Preprint of an article in: ICCA Journal 22(3), pp. 123–132, Sep. 1999.

Adaptive Null-Move 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

September 24, 1999

Abstract

General wisdom deems strong computer-chess programs to be "brute-force searchers" that explore game trees as exhaustively as possible within the given time limits. We review the results of the latest World Computer-Chess Championships and show how grossly wrong this notion actually is. The typical brute-force searchers lost their dominance of the field around 1990 when the *null move* became popular in micro-computer practice. Today, nearly all world-class chess programs apply various selective forward-pruning schemes with overwhelming success.

To this end, we extend standard null-move pruning by a variable depth reduction and introduce what we call *adaptive null-move pruning*. Quantitative experiments with our chess program DarkThought[1] show that adaptive null-move pruning adds a new member to the collection of successful forward-pruning techniques in computer chess. It preserves the tactical strength of DarkThought while reducing its search effort by 10%–30% on average in comparison with standard null-move pruning at search depths of 8–12 plies. Moreover, adaptive null-move pruning is easy to implement and scales nicely with progressing search depth

1 Introduction

Early research in computer chess mostly tried to mimic the human game-playing approach within computer systems [36]. The resulting chess programs relied on so-called "plausible" move generators. These were very

*Email: <heinze@ira.uka.de>, WWW = <http://wwwipd.ira.uka.de/~heinze/>
[1]WWW = <http://wwwipd.ira.uka.de/Tichy/DarkThought/>

knowledge-intensive and cut off bad-looking ("implausible") moves at all levels of the search tree, thus statically forward-pruning large parts of it. Then, in the mid-1970s, brute-force searchers like TECH [19] and CHESS 4.X [43] took the lead as soon as they reached search depths of 5 plies and more in the middlegame. They routinely punished the tactical weaknesses of plausible move generation and started their own reign.

Except for search extensions which increase the lookahead in important lines of play (e.g. after checks or recaptures), typical brute-force programs of this era featured no selectivity at all in the full-width parts of their searches. The famous special-purpose chess machines BELLE [14, 15], DEEP THOUGHT [30, 31], and HITECH [6, 7, 17] belong to this class. Together with the supercomputer program CRAY BLITZ [32] these brute-force searchers dominated the computer-chess scene until roughly 1990. Because they were the first to compete successfully against human chess masters, the brute-force paradigm received broad publicity and media attention. This is probably the reason for the unfortunate misconception that strong chess programs must be inherently boring brute-force searchers.

Forward-pruning chess programs dethroned the traditional brute-force searchers in the early 1990s shortly after the advent of standard *null-move pruning* [4, 5, 16, 20]. Since then, highly selective microcomputer programs have won all official World Computer-Chess Championships as hosted by the International Computer Chess Association in 1992, 1995, and 1999 [27, 44, 18]. We briefly sketch the achievements of the three champions below.

7th World Computer-Chess Championship (1992). Based on a single 32 MHz ARM processor, the 1992 champion THE CHESSMACHINE SCHRÖDER outclassed the special-purpose chess machine HITECH and the massively parallel supercomputer program ZUGZWANG (then a brute-force searcher on 1024x 16 MHz T800 transputers).

8th World Computer-Chess Championship (1995). Running on a 90 MHz Pentium, the 1995 champion FRITZ beat DEEP THOUGHT II (the direct predecessor of IBM's famous special-purpose chess machine DEEP BLUE) and the massively parallel supercomputer program STARSOCRATES (employing an Intel Paragon with 1824x 66 MHz i860 CPUs). Furthermore, FRITZ outclassed the special-purpose chess machine HITECH as well as the massively parallel supercomputer programs FRENCHESS (using a Cray T3D with 256x 150 MHz Alpha-21064 CPUs) and ZUGZWANG (now relying on a Parsytec GC with 96x 66 MHz PowerPC-601 CPUs).

9th World Computer-Chess Championship (1999). Playing on a 550 MHz Pentium-III PC, the 1999 champion SHREDDER outclassed the massively parallel supercomputer programs CILKCHESS (employing an SGI Origin 2000 with 256x 225 MHz R10000 CPUs), P.CONNERS (using a cluster of 180x 450 MHz Pentium-II PCs), ZUGZWANG (relying on a Cray T3E with 512x 300 MHz Alpha-21164a CPUs), and other programs running on symmetric multiprocessors with 4x 450 MHz Pentium-II CPUs (FERRET, FRITZ, and JUNIOR).

