6] differ from the selective pruning schemes mentioned so far because they base their pruning decisions on dynamic rather than static criteria at the respective nodes. Both try to save work by speculatively replacing normal searches with much cheaper searches of reduced depths. The full-width nature of the reduced searches makes these two forward pruning techniques quite safe and successful in practice.[2]

3 Futility Pruning at Frontier Nodes

The well-known technique of futility pruning at frontier nodes (depth = 1) exploits the peculiarities of "standing pat" at horizon nodes (depth = 0). "Standing pat" means to compute the static evaluation score of a node in order to test it against the upper search bound for a possible fail-high beta cutoff without further lookahead. Thus, "standing pat" implements the null-move assumption during the quiescence search with static node evaluations serving as null-move scores of zero depth. The following two sections describe the theory and practice of futility pruning at frontier nodes in detail because they provide the foundations for our new pruning scheme.

3.1 Theory

Futility pruning at frontier nodes saves a lot of work by deciding in advance which moves will cause a "stand pat" beta cutoff at the direct successor nodes on the search horizon. This requires the search to determine lower bounds for the static evaluation scores of horizon nodes based on the situations and moves at frontier nodes. Such bounds are easy to obtain for non-checking moves and typical chess evaluation functions that simply add material balances and positional scores.

[2] Although Buro developed and tested PROBCUT within his master-strength othello program LOGISTELLO it should work in computer chess as well.

Lower Bounds for Static Evaluation Scores

$$eval\_score(node) \;=\; mat\_balance(node) \,+\, posn\_score(node) \tag{1}$$

$$0 \;\leq\; |\,posn\_score(node)| \;\leq\; max\_posn\_score \in \mathbf{N}$$

$$\Downarrow$$

$$eval\_score(node) \;\geq\; mat\_balance(node) \,-\, max\_posn\_score \tag{2}$$

With max_posn_score denoting a constant upper bound on the absolute values of all positional scores, the term $mat\_balance(node) - max\_posn\_score$ represents a tight lower bound for the static score of a node as computed by typical evaluation functions. This tight lower bound allows for the formulation of a theoretically sound futility condition at frontier nodes that enables the advance detection of subsequent "stand pat" beta cutoffs for non-checking moves. The theoretical soundness of the futility condition follows from the well-known relations of alpha, beta, and the material balances at direct successors of a node.

Alpha / Beta and Material Balances at Direct Successor Nodes

$$alpha(node) \;=\; -beta(succ(node, move)) \tag{3}$$

$$mat\_balance(succ(node, move)) \;=\; -mat\_balance(node) - mat\_gain(move) \tag{4}$$

Theoretically Sound Futility Condition at Frontier Nodes

$$mat\_balance(node) \,+\, mat\_gain(move) \,+\, max\_posn\_score \;\leq\; alpha(node) \tag{5}$$

$$\Downarrow$$

Futility Condition Applied to Successor Nodes

$$mat\_balance(succ(node, move)) \,-\, max\_posn\_score \;\geq\; beta(succ(node, move))$$

$$\Downarrow$$

"Stand Pat" Beta Cutoff at Horizon Nodes

$$eval\_score(succ(node, move)) \;\geq\; beta(succ(node, move))$$

Given a suitable value for max_posn_score, it is therefore safe to simply cut off all non-checking moves at frontier nodes for which the futility condition $mat\_balance(node) + mat\_gain(move) + max\_posn\_score \leq alpha(node)$ holds. The according futility cutoff-rate heavily depends on the value of max_posn_score and the material balances of the positions encountered during the search. The tighter the bound on positional scores (i.e., the smaller *max_posn_score*) and the greater the material imbalances are, the more cutoffs occur.

3.2 Practice

Real chess programs tend to sacrifice the theoretical soundness of futility pruning in favor of using it as a truly selective pruning technique. As mentioned above the tightness of the bound on positional scores greatly influences the futility cutoff-rate. Viewed the other way round, the introduction of a configurable futility margin instead of the theoretical bound *max_posn_score* allows for a variable leverage of the cutoff-rate. By choosing specific margins, chess programs can thus exert effective control over the futility cutoff-rate and the desired degree of selectiveness.

