

Extended Null-Move Reductions

Omid David-Tabibi¹ and Nathan S. Netanyahu^{1,2}

¹ Department of Computer Science, Bar-Ilan University,
Ramat-Gan 52900, Israel

`mail@omidavid.com`, `nathan@cs.biu.ac.il`

² Center for Automation Research, University of Maryland,
College Park, MD 20742, USA
`nathan@cfar.umd.edu`

Abstract. In this paper we review the conventional versions of null-move pruning, and present our enhancements which allow for a deeper search with greater accuracy. While the conventional versions of null-move pruning use reduction values of $R \leq 3$, we use an aggressive reduction value of $R = 4$ within a verified adaptive configuration which maximizes the benefit from the more aggressive pruning, while limiting its tactical liabilities. Our experimental results using our grandmaster-level chess program, FALCON, show that our *null-move reductions* (NMR) outperform the conventional methods, with the tactical benefits of the deeper search dominating the deficiencies. Moreover, unlike standard null-move pruning, which fails badly in zugzwang positions, NMR is impervious to zugzwangs. Finally, the implementation of NMR in any program already using null-move pruning requires a modification of only a few lines of code.

1 Introduction

Chess programs trying to search the same way humans think by generating “plausible” moves dominated until the mid-1970s. By using extensive chess knowledge at each node, these programs selected a few moves which they considered plausible, and thus pruned large parts of the search tree. However, plausible-move generating programs had serious tactical shortcomings, and as soon as brute-force search programs such as TECH [17] and CHESS 4.X [29] managed to reach depths of 5 plies and more, plausible-move generating programs frequently lost to brute-force searchers due to their tactical weaknesses. Brute-force searchers rapidly dominated the computer-chess field.

The introduction of null-move pruning [3,13,16] in the early 1990s marked the end of an era, as far as the domination of brute-force programs in computer chess is concerned. Unlike other forward-pruning methods (e.g., *razoring* [6], GAMMA [23], and *marginal forward pruning* [28]), which had great tactical weaknesses, null-move pruning enabled programs to search more deeply with minor tactical risks. Forward-pruning programs frequently outsearched brute-force searchers, and started their own reign which has continued ever since; they have won all World Computer Chess Championships since 1992. DEEP BLUE [18,21]

was probably the last brute-force searcher. Today almost all top-tournament playing programs use forward-pruning methods, null-move pruning being the most popular of them [14].

In this article we introduce our *extended null-move reductions*, and demonstrate empirically its improved performance in comparison to standard null-move pruning and its conventional variations. In Sect. 2 we review standard null-move pruning and its enhancements, and in Sect. 3 we introduce extended null-move reductions. Section 4 presents our experimental results, and Sect. 5 contains concluding remarks.

2 Standard Null-Move Pruning

As mentioned earlier, brute-force programs refrained from pruning any nodes in the full-width part of the search tree, deeming the risks of doing so as being too high. Null-move [3,13,16] introduced a new pruning scheme which based its cutoff decisions on dynamic criteria, and thus gained greater tactical strength in comparison with the static forward-pruning methods that were in use at that time.

Null-move pruning is based on the following assumption: in every chess position, doing nothing (i.e., doing a null move) would not be the best choice even if it were a legal option. In other words, the best move in any position is better than the null move. This idea enables us easily to obtain a lower bound α on the position by conducting a null-move search. We make a null move, i.e., we merely swap the side whose turn it is to move. (Note that this cannot be done in positions where that side is in check, since the resulting position would be illegal. Also, two null moves in a row are forbidden, since they result in nothing [13].) We then conduct a regular search with reduced depth and save the returned value. This value can be treated as a lower bound on the position, since the value of the best (legal) move has to be better than that obtained from the null move. If this value is greater than or equal to the current upper bound (i.e., $value \geq \beta$), it results in a cutoff (or what is called a fail-high). Otherwise, if it is greater than the current lower bound α , we define a narrower search window, as the returned value becomes the new lower bound. If the value is smaller than the current lower bound, it does not contribute to the search in any way. The main benefit of null-move pruning is due to the cutoffs, which result from the returned value of null-move search being greater than the current upper bound. Thus, the best way to apply null-move pruning is by conducting a minimal-window null-move search around the current upper bound β , since such a search will require a reduced search effort to determine a cutoff. A typical null-move pruning implementation is given by the pseudo-code of Fig. 1.