The greatest disadvantage of selective pruning are the tactical weaknesses which it incurs. Only those techniques that feature both low tactical risks and substantial savings of search effort prove to be successful in practice. Fortunately, such well-behaved selective pruning schemes do exist in computer chess. Today, IBM's DEEP BLUE [21, 28, 29] is probably the last proponent of the genuine brute-force paradigm. The developers obviously felt they had enough computing power to spare such that they could refrain from taking any chances.

The work presented in this article aims to augment the collection of successful forward-pruning methods in computer chess. We introduce our new *adaptive null-move pruning* scheme and thoroughly evaluate its performance on tactical test suites. Before concentrating on the fine details of adaptive null-move pruning in Section 4, we elaborate on related work in Section 2 and on standard null-move pruning in Section 3.

2 Related Work

In his excellent overview of computer chess and search which covers developments up to the late 1980s, Marsland [33] mentions three statically selective forward-pruning techniques that gained broad attention in the early days of the field. Today, however, Birmingham and Kent's *razoring* [8], Newborn's GAMMA algorithm [35], and Slagle's *marginal forward pruning* [42] generate only little interest (if any at all) because their aggressive selectiveness incurs too many tactical risks. Nevertheless, the fundamental ideas of these methods still bear fruits. As shown by us for *extended futility pruning* and *limited razoring* in [25], static forward pruning can work very well in practice if you confine it to the lower parts of the full-width search and add some basic safety precautions.

Another long-standing pruning scheme continues to enjoy wide-spread use in chess programs although there does not exist much published material about it. Slate and Atkin [43] introduced a limited form of what we call *normal futility pruning* in the early 1970s. Then, Schaeffer [39] extended the underlying idea in the mid-1980s and we re-explained it in more detail in 1998 [25]. Schaeffer also made some quantitative measurements that illustrated the practical effectiveness of the scheme. Ye and Marsland [45] later confirmed Schaeffer's findings by their own independent experiments with normal futility pruning in Chinese chess.

The first reports about successful experiments with the *null move* [4, 5, 20] initiated a renaissance of selective forward-pruning in computer chess around 1990. Null-move pruning exhibits only minor tactical weaknesses while reducing the overall search effort substantially. Moreover, it scales well with progressing search depth. Null-move pruning bases the cutoff decisions on dynamic rather than static criteria at the respective nodes. To this end, it speculatively replaces normal searches with much cheaper searches of reduced depths (see Section 3). The spectacular successes of null-move pruning in microcomputer practice made it one of the most popular ingredients of modern chess programs.

Björnsson and Marsland [9, 10] proposed *multi-cut alpha-beta pruning* and *variable-bound null-move pruning* as further forward-pruning methods in computer chess. Unfortunately, they only present the results of a few test games as qualitative empirical evidence for the promise of their new techniques. The rather balanced results of these test games do not provide convincing support for the promise in our opinion. In order to quantify the real benefits of their new schemes, Björnsson and Marsland ought to perform much more extensive experiments with them. Last but not least, in computer Othello the dynamic forward-pruning schemes ProbCut [12], Multi-ProbCut [11], and improvements of them excel. But up to now, there are no publications about either ProbCut or Multi-ProbCut in computer chess.

3 Standard Null-Move Pruning

The brute-force paradigm does not allow for any forward pruning in the full-width part of the search. Beyond the full-width horizon during the so-called "quiescence search", however, even genuine brute-force programs perform static forward pruning without hesitation. Otherwise, the number of quiescence nodes explodes so quickly that the whole search gets stuck at shallow full-width depths. As soon as traditional brute-force searches with extensions approach full-width depths of 9–10 plies in the middlegame, the same phenomenon re-emerges at the full-width level. This is exactly where forward pruning steps in again. It increases the scalability of the search at all levels if we abandon the pure brute-force paradigm.