With margins ranging from 2–4 Pawns, selective futility pruning at frontier nodes proves to be quite safe in practice. While causing hardly any tactical mistakes it shrinks the search trees by 60% on average according to independent experiments by various researchers (e.g. Schaeffer [19], Ye and Marsland [24]). Many implementations increase the selectiveness even further by severely restricting possible search extensions in the subtrees of non-futile captures and checking moves at otherwise futile frontier nodes.

Selective Futility Condition at Frontier Nodes

$$mat\_balance(node) + mat\_gain(move) + futil\_margin \leq alpha(node) \quad (6)$$
$$futil\_margin \in \{2 * pawn\_val \ldots 4 * pawn\_val\} < max\_posn\_score$$

4 Futility Pruning at Pre-Frontier Nodes

Our new pruning technique extends on the scheme of selective futility pruning at frontier nodes and combines it with an idea from razoring. Therefore, we call it "extended futility pruning". Extended futility pruning cuts complete branches of non-checking moves at pre-frontier nodes (depth = 2) according to solely static criteria at the respective nodes as described in Section 4.1. This is of course theoretically unsound and also slightly riskier than normal futility pruning. However, extensive experiments with our master-strength chess program DarkThought as discussed in Section 4.2 show that extended futility pruning works markedly well in practice.

4.1 Theory

The practical success of futility pruning at frontier nodes springs from the chess-related meaning of the selective futility condition. If the condition $mat\_balance(node) + futil\_margin \leq alpha(node)$ holds for a node the side-to-move suffers from a material disadvantage of at least $futil\_margin \geq 2 * pawn\_val$ with respect to the lower search bound. Because this bound must be exceeded in order to avoid a fail-low alpha cutoff, the side-to-move clearly features a bad position at the given node (relative to the bound).

In bad positions it is generally quite unusual in chess that quiet moves or non-checking captures with small material gains substantially improve the situation. The probability of such an improvement depends on the actual disadvantage $alpha(node) - mat\_balance(node)$ and the remaining search depth at the node. The greater the disadvantage and the smaller the remaining depth are, the more unlikely will searches of these moves reveal any improvements over checking moves and captures with large material gains. In the case of selective futility pruning at frontier nodes the disadvantage amounts to at least 2 Pawns with a remaining depth$\,=1$. The practical results of selective futility pruning at frontier nodes show that 1-ply searches of presumably futile moves almost never discover better compensation for disadvantages of ≥ 2 Pawns than 1-ply searches of truly non-futile moves.

Because extended futility pruning applies at pre-frontier nodes with a remaining depth$\,=2$, the margin of material disadvantage triggering extended cutoffs of presumably futile moves must be considerably higher than the normal futility margin. Otherwise, the selective risks of extended futility pruning would easily exceed its gains. Values ranging from Rook to Rook+Pawn for the extended futility margin make good sense from a chess point of view and they prove to work very well in practice (see Section 4.2). For additional safety, extended futility pruning reuses an idea from razoring and does not perform any cutoffs at pre-frontier nodes if the immediately predecessing move caused an extension of search depth.

Extended Futility Condition at Pre-Frontier Nodes

$$mat\_balance(node) + mat\_gain(move) + extd\_futil\_margin \leq alpha(node) \quad (7)$$

$$extd\_futil\_margin \in \{rook\_val \ldots rook\_val + pawn\_val\}$$

4.2 Practice

Our chess program DarkThought relied on extended futility pruning with a cutoff margin of 6 Pawns during the 15th ICCA World Microcomputer Chess Championship (WMCC) in October 1997. DarkThought finished the championship on a shared 4th place among 34 strong participants. Hence, the new pruning scheme already inaugurated its practical value with a successful real-life performance under tough tournament conditions.

In our quantitative experiments with all 2180 postions from the test suites "Encyclopedia of Chess Middlegames" (ECM, 879 positions), "Win at Chess" (WAC, 300 positions), and "1001 Winning Chess Sacrifices" (WCS, 1001 positions) extended futility pruning performed equally well. The version of DarkThought used for the experiments was mostly identical with the one described in our article "How DarkThought plays chess" [12].

DarkThought is a fast yet sophisticated alpha-beta searcher using PVS/NegaScout [7, 17] with state-of-the-art enhancements like normal futility pruning, internal iterative deepening [2, 21], dynamic move ordering (history+killer heuristic) [1, 9, 18, 20, 23], recursive null-move pruning [4, 8, 10], selective extensions [2, 3], interior-node recognizers [11], and an extended transposition table [15, 23]. On average, all enhancements taken together reduce the effective branching factor of DarkThought to 2–3 and its search-tree size to roughly 55% of that of the according minimal tree [13]. Therefore, the program already generated very slim search trees without the new pruning scheme[3] while routinely reaching search depths of 11–13 plies in normal middle-game positions at tournament time-controls.