There are positions in chess where any move will deteriorate the position, so that not making a move is the best option. These positions are called *zugzwang* positions. While *zugzwang* positions are rare in the middle game, they are not an exception in endgames, especially endgames in which one or both sides are left with King and Pawns. Null-move pruning will fail badly in *zugzwang* positions since the basic assumption behind the method does not hold. In fact, the

```

#define R 2 // depth reduction value
int Search (alpha, beta, depth) {
    if (depth <= 0)
        return Evaluate(); // in practice, Quiescence() is called here
    // conduct a null-move search if it is legal and desired
    if (!InCheck() && NullOk()){
        MakeNullMove();
        // null-move search with minimal window around beta
        value = -Search(-beta, -beta + 1, depth - R - 1);
        UndoNullMove();
        if (value >= beta) // cutoff in case of fail-high
            return value;
    }
    // continue regular alphabeta/PVS search
    ...
}

```

Fig. 1. Standard null-move pruning

null-move search's value is an upper bound in such cases. As a result, null-move pruning is avoided in such endgame positions. Here we remark that in the early 1990s Diepeveen suggested a double null-move to handle zugzwang positions. It is an unpublished idea [12].

As previously noted, the major benefit of null-move pruning stems from the depth reduction in the null-move searches. However, these reduced-depth searches are liable to tactical weaknesses due to the *horizon effect* [5]. A horizon effect results whenever the reduced-depth search misses a tactical threat. Such a threat would not have been missed, had we conducted a search without any depth reduction. The greater the depth reduction R , the greater the tactical risk due to the horizon effect. So, the saving resulting from null-move pruning depends on the depth reduction factor, since a shallower search (i.e., a greater R) will result in faster null-move searches and an overall smaller search tree.

In the early days of null-move pruning, most programs used $R = 1$, which ensures the least tactical risk, but offers the least saving in comparison with other R values. Other reduction factors that were experimented with were $R = 2$ and $R = 3$. Research conducted over the years, most extensively by Heinz [20], showed that in his program, DARKTHOUGHT, $R = 2$ performed better than $R = 1$ and $R = 3$.

Donninger [13] was the first to suggest an adaptive rather than a fixed value for R . Experiments conducted by Heinz in his article on adaptive null-move pruning [20] showed that an adaptive rather than a fixed value can be selected for the reduction factor. By using $R = 3$ in upper parts of the search tree and $R = 2$ in its lower parts (close to the leaves) pruning can be achieved at a smaller costs (as null-move searches will be shallower) while the overall tactical strength will be maintained.

Several methods have been suggested for enabling null-move pruning to deal with zugzwang positions, but mostly at a heavy cost of making the search much more expensive [16,24]. In our 2002 paper, we introduced *verified null-move pruning* [10], which manages to cope with most zugzwang positions, with minimal additional cost. In verified null-move pruning, whenever the shallow null-move search indicates a fail-high, instead of cutting off the search from the current node, the search is continued with reduced depth. Only if another null-move fail-high occurs in the subtree of a fail-high reported node, then a cutoff will take place. Using $R = 3$ in all parts of the search tree, our experimental results showed that the size of the constructed tree was closer to that of standard $R = 3$ rather than $R = 2$ (i.e., considerably smaller tree in comparison to that constructed by using standard $R = 2$), and greater overall tactical accuracy than standard null-move pruning.