Null-move pruning is a dynamic forward-pruning scheme that generates selective cutoffs at nominal full-width nodes where the side-to-move is not in check [5, 16, 20]. Even the recursive application of null-move pruning along a single search path exhibits only minor tactical weaknesses while reducing the overall search effort substantially. Furthermore, it scales nicely with search depth as the savings increase and the tactical shortcomings diminish at higher depths. The significant savings of null-move pruning spring mostly from the depth reduction of the null-move searches. Given the additional effort which unsuccessful null-move searches waste, the performance gains of null-move pruning in practice are all the more remarkable. Regarding the depth reduction of null-move searches, the accepted standard recommended the usage of a constant depth-reduction factor $R = 2$. Independent studies by several researchers confirmed that recursive null-move pruning with $R = 2$ behaved markedly better than both the too conservative $R = 1$ and the too aggressive $R = 3$.

We verified the standard recommendation for our own master-strength chess program DarkThought [26] which successfully participated in all official ICCA world championships since 1995. DarkThought is a fast yet sophisticated alpha-beta searcher using PVS/NegaScout [13, 37] with state-of-the-art enhancements such as normal and extended futility pruning [25, 39, 43], internal iterative deepening [2, 41], dynamic move ordering (history+killer heuristic) [1, 19, 38, 40, 43], selective extensions [2, 3], interior-node recognizers [24], knowledgeable endgame databases [23], and an extended transposition table [34, 43]. The program routinely reaches search depths of 11–13 plies in the middlegame at tournament time-controls. DarkThought already generated very slim search trees while using standard recursive null-move pruning with $R = 2$. During the World Microcomputer-Chess Championships in 1995, 1996, and 1997, DarkThought always searched among the smallest trees of all participating programs as confirmed by private and public discussions with the authors.

Table 1 lists the results of our quantitative null-move experiments with all 2180 positions from the well-known tactical test suites "Encyclopedia of Chess Middlegames" (ECM, 879 positions), "Win at Chess" (WAC, 300 positions), and "1001 Winning Chess Sacrifices" (WCS, 1001 positions) as publicly available on the Internet. Three different versions of DarkThought, employing recursive null-move pruning with $R = 1$, $R = 2$, and $R = 3$ respectively, searched the 2180 positions to fixed depths of 8, 10, and 12 plies each (see Appendix A for a detailed description of the exact experimental setup). The relative numbers of nodes searched and positions solved show that the standard recommendation $R = 2$ clearly outperforms the other constant depth-reduction factors for DarkThought.

Test Suite	R = 1 Δ Nodes	R = 1 Δ Sol	R = 2 #Nodes	R = 2 #Solved	R = 3 Δ Nodes	R = 3 Δ Sol
ECM–08	+96.29%	+8	1,232,004,798	552 / 879	−23.23%	−21
ECM–10	+198.76%	+3	8,823,781,692	642 / 879	−33.55%	−17
ECM–12	+304.35%	+1	83,443,531,950	704 / 879	−44.50%	−11
WAC–08	+101.63%	0	146,094,041	285 / 300	−27.88%	−4
WAC–10	+205.97%	0	946,867,509	296 / 300	−36.06%	−2
WAC–12	+317.48%	0	8,998,551,515	296 / 300	−48.04%	−2
WCS–08	+100.02%	0	750,804,397	841 / 1001	−27.63%	−10
WCS–10	+201.13%	+5	5,398,696,585	866 / 1001	−36.34%	−11
WCS–12	+295.46%	+2	52,801,555,626	874 / 1001	−42.00%	−8
Sum–08	**+97.97%**	**+8**	2,128,903,236	1678 / 2180	**−25.10%**	**−35**
Sum–10	**+200.05%**	**+8**	15,169,345,786	1804 / 2180	**−34.70%**	**−30**
Sum–12	**+301.93%**	**+3**	145,243,639,091	1874 / 2180	**−43.74%**	**−21**

Table 1: Performance of Null-Move Depth Reduction $R = 1,\ 2,\ 3$.