Even at fixed search depths, extended futility pruning exhibits hardly any loss of tactical strength when $extd\_futil\_margin = 6 * pawn\_val$. Yet it shrinks the search trees by additional 10%–20% in comparison with our state-of-the-art implementation of normal futility pruning. Table 1 gives the according performance data as obtained by searching all 2180 positions of the ECM, WAC, and WCS test suites to fixed depths of 8, 10, and 12 plies with the 1997 WMCC version of DarkThought (see Appendix A for a detailed description of the exact experimental setup). The data shows that extended futility pruning scales nicely with higher search depths. The overall relative savings as counted in number of nodes visited increase from 12% at a fixed search depth of 8 plies to 21% at a fixed search depth of 12 plies. Simultaneously, the already extremely low quota of less overall

[3] Private and public discussions with authors of other chess programs confirm this notion. During the last three WMCCs DarkThought seemed to always search among the smallest trees of all participating programs.

Test Suite	Normal #Nodes	Normal #Solved	Extd. Δ Nodes	Extd. Δ Solved	(No.)
ECM–08	1,232,004,798	552 / 879	−11.86%	−1.81%	(–10)
ECM–10	8,823,781,692	642 / 879	−13.41%	−0.94%	(–6)
ECM–12	83,443,531,950	704 / 879	−16.70%	−1.14%	(–8)
WAC–08	146,094,041	285 / 300	−14.10%	0.00%	(0)
WAC–10	946,867,509	296 / 300	−20.38%	−0.68%	(–2)
WAC–12	8,998,551,515	296 / 300	−29.89%	0.00%	(0)
WCS–08	750,804,397	841 / 1001	−11.86%	−0.48%	(–4)
WCS–10	5,398,696,585	866 / 1001	−17.22%	−0.35%	(–3)
WCS–12	52,801,555,626	874 / 1001	−26.58%	+0.23%	(+2)
Sum–08	2,128,903,236	1678 / 2180	**−12.01%**	**−0.83%**	**(–14)**
Sum–10	15,169,345,786	1804 / 2180	**−15.20%**	**−0.61%**	**(–11)**
Sum–12	145,243,639,091	1874 / 2180	**−21.11%**	**−0.32%**	**(–6)**

Table 1: Performance of Normal vs. Extended Futility Pruning.

solutions further decreases from 0.8% at a fixed search depth of 8 plies to 0.3% at a fixed search depth of 12 plies.

The preserved tactical strength clearly distinguishes extended futility pruning from pure razoring. As for our own experience, pure razoring greatly suffers from tactical misconceptions that often tend to delay critical findings for some search plies which in turn causes gross blunders with inacceptable frequency. Moreover, our new scheme may even be easier to implement than pure razoring.

5 Limited Razoring at Pre-Pre-Frontier Nodes

Pushing the idea of futility pruning one step further to pre-pre-frontier nodes ($\text{depth} = 3$), we combine it with the depth-reduction mechanism of deep razoring. Section 5.1 describes the rules of the ensuing pruning technique and explains in detail how to integrate both extended futility pruning and limited razoring with an existing alpha-beta search. Section 5.2 presents our extensive experiments which show that limited razoring works very well in practice, especially in combination with extended futility pruning.

5.1 Theory

Because of the high remaining search depth, only a giant margin of material disadvantage ranging from at least Queen to Queen+Pawn allows for the sufficiently riskless triggering of the razor. We achieve additional safety by neither razoring nodes with ≤ 3 opposing pieces nor if the immediately predecessing move caused an extension of search depth. Otherwise, our conservative version of razoring reduces the remaining search depth of pre-pre-frontier nodes by 1 ply if they fulfill the razor condition. Then it applies extended futility pruning to the freshly razored nodes. Thence, the overall effect of limited razoring at pre-pre-frontier nodes is to cut off quiet moves and non-checking captures with insufficient material gains and a remaining depth of 3 plies while reducing the search depth of all other moves by 1 ply if the side-to-move suffers from a severe disadvantage.