So far, all publications regarding null-move pruning considered at most a reduction value of $R = 3$, and any value greater than that was considered far too aggressive for practical use. In the next section we present our *extended null move reductions* algorithm which uses an aggressive reduction value of $R = 4$, by bringing together verified and adaptive principles.

3 Extended Null-Move Reductions

In this section we describe how we combine adaptive and verified null-move pruning concepts into our *extended null move reductions* (NMR), which enable us to use an aggressive reduction value of $R = 4$.

The greater the reduction value R is, the faster will the null-move search be, which will have a large impact on the overall size of the search tree. Thus, using $R = 4$ instead of the common values of $R = 2$ and $R = 3$ would construct a smaller search tree, enabling the program to search more deeply. However, as explained in the previous section, greater R values result in overlooking more tactical combinations. In other words, the benefit of deeper search comes at the cost of taking a greater risk of missing correct moves.

The basic idea behind NMR is using the null-move concept for reducing the search depth only, instead of pruning it altogether. Whenever the null-move search returns a value greater or equal to the upper bound, indicating fail-high ($value \geq \beta$), we reduce the depth and continue the normal search. This concept is different from verified null-move pruning where a fail-high in the subtree of a fail-high reported node results in an immediate cutoff, while in NMR, the subtree is not treated any differently.

There are some similarities between NMR and Feldmann's *fail high reductions* (FHR) [15]. In FHR, in each node a static evaluation is applied, and if the value is greater than or equal to β , the remaining depth is reduced by one ply. The major difference between NMR and FHR is that in the former the decision to reduce the depth is made after a dynamic search, while in the latter the decision is static only. In other words, in subsequent iterations when we revisit the current position, the null-move search will be deeper accordingly,

while the static evaluation at the current position will always return the same value, regardless of the search depth. As we mentioned in the Introduction, null-move pruning succeeded where other forward-pruning methods failed, thanks to basing the pruning decision on dynamic criteria.

Using the null-move concept for depth reduction instead of pruning has the advantage of reducing the tactical weaknesses caused by the horizon effect, since by continuing the search we may be able to detect threats which the shallow null-move search overlooked. Additionally, since NMR does not cutoff based on a fail-high, it is completely impervious to zugzwangs (while verified null-move manages to deal successfully with most zugzwangs, it is not completely impervious since the subtree of the fail-high node is searched normally). Thus, NMR facilitates the usage of the null-move concept even in endgames where zugzwangs are frequent.

Obviously, the disadvantage of NMR is that it has to search a larger tree in comparison to standard null-move pruning with the same R value, as the latter terminates the search at the node immediately upon a fail-high. Considering the pros and cons, the success of NMR depends on the result of this cost benefit analysis. Our experiments in the next section show that the benefit from the reduced searches justifies their additional cost.

So far, we mentioned that whenever the null-move search indicates a fail-high, in NMR we reduce the search depth and continue the normal search. The

```

// depth reduction values for null-move search
#define MAX_R 4
#define MIN_R 3
#define DR 4 // depth reduction value for normal search
int Search (alpha, beta, depth) {
    if (depth <= 0)
        return Evaluate(); // in practice, Quiescence() is called here
    // conduct a null-move search if it is legal and desired
    if (!InCheck() && NullOk()){
        MakeNullMove();
        R = depth > 6 ? MAX_R : MIN_R ;
        // null-move search with minimal window around beta
        value = -Search(-beta, -beta + 1, depth - R - 1);
        UndoNullMove();
        if (value >= beta) { // reduce the depth in case of fail-high
            depth -= DR;
            if (depth <= 0)
                return Evaluate();
        }
    }
    // continue regular alphabeta/PVS search
    ...
}

```

Fig. 2. Extended null-move reductions

success of NMR depends on the depth reduction (DR) applied here. Reducing the remaining depth by only one ply ($DR = 1$) is too conservative, as the remaining search will still be expensive. Our experiments showed that together with a reduction value of $R = 4$ for null-move search, the best reduction value for the remaining search depth is also an aggressive reduction of 4 plies ($DR = 4$). Reducing the remaining depth by a large number reduces the additional cost in comparison to standard $R = 4$ where the search is cutoff immediately.