Despite the additional savings in search effort, $R = 3$ does not represent a viable alternative because of its severe tactical deficiencies. The numbers of solutions missed by $R = 3$ in comparison with $R = 2$ consistently exceed the standard errors of the overall $R = 2$ results by far. We calculate the *absolute* standard error $SE = n * \sqrt{p * (1-p)/n}$ for a score of k solutions in n test positions by multiplying its relative counterpart with n where $p = k/n$. For the overall $R = 2$ results as listed in rows "Sum–08", "Sum–10", and "Sum–12" of Table 1, the SE formula yields $SE_8 = 20$, $SE_{10} = 18$, and $SE_{12} = 16$ at the tested search depths of 8, 10, and 12 plies. This makes it highly unlikely that the observed decline in tactical ability of the $R = 3$ version was just random noise.

4 Recursively Adaptive Null-Move Pruning

The significantly lower search effort of recursive null-move pruning with $R = 3$ is a tempting incentive to look for possible ways in order to exploit it without compromising the tactical strength of $R = 2$. In this respect, variable rather than constant depth reduction for null moves quickly springs to mind as an intuitive and straightforward idea. Other researchers already speculated about such schemes in the past [16, 20]. They suggested to increase the depth reduction of null-move searches with decreasing distance from the full-width horizon. In particular, Donninger [16] hinted at using $R = 2$ in the upper parts of the full-width search and $R = 3$ in the lower parts thereof. Unfortunately, we do not know of any publications that report about the proposed method of variable null-move depth reduction and experiments with it in more detail. Hence, we performed the necessary tests and measurements ourselves. To our disappointment, all trials with values of R ranging from 2–4 and various different thresholds of remaining search depth at which to vary R (ranging from 2–8) failed miserably. None of those parameter combinations improved the tactical abilities of DarkThought beyond the unacceptable level of the plain $R = 3$ constant (see Table 1).

Since then we strongly doubt the practical usefulness of variable null-move depth reduction with increasing values of R towards the full-width horizon. Instead, we started to consider the reverse strategy of variable depth reduction with decreasing values of R in the lower parts of the full-width search. This obviously lowers rather than raises the degree of selectivity with increasing distance from the root position which not only contradicts general wisdom but also our prior experiences with other forward-pruning techniques. Therefore, we did not give the reversed scheme much chances at first and even judged it as counter-productive. To our utter surprise, however, already the very first test implementation of our new *adaptive* method for null-move depth reduction turned out to be an overwhelming success. Meanwhile, our proven implementation of *adaptive null-move pruning* in DarkThought easily combines the merits of both $R = 2$ (tactical safety) and $R = 3$ (reduced search effort) as detailed in Section 4.2.

4.1 Theory

The core idea of our adaptive scheme is to apply a reduction factor of $R = 3$ at full-width nodes with a remaining search depth of more than 6 plies and a reduction factor of $R = 2$ otherwise. We denote this specific method of adaptive depth reduction by $R = 3 \underset{6}{\sim} 2$. Qualitative and quantitative empirical measurements showed that a remaining depth of 6 plies was clearly the best threshold for switching from $R = 3$ to $R = 2$ in DarkThought. While lower thresholds did not provide enough tactical safety, higher thresholds did not gain any and thus only wasted precious search effort. In addition to the remaining search depth, the latest version of DarkThought also takes the material situation at the respective node into account when deciding about the actual depth reduction $R_{adpt} = f(depth, material)$. We augmented the scope of our adaptive method by the number of pieces on the chess board because we discovered that the bare core idea applied null moves too aggressively in endgames with reduced material. Our revised and fine-tuned formula for the calculation of R_{adpt} now restricts the application of $R = 3$ to nodes with a remaining search depth of more than 8 plies if the maximal number of pieces per side drops below three. [Remark: For the sake of convenience, we define $R = 3\tilde{_6}2$ to equal the fine-tuned formula from here on.]

Fine-Tuned Adaptive Null-Move Depth Reduction

$$R_{adpt} = \begin{cases} 2 & \text{if } (depth \leq 6) \text{ or } ((depth \leq 8)\ \&\ (max\_pieces\_per\_side < 3)) \\ 3 & \text{if } (depth > 8) \text{ or } ((depth > 6)\ \&\ (max\_pieces\_per\_side \geq 3)) \end{cases}$$

Despite our initial misjudgement, we quickly understood why the "reversed" adaptive depth reduction works so well. Our research suggests that there is an important distinction between dynamic and static forward-pruning schemes regarding their preferred degrees of selectivity in different parts of the full-width search. While static forward pruning generally thrives on low selectivity in the upper parts and increased selectivity in the lower ones, dynamic forward pruning excels with the exact opposite. Hence, dynamic forward pruning prefers high remaining search depths and static forward pruning low ones for the application of aggressive selectivity. This allows both styles of forward pruning to sustain tactical safety. Our adaptive depth reduction simply exploits the dynamic nature of null-move pruning in this very respect.