Limited Razor Condition at Pre-Pre-Frontier Nodes

$$mat\_balance(node) + razor\_margin \leq alpha(node) \tag{8}$$
$$razor\_margin \in \{queen\_val \ \ldots \ queen\_val + pawn\_val\}$$

The integration of both extended futility pruning and limited razoring with an existing alpha-beta searcher proves to be quite simple. Figure 1 provides the skeleton of a correspondingly selective search function that outlines the necessary additions. Because of its verbose ANSI-C style and the numerous embedded comments, the search skeleton is quite self-explanatory. It assumes all not explicitly declared identifiers to be suitably defined and initialized. Furthermore, the following two general observations are noteworthy.

- The result variable of the search function requires proper initialization in case of a selective search because all legal moves might get pruned.
- The score of a search whose mode of operation depends on the values of alpha or beta at the current node (e.g. extended futility pruning) must not be stored as an exact result in the transposition table. Storing it as a bound is often possible but requires careful handling, too.

```
int search(int alpha, int beta, int depth, int move, node parent) {
    node current;
    int extend, fmax, fprune, fscore, score, selective;

    /* execute the opponent's move and initialize some local variables */
    make_move(parent, move, &current);
    fprune = selective = 0;
    score = -INFINITY;

    /* determine if and how to extend the search at the current node */
    extend = extensions(depth, move, current);
    depth += extend;

    /* decide about limited razoring at pre-pre-frontier nodes */
    fscore = (mat_balance(current) + razor_margin);
    if (!extend && (depth==PRE_PRE_FRONTIER) && (fscore <= alpha)
          && (opposing_pieces(current) > 3)) depth = PRE_FRONTIER;

    /* decide about extended futility pruning at pre-frontier nodes */
    fscore = (mat_balance(current) + extd_futil_margin);
    if (!extend && (depth == PRE_FRONTIER) && (fscore <= alpha)) {
        fprune = selective = 1;
        score = fmax = fscore;
    }

    /* decide about selective futility pruning at frontier nodes */
    fscore = (mat_balance(current) + futil_margin);
    if (!check(move) && (depth == FRONTIER) && (fscore <= alpha)) {
        fprune = selective = 1;
        score = fmax = fscore;
    }

    /*  "selective == 1" and "-INFINITY < score ≤ alpha" hold if
     *  the search will be selective at the current node.
     *
     *  continue like normal but prune all futile moves if "fprune == 1"
     */
    ...
    for (move=first(current), (move!=0), move=next(current,move))
        if (!fprune || check(move)          /* recursive search call */
                || (fmax + mat_gain(move) > alpha)) {...}
    ...
}
```

Figure 1: Search with Extended Futility Pruning and Limited Razoring.

5.2 Practice

According to our quantitative experiments with $razor\_margin = queen\_val$, limited razoring at pre-pre-frontier nodes exhibits no measurable loss in tactical strength as compared with extended futility pruning alone. Yet it shrinks the search trees by additional 5%–15% on average at fixed search depths > 10 plies. Table 2 lists the according performance data as obtained by searching all 2180 positions of the ECM, WAC, and WCS test suites to fixed depths of 8, 10, and 12 plies with the 1997 WMCC version of DarkThought (see Appendix A for a detailed description of the exact experimental setup). The data shows that limited razoring hardly influences searches of fixed depths ≤ 10 plies but then really takes off at fixed depth of 12 plies. We observe even greater savings (10%–15%) in other test positions with deep solutions which need search depths ≥ 13 plies to be found.

Test Suite	**Extd. #Nodes**	**Extd. #Solved**	**+Razor Δ Nodes**	**+Razor Δ Solved**	(No.)
ECM–08	1,085,889,029	542 / 879	−0.91%	+0.92%	(+5)
ECM–10	7,640,512,567	636 / 879	−1.50%	0.00%	(0)
ECM–12	69,508,462,114	696 / 879	−9.64%	−0.43%	(–1)
WAC–08	125,494,781	285 / 300	−0.21%	0.00%	(0)
WAC–10	753,895,911	294 / 300	−1.93%	0.00%	(0)
WAC–12	6,308,884,467	296 / 300	−3.87%	0.00%	(0)
WCS–08	661,758,996	837 / 1001	−2.41%	−0.24%	(–2)
WCS–10	4,469,041,033	863 / 1001	−2.54%	+0.23%	(+2)
WCS–12	38,766,902,141	876 / 1001	−6.68%	+0.11%	(+1)
Sum–08	1,873,142,806	1664 / 2180	**−1.39%**	**+0.18%**	**(+3)**
Sum–10	12,863,449,511	1793 / 2180	**−1.89%**	**+0.11%**	**(+2)**
Sum–12	114,584,248,722	1868 / 2180	**−8.32%**	**0.00%**	**(0)**

Table 2: Performance of Extended Futility Pruning vs. Added Razoring.