Finally, to make this aggressive configuration safer, we also incorporate the adaptive null-move concept, i.e., we use a reduction value of $R = 3$ near leaf nodes. Using this adaptive $R = 3 \sim 4$ makes the null-move search less susceptible to overlooking tactics, while keeping the search tree small enough to justify the additional cost. Our results in the next section show that NMR with $R = 3 \sim 4$ and depth reduction of $DR = 4$ outperforms other variations of null-move pruning. Implementation of our extended null-move reductions is very easy in a program already using null-move pruning. Figure 2 shows our NMR implemented around the existing standard null-move pruning code (additions are in bold).

4 Experimental Results

Before discussing the performance of NMR in comparison to other null-move pruning variations, we would like to discuss briefly some basic issues about experimental results in computer chess. Most published papers compare various search methods to each other using fixed depth tests. Usually both method A and method B search the same test suites to fixed depths, and then the results (and number of solved positions) are compared. If method A produces a smaller tree (fewer nodes at the fixed depth) and also solves more positions, then it can be safely concluded that method A outperforms method B. However, in many cases the results will not be so clear. For example, comparing standard null-move pruning with $R = 2$ and $R = 3$, the latter constructs a smaller tree, but solves fewer positions at the fixed depth search.

Fixed time tests, in contrast to fixed depth tests, allow for an objective comparison of various methods. For example, method A can sometimes find the correct move a ply or two later than method B (e.g., because it uses a more aggressive pruning), but considering the elapsed time, method A finds the solution faster. In this case, it would be correct to say that method A performs better, even though in a fixed depth comparison method B solves more positions.

The second issue is which test suites to use. Traditionally, three standard test suites have been used for measuring tactical strength, namely Encyclopedia of Chess Middlegames (ECM), Win at Chess (WAC), and Winning Chess Sacrifices (WCS). While for many years these three test suites posed serious challenges to computer programs, today thanks to the fast hardware most of these positions succumb to the processing power in a fraction of a second. This is natural, as these three test suites were intended for testing humans not machines. Amongst the abovementioned test suites, ECM is the only one which poses some challenge to the engines, provided the time per position is limited to a small value. We

Table 1. Number of ECM positions solved by each engine (time: 5s per position)

JUNIOR 10	FRITZ 8	SHREDDER 10	HIARCS 9	CRAFTY 19	FALCON
681	640	639	642	593	644

Table 2. Total node count of standard $R = 1, 2, 3,$ and 4 and NMR $R = 3 \sim 4$ for CRAFTY benchmark

Std $R = 1$	Std $R = 2$	Std $R = 3$	Std $R = 4$	NMR $R = 3 \sim 4$
42,248,908	21,554,578	11,510,995	8,254,261	8,606,334
(+390.9%)	(+150.45%)	(+33.75%)	(-4.09%)	-

Table 3. Number of ECM positions solved by each method (time: 5s per position)

Std $R = 2$	Std $R = 3$	Std $R = 4$	Adpt $R = 2 \sim 3$	NMR $R = 3 \sim 4$
627	635	632	636	644

used the ECM test suite consisting of 879 positions, with 5 seconds per position. To double check the results (and avoid external interferences with CPU time allocations) we ran each test twice, to make sure the same results are obtained.

We conducted our experiments using FALCON, a grandmaster-level chess program which has successfully participated in two World Computer Chess Championships (7th place in 2004 World Computer Chess Championship, and 3rd place in 2004 World Computer Speed Chess Championship). FALCON uses NEGASCOUT/PVS [9,25] search, with enhancements like internal iterative deepening [2,27], dynamic move ordering (history+killer heuristic) [1,17,26], multi-cut pruning [7,8], selective extensions [2,4] (consisting of check, one-reply, mate-threat, recapture, and passed pawn extensions), transposition table [22,29], futility pruning near leaf nodes [19], and blockage detection in endgames [11]. Table 1 compares FALCON's tactical performance to other top tournament-playing engines. The results show that FALCON's tactical strength is on par with the strongest chess programs today.