Building on the explanations above, we illustrate the crucial algorithmic aspects of adaptive null-move pruning in Figure 1. There, we present the skeleton of a selective search function that performs adaptive null-move searches with a minimal window centered around the value of beta at the respective node. It returns the actual score in fail-soft manner if the null-move search fails high. We intentionally omitted the details of how to store the final search result in the transposition tables in order to keep the code fragment focused. Still, the skeleton tries to avoid superfluous null-move searches that do not really promise to cut off as decided by the function `try_null()`. Because null moves do not make much sense if the side-to-move is in check or the opponent executed a null move directly before, we further guard the null-move searches by the predicates `!check(move)` and `null_okay()`.

```
/* macro definition of the adaptive null-move depth reduction */
#define R_adpt(node, depth) ( \
    2 + ((depth) > (6 + ((max_pieces_per_side(node)<3) ? 2 : 0))))

/* recursive PVS/NegaScout search function for nominal full-width nodes */
int search(int alpha, int beta, int move, node parent, int depth) {

    node current; int null_score, tt_hit; tt_entry tt_ref;

    /* execute the opponent's move */
    make_move(parent, move, &current);

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

    /* probe the transposition tables at the current node */
    tt_hit = probe_transposition_tables(current, depth, &tt_ref);
    if (tt_hit) { ... } else { ... }
    ...

    /* try the adaptive null-move search with a minimal window around */
    /* "beta" only if it is allowed, desired, and really promises to cut off */
    if (!check(move) && null_okay(current, move)
          && try_null(alpha, beta, current, depth, move, tt_ref)) {
        ...
        null_score = -search(-beta, -beta + 1, null_move, current,
                             depth - R_adpt(current, depth) - 1);
        /* test for a potential fail-high null-move cutoff */
        if (null_score >= beta) return null_score;
        ...
    }

    /* now continue as usual at the current node (e.g. recursive PVS part) */
    ...
}
```

Figure 1: Selective Search with Adaptive Null-Move Pruning.

4.2 Practice

DarkThought employed adaptive null-move pruning during the 9th World Computer-Chess Championship which took place in Paderborn (Germany), June 1999. It finished the tournament on shared 5th place as 6th of 30 participants and earned the rank of vice champion in the microcomputer category. Hence, adaptive null-move pruning already demonstrated its practical value with a successful real-life performance.

Moreover, this championship version of DarkThought played hundreds of self-play test games and even more test games versus strong commercial chess programs at tournament time-controls (including the latest versions of numerous world champions of the 1990s).[2] The self-play games pitted DarkThought with adaptive null-move pruning, extended futility pruning, and limited razoring against the normal version with $R = 2$ null-move pruning but neither extended futility pruning nor limited razoring. The detailed results of these self-play matches are published elsewhere [22]: in 100 games at a time control of 60 moves in 90 minutes, for instance, the aggressively pruning version of DarkThought scored 67.5% against the normal $R = 2$ one. The self-play games and the test games versus strong commercial chess programs taken together strongly support the practical usefulness of adaptive null-move pruning in serious game play.

We quantified the savings in search effort of adaptive null-move pruning by letting DarkThought with recursive application of null moves and our fine-tuned adaptive $R = 3 \overset{\sim}{{}_6} 2$ search all 2180 positions of ECM, WAC, and WCS to nominally fixed depths of 8, 10, and 12 plies respectively (see Appendix A for a detailed description of the exact experimental setup). Table 2 compares the results of these test runs with the according numbers for the standard $R = 2$ version of DarkThought which we already listed in Table 1. The data shows that even at fixed search depths adaptive null-move pruning exhibits the same tactical strength as standard null-move pruning with $R = 2$. Yet the adaptive scheme reduces the search effort by 10%–30% on average at search depths of 8–12 plies. Moreover, the relative overall savings as counted in number of nodes visited scale nicely with progressing search depth. They increase linearly from 8.5% at a fixed search depth of 8 plies to 30.5% at a fixed search depth of 12 plies.