Last but not least, we like to highlight the synergetic performance of both extended futility pruning and limited razoring combined. Table 3 presents the according data by comparing the normal version of DarkThought (see Section 4.2) with one that is identical except for the addition of extended futility pruning and limited razoring.

Test Suite	Normal #Nodes	Normal #Solved	Ex.+Rz. Δ Nodes	Ex.+Rz. Δ Solved	(No.)
ECM–08	1,232,004,798	552 / 879	−12.66%	−0.91%	(–5)
ECM–10	8,823,781,692	642 / 879	−14.71%	−0.94%	(–6)
ECM–12	83,443,531,950	704 / 879	−24.73%	−1.28%	(–9)
WAC–08	146,094,041	285 / 300	−14.28%	0.00%	(0)
WAC–10	946,867,509	296 / 300	−21.92%	−0.68%	(–2)
WAC–12	8,998,551,515	296 / 300	−32.60%	0.00%	(0)
WCS–08	750,804,397	841 / 1001	−13.98%	−0.71%	(–6)
WCS–10	5,398,696,585	866 / 1001	−19.32%	−0.12%	(–1)
WCS–12	52,801,555,626	874 / 1001	−31.49%	+0.34%	(+3)
Sum–08	2,128,903,236	1678 / 2180	**−13.24%**	**−0.66%**	**(–11)**
Sum–10	15,169,345,786	1804 / 2180	**−16.81%**	**−0.50%**	**(–9)**
Sum–12	145,243,639,091	1874 / 2180	**−27.68%**	**−0.32%**	**(–6)**

Table 3: Performance of Normal vs. Extended Futility Pruning + Razoring.

6 Conclusion

Extended futility pruning and limited razoring apply the futility idea to pre- and pre-pre-frontier nodes. Slight modifications and additions with respect to normal futility pruning provide the proper safety precautions that greatly reduce the selective risks of the new schemes. Extensive experiments with 2180 positions from the well-known test suites ECM, WAC, and WCS provide quantitative empirical evidence for the benefits of each scheme alone and of both combined. While scaling well with search depth and hardly compromising any tactical strength, extended futility pruning and limited razoring substantially reduce the search trees of our master-strength chess program DarkThought by 10%–30% on average at fixed depths of 8–12 plies. Moreover, extended futility pruning demonstrated its practical viability with a successful real-life performance under tough tournament conditions during the 15th WMCC in October 1997. These convincing results show that statically selective forward-pruning techniques do not only promise some success but really deliver on making alpha-beta search more scalable at high search depths.

A Experimental Setup

- Test suites "Encyclopedia of Chess Middlegames" (ECM, 879 positions), "Win at Chess" (WAC, 300 positions), and "1001 Winning Chess Sacrifices" (WCS, 1001 positions) as available on the Internet,

- DarkThought as of October 31, 1997 with 8M transposition-table entries (4M per side), 1M King hash-table entries (512K per side), and 512K Pawn hash-table entries,
- Digital Unix 4.0b program development tools and operating system,
- 500MHz AlphaPC164 workstation (500MHz DEC Alpha-21164a CPU, 8KB/8KB on-chip I/D L1 caches, 96KB unified on-chip L2 cache, 1MB unified off-chip L3 cache, DEC PC164 mainboard, 8x32MB PS/2 SIMMs = 256MB RAM).[4]

References

[1] Akl, S. G. and Newborn, M. M. (1977). The principal continuation and the killer heuristic. *ACM '77 National Conference*, Proceedings, pp. 466–473, ACM Press.

[2] Anantharaman, T. S. (1991). Extension heuristics. *ICCA Journal*, Vol. 14, No. 2, pp. 47–65.

[3] Beal, D. F. and Smith, M. C. (1995). Quantification of search extension benefits. *ICCA Journal*, Vol. 18, No. 4, pp. 205–218.