Before we compare the tactical strength of various methods, we use a fixed depth benchmark of six positions (CRAFTY benchmark, see Appendix A) to show how significant the impact of the reduction value R is. Table 2 provides the total node count, comparing standard null-move pruning with reduction values of $R = 1, 2, 3,$ and 4, and NMR using $R = 3 \sim 4$ and $DR = 4$. The results clearly show that the R value has a critical role in determining the size of the constructed search tree. The table further shows that as far as node count is concerned, NMR with $R = 3 \sim 4$ is close to standard null-move with $R = 4$. But as discussed above, this table merely shows that the greater the R value is, the deeper the engine will be able to search, saying nothing about the tactical strength.

To compare the overall tactical performance, we let standard $R = 2, 3$ and 4, adaptive $R = 2 \sim 3$ and NMR $R = 3 \sim 4$ process the ECM test suite with 5

Table 4. Number of ECM positions solved by each method (time: 5s per position)

Std $R = 4$	Adpt $R = 3 \sim 4$	NMR $R = 4$	NMR $R = 3 \sim 4$
632	637	640	644

Table 5. Number of ECM positions solved by NMR using various DR values (time: 5s per position)

$DR = 1$	$DR = 2$	$DR = 3$	$DR = 4$
633	641	638	644

Table 6. 1000 self-play matches between two versions of FALCON using NMR $R = 3 \sim 4$ and Adpt $R = 2 \sim 3$, at 10 minutes per game (W% is the winning percentage, and RD is the Elo rating difference)

Match	Result	Score	W%	RD
NMR $R = 3 \sim 4$ vs. Adpt $R = 2 \sim 3$	+309 -217 =474	546.0 - 454.0	54.6%	+32

seconds per position. Table 3 provides the results. These results show that NMR $R = 3 \sim 4$ performs better than the others. We also see that standard $R = 3$ slightly outperforms both standard $R = 2$ and standard $R = 4$, with adaptive $R = 2 \sim 3$ faring about the same.

In order to see what contributes to the success of NMR $R = 3 \sim 4$, we break it down to its components. Table 4 shows a comparison of standard $R = 4$, adaptive $R = 3 \sim 4$, NMR $R = 4$, and NMR $R = 3 \sim 4$. The results show that both adaptive $R = 3 \sim 4$ and NMR $R = 4$ outperform standard $R = 4$, which explains why their combination, NMR $R = 3 \sim 4$, provides the best outcome.

Finally, in all our results above we used a depth reduction value of 4 ($DR = 4$), i.e., whenever a fail-high is indicated, the depth is reduced by 4 plies. Table 5 compares other values for DR . The results show that a value of 4 performs best.

The results so far showed that NMR $R = 3 \sim 4$ solves more positions in comparison to other methods, with adaptive $R = 2 \sim 3$ coming second. To test how NMR fares in practice, we ran 1000 self-play matches between two versions of FALCON, one using NMR $R = 3 \sim 4$ and the other using adaptive $R = 2 \sim 3$, at a time control of 10 minutes per game. Table 6 provides the results.

The results of 1000 self-play matches show that NMR $R = 3 \sim 4$ outperforms adaptive $R = 2 \sim 3$ by about 32 Elo points (see Appendix B for calculation of expected Elo difference for self-play matches). Even though this is a small rating difference, the large number of games (1000) allows for obtaining a high level of statistical confidence. At 95% statistical confidence (2 standard deviations), the rating difference is 32 ± 16 Elo, and at 99.7% statistical confidence (3 standard deviations) the rating difference is 32 ± 24 Elo. That is, NMR $R = 3 \sim 4$ is superior to adaptive $R = 2 \sim 3$ with a statistical confidence of over 99.7%.