Our long-standing experience with the application of adaptive null-move pruning in DarkThought confirms its tactical safety. Up to now we do not know of any position where $R = 3 \overset{\sim}{{}_6} 2$ delays the discovery of the key move by the search for more than a single ply as compared with $R = 2$.

[2]Please find the complete move lists of the latter on our WWW pages.

Test Suite	$\mathbf{R = 2}$ #Nodes	$\mathbf{R = 2}$ #Solved	$\mathbf{R = 3_6^{\sim}2}$ Δ Nodes	$\mathbf{R = 3_6^{\sim}2}$ Δ Solved	
ECM–08	1,232,004,798	552 / 879	−7.31%	0	0.00%
ECM–10	8,823,781,692	642 / 879	−18.14%	+1	+0.16%
ECM–12	83,443,531,950	704 / 879	−33.79%	+1	+0.14%
WAC–08	146,094,041	285 / 300	−11.78%	0	0.00%
WAC–10	946,867,509	296 / 300	−23.65%	0	0.00%
WAC–12	8,998,551,515	296 / 300	−38.13%	0	0.00%
WCS–08	750,804,397	841 / 1001	−10.89%	+2	+0.24%
WCS–10	5,398,696,585	866 / 1001	−19.93%	−1	−0.12%
WCS–12	52,801,555,626	874 / 1001	−27.06%	+1	+0.11%
Sum–08	2,128,903,236	1678 / 2180	**−8.88%**	**+2**	**+0.12%**
Sum–10	15,169,345,786	1804 / 2180	**−19.12%**	**0**	**0.00%**
Sum–12	145,243,639,091	1874 / 2180	**−31.61%**	**+2**	**+0.11%**

Table 2: Performance of Null-Move Depth Reduction $R = 2$ vs. $R = 3_6^{\sim}2$.

Fortunately, such delays are extremely rare in practice. The solution rates of adaptive null-move pruning from Table 2 suggest that the exceptional delays are at least neutralized in general by equally exceptional cases of acceleration where $R = 3_6^{\sim}2$ makes the search lock onto key moves earlier than $R = 2$. Surprising as it may seem at first glance, this "acceleration phenomenon" is easy to verify because it actually happened with all selective forward-pruning schemes which we ever tried.

Finally, we elaborate on another interesting observation concerning the scalability of adaptive null-move pruning. Table 3 contrasts the test-suite results of DarkThought using our fine-tuned adaptive $R = 3_6^{\sim}2$ with those of DarkThought employing a constant $R = 3$. The data reveals that the relative differences in search effort as counted in number of nodes visisted remain almost constant at 17%–19% for these two program versions regardless of search depth. Consequently, adaptive null-move pruning scales as well as $R = 3$ null-move pruning at fixed search depths of 8–12 plies in DarkThought. This provides further empirical evidence for our notion that adaptive null-move pruning combines the best of both $R = 2$ and $R = 3$ null-move pruning. In our opinion, the difference in search effort between $R = 3_6^{\sim}2$and $R = 3$ is fairly low given the large number of solutions which $R = 3$ missed. While retaining the same level of tactical strength as $R = 2$, the savings in search effort of the adaptive scheme do not only increase and scale like those of $R = 3$ with additional search depth but they also come close to them.