[4] Beal, D. F. (1989). Experiments with the null move. *Advances in Computer Chess 5*, D. F. Beal (ed.), pp. 65–79, Elsevier Science, ISBN 0-444-87159-4.

[5] Birmingham, J. A. and Kent, P. (1977). Tree-searching and tree-pruning techniques. *Advances in Computer Chess 1*, M. R. B. Clarke (ed.), pp. 89–107, Edinburgh University Press [reprinted in *Computer Chess Compendium*, D. N. L. Levy (ed.), pp. 123–128, Springer, 1989, ISBN 0-387-91331-9].

[6] Buro, M. (1995). ProbCut: An effective selective extension of the α-β algorithm. *ICCA Journal*, Vol. 18, No. 2, pp. 71–76.

[7] Campbell, M. S. and Marsland, T. A. (1983). A comparison of minimax tree search algorithms. *Artificial Intelligence*, Vol. 20, No. 4, pp. 347–367.

[8] Donninger, C. (1993). Null move and deep search: Selective search heuristics for obtuse chess programs. *ICCA Journal*, Vol. 16, No. 3, pp. 137–143.

[9] Gillogly, J. J. (1972). The technology chess program. *Artificial Intelligence*, Vol. 3, No. 1–3, pp. 145–163 [reprinted in *Computer Chess Compendium*, D. N. L. Levy (ed.), pp. 67–79, Springer, 1989, ISBN 0-387-91331-9].

[4] Provided by Digital Equipment Corp. under external technology research contract DE-0032-97 entitled "Optimization and fine-tuning of a sophisticated tree-searching application for 64bit Alpha CPUs".

[10] Goetsch, G. and Campbell, M. S. (1990). Experiments with the null-move heuristic. *Computers, Chess, and Cognition*, T. A. Marsland and J. Schaeffer (eds.), pp. 159–168, Springer, ISBN 0-387-97415-6/3-540-97415-6.

[11] Heinz, E. A. (1998). Efficient interior-node recognition. *ICCA Journal*, Vol. 21, No. 3, pp. 156–167.

[12] Heinz, E. A. (1997). How DarkThought plays chess. *ICCA Journal*, Vol. 20, No. 3, pp. 166–176.

[13] Knuth, D. E. and Moore, R. W. (1975). An analysis of alpha-beta pruning. *Artificial Intelligence*, Vol. 6, No. 4, pp. 293–326.

[14] Marsland, T. A. (1992). Computer chess and search. *Encyclopedia of Artificial Intelligence*, S. C. Shapiro (ed.), pp. 224–241, Wiley & Sons, 2nd ed. 1992, ISBN 0-471-50305-3.

[15] Nelson, H. L. (1985). Hash tables in Cray Blitz. *ICCA Journal*, Vol. 8, No. 1, pp. 3–13.

[16] Newborn, M. M. (1975). *Computer Chess*. Academic Press, New York.

[17] Reinefeld, A. (1983). An improvement to the Scout tree-search algorithm. *ICCA Journal*, Vol. 6, No. 4, pp. 4–14.

[18] Schaeffer, J. (1989). The history heuristic and alpha-beta search enhancements in practice. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 11, No. 11, pp. 1203–1212.

[19] Schaeffer, J. (1986). *Experiments in Search and Knowledge*. Ph.D. Thesis, University of Waterloo [reprinted as Technical Report TR 86–12, Department of Computing Science, University of Alberta].

[20] Schaeffer, J. (1983). The history heuristic. *ICCA Journal*, Vol. 6, No. 3, pp. 16–19.

[21] Scott, J. J. (1969). A chess-playing program. *Machine Intelligence 4*, B. Meltzer and D. Michie (eds.), pp. 255–265, Edinburgh University Press, 1973, ISBN 0-852-24062-7.

[22] Slagle, J. R. (1971). *Artificial Intelligence: The Heuristic Programming Approach*. McGraw-Hill, New York.

[23] Slate, D. J. and Atkin, L. R. (1977). Chess 4.5 – The Northwestern University chess program. *Chess Skill in Man and Machine*, P. W. Frey (ed.), pp. 82–118, Springer, 2nd ed. 1983, ISBN 0-387-90790-4/3-540-90790-4.

[24] Ye, C. and Marsland, T. A. (1992). Experiments in forward pruning with limited extensions. *ICCA Journal*, Vol. 15, No. 2, pp. 55–66.