Test Suite	$\mathbf{R = 3 \tilde{_6} 2}$ **#Nodes**	$\mathbf{R = 3 \tilde{_6} 2}$ **#Solved**	$\mathbf{R = 3}$ **Δ Nodes**	$\mathbf{R = 3}$ **Δ Solved**	
ECM–08	1,141,945,247	552 / 879	−17.18%	−21	−3.80%
ECM–10	7,223,147,693	643 / 879	−18.83%	−18	−2.80%
ECM–12	55,247,962,504	705 / 879	−16.18%	−12	−1.70%
WAC–08	128,884,163	285 / 300	−18.25%	−4	−1.40%
WAC–10	722,933,343	296 / 300	−16.25%	−2	−0.68%
WAC–12	5,567,403,822	296 / 300	−16.01%	−2	−0.68%
WCS–08	669,041,798	843 / 1001	−18.78%	−12	−1.42%
WCS–10	4,322,736,356	865 / 1001	−20.50%	−10	−1.16%
WCS–12	38,513,454,674	875 / 1001	−20.48%	−9	−1.03%
Sum–08	1,939,871,208	1680 / 2180	**−17.80%**	**−37**	**−2.20%**
Sum–10	12,268,817,392	1804 / 2180	**−19.26%**	**−30**	**−1.66%**
Sum–12	99,328,821,000	1876 / 2180	**−17.73%**	**−23**	**−1.23%**

Table 3: Performance of Null-Move Depth Reduction $R = 3 \tilde{_6} 2$ vs. $R = 3$.

5 Conclusion

Our brief historical overview of search paradigms in computer chess argued that the pure brute-force searchers lost their traditional dominance of the field around 1990. Since then selective forward-pruning techniques have again taken the lead, guided by the spectacular achievements of the null move in microcomputer practice. Today, nearly all world-class chess programs apply various clever forward-pruning schemes with good success.

This article adds adaptive null-move pruning to the collection of successful forward-pruning methods in computer chess. Extensive experiments show that adaptive null-move pruning preserves the tactical strength of our sophisticated and tournament-proven chess program DarkThought while reducing its search effort by 10%–30% on average in comparison with standard $R = 2$ null-move pruning at search depths of 8–12 plies. Furthermore, adaptive null-move pruning and its savings in search effort scale equally well with search depth as those of the extremely risky $R = 3$ scheme. Overall, our fine-tuned implementation of adaptive null-move pruning in DarkThought combines the merits of both standard $R = 2$ and aggressive $R = 3$ null-move pruning in a convincing way. In view of its tactical safety and the neatly reduced search effort, we deem it quite remarkable that adaptive null-move pruning is very easy to implement.

During the course of our research on adaptive null-move pruning we also discovered an important distinction between dynamic and static forward-pruning schemes regarding their preferred degrees of selectivity in different parts of the full-width search. While static forward pruning generally thrives on low selectivity in the upper parts and increased selectivity in the lower ones, dynamic forward pruning excels with the exact opposite. Hence, dynamic forward pruning rather prefers high remaining search depths for the application of aggressive selectivity. Their specific preferences allow both styles of forward pruning to sustain the desired degrees of tactical safety.

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 March 31, 1998 with 8M transposition-table entries (4M per side), 1M King hash-table entries (512K per side), and 512K Pawn hash-table entries,
- Digital Unix 4.0d program development tools and operating system,
- 600MHz AlphaLX164 workstation (600MHz DEC Alpha-21164a CPU, 8KB/8KB on-chip I/D L1 caches, 96KB unified on-chip L2 cache, 2MB unified off-chip L3 cache, DEC LX164 mainboard, 2x128MB SDRAM DIMMs = 256MB RAM).

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. (1990). A generalised quiescence search. *Artificial Intelligence*, Vol. 43, No. 1, pp. 85–98.

[5] 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.

[6] Berliner, H. J. and Ebeling, C. (1990). HITECH. *Computers, Chess, and Cognition*, T. A. Marsland and J. Schaeffer (eds.), pp. 79–109, Springer, ISBN 0-387-97415-6/3-540-97415-6.

[7] Berliner, H. J. (1987). Some innovations introduced by HITECH. *ICCA Journal*, Vol. 10, No. 3, pp. 111–117.

[8] 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, ISBN 0-852-24292-1 [reprinted in *Computer Chess Compendium*, D. N. L. Levy (ed.), pp. 123–128, Springer, 1989, ISBN 0-387-91331-9].

[9] Björnsson, Y. and Marsland, T. A. (1999). Multi-cut pruning in alpha-beta search. *1st International Conference on Computers and Games*, Proceedings, H. J. van den Herik and H. Iida (eds.), pp. 15–24, LNCS 1558, Springer, ISBN 3-540-65766-5.

[10] Björnsson, Y. and Marsland, T. A. (1998). *Risk management in game-tree pruning*. Technical Report TR 98–07, Department of Computing Science, University of Alberta.

[11] Buro, M. (1997). *Experiments with* MULTI-PROBCUT *and a new high-quality evaluation function for Othello*. Technical Report No. 96, NEC Research Institute, 1997.

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

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

[14] Condon, J.H. and Thompson, K. (1983). BELLE. *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.

[15] Condon, J.H. and Thompson, K. (1983). BELLE chess hardware. *Advances in Computer Chess 3*, M. R. B. Clarke (ed.), pp. 45–54, Pergamon, ISBN 0-080-26898-6 [reprinted in *Computer Chess Compendium*, D. N. L. Levy (ed.), pp. 286–292, Springer, 1989, ISBN 0-387-91331-9].

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

[17] Ebeling, C. (1986). *All the Right Moves: A VLSI Architecture for Chess*. MIT Press, ISBN 0-262-05035-8.

[18] Feist, M. (1999). The 9th World Computer-Chess Championship: Report on the tournament. *ICCA Journal*, Vol. 22, No. 3, pp. 155–164.

[19] 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].

[20] 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.

[21] Hammilton, S. and Garber, L. (1997). DEEP BLUE's hardware-software synergy. *IEEE Computer*, Vol. 30, No. 10, pp. 29–35.

[22] Heinz, E. A. (1999). *Scalable Search in Computer Chess*. Dissertation (Ph.D. Thesis), University of Karlsruhe.

[23] Heinz, E. A. (1999). Knowledgeable encoding and querying of endgame databases. *ICCA Journal*, Vol. 22, No. 2, pp. 81–97.

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

[25] Heinz, E. A. (1998). Extended futility pruning. *ICCA Journal*, Vol. 21, No. 2, pp. 75–83.

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

[27] van den Herik, H. J. and Herschberg, I. S. (1992). The 7th World Computer-Chess Championship: Report on the tournament. *ICCA Journal*, Vol. 15, No. 4, pages 208–209.

[28] Hsu, F.-h. (1999). IBM's DEEP BLUE chess grandmaster chips. *IEEE Micro*, Vol. 19, No. 2, pp. 70–80.

[29] Hsu, F.-h. (1998). Designing a single-chip chess grandmaster while knowing nothing about chess. *Hot Chips 10*, Proceedings (slides of presentations), pp. 59–69.

[30] Hsu, F.-h. and Anantharaman, T. S. and Campbell, M. S. and Nowatzyk, A. (1990). A grandmaster chess machine. *Scientific American*, Vol. 263, No. 4, pp. 44–50.

[31] Hsu, F.-h. and Anantharaman, T. S. and Campbell, M. S. and Nowatzyk, A. (1990). DEEP THOUGHT. *Computers, Chess, and Cognition*, T. A. Marsland and J. Schaeffer (eds.), pp. 55–78, Springer, ISBN 0-387-97415-6/3-540-97415-6.

[32] Hyatt, R. M. and Gower, A. E. and Nelson, H. L. (1990). CRAY BLITZ. *Computers, Chess, and Cognition*, T. A. Marsland and J. Schaeffer (eds.), pp. 111–130, Springer, ISBN 0-387-97415-6/3-540-97415-6.

[33] 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.

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

[35] Newborn, M. M. (1975). *Computer Chess*. Academic Press, ISBN 0-125-17250-8.

[36] Newell, A. and Shaw, C. and Simon, H. A. (1958). Chess-playing programs and the problem of complexity. *IBM Journal of Research and Development*, Vol. 2, pp. 320–335 [reprinted in *Computers and Thought*, E. A. Feigenbaum and J. Feldman (eds.), pp. 39–70, McGraw-Hill, 1963; excerpts reprinted in *Computer Chess Compendium*, D. N. L. Levy (ed.), pp. 29–42, Springer, 1989, ISBN 0-387-91331-9].

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

[38] 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.

[39] 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].

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

[41] 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.

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

[43] 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.

[44] Tsang, H. K. and Beal, D. F. (1995). The 8th World Computer-Chess Championship: Report on the tournament and the contestants' programs described. *ICCA Journal*, Vol. 18, No. 2, pages 93–101.

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