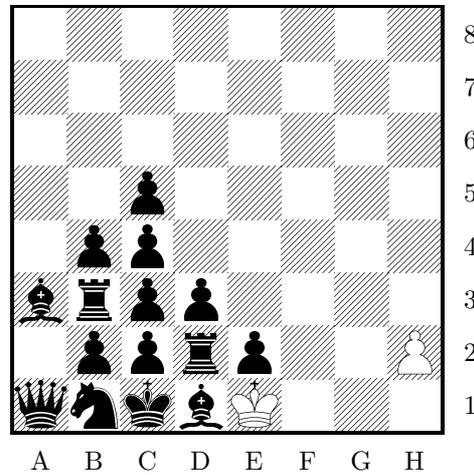


Fig. 3. 1. h3 mates in 15

Table 7. Analysis of the position in Fig. 3. All the engines are given infinite time until they reach their maximum depth.

	JUNIOR 10	FRITZ 8	SHREDDER 10	HIARCS 9	CRAFTY 19	FALCON
Move (score)	1.h4 (0.00)	1.h3 (0.00)	1.h3 (#15)	1.h4 (0.00)	1.h4 (0.00)	1.h3 (#15)
Depth	62	60	31	30	60	30

Finally, in late endgames, where zugzwangs are abundant, standard null-move pruning is completely crippled. In contrast, NMR can be safely applied to all stages of the game. The position appearing in Fig. 3, while being a constructed position unlikely to occur in a real game, shows how strong the effect of zugzwang on null-move pruning can be. The only correct move in this position is 1. h3 resulting in mate in 15. The other move 1. h4, results in a draw. Table 7 shows what each engine plays in this position, given infinite time. FALCON and SHREDDER instantly declare mate in 15 with 1. h3 at the depth of 30 plies (this suggests that SHREDDER is probably also applying some verification process to null-move pruning). The other engines search to their maximum search depth, all of them declaring a draw. FRITZ produces the correct move 1. h3 but with a draw score, suggesting that it has just randomly picked 1. h3 instead of 1. h4.

5 Conclusion

In this article we introduced *extended null-move reductions*, which outperformed conventional null-move pruning techniques both in tactical tests and in long

series of self-play matches. This method facilitates a safe use of the aggressive reduction value of $R = 4$, which is widely considered as too aggressive for practical use. It results in a considerably smaller search tree, enabling the program to search more deeply, thus improving its tactical and positional performance. Moreover, NMR, by the fact that it does not prune based on fail-high, is impervious to zugzwang, and so it can be safely employed in all stages of the game.

NMR and its modified versions have been evolving in FALCON for the past six years, and the results have been promising. In this paper we provided a small fraction of the experiments we have conducted during this period. However, despite our success, we would like to be cautious with any generalization. FALCON has an aggressively tuned king-safety evaluation, and uses many extensions in its search that enable it to spot faster tactical combinations. As such, it is possible that our aggressive method works in FALCON because other components of the engine are tuned for detecting tactics, and they “cover” the blind spots of our NMR.

We believe the main contribution of this paper is that it presents a method for successful incorporation of the seemingly impractical value of $R = 4$ within the null-move search, and even if our method does not achieve exactly the same result in another program, we believe trying other implementations using $R = 4$ is worthy of experimenting with, due to the high potential reward.

In this paper we presented one of the enhancements we have developed during the past few years. It is very probable that our method, or improved incarnations of it, are independently developed by the programmers of other top chess engines.

Acknowledgments. We would like to thank Vincent Diepeveen for his enlightening remarks and suggestions. We would also like to thank the two anonymous referees for their helpful comments.

References

1. Akl, S.G., Newborn, M.M.: The principal continuation and the killer heuristic. In: Proceedings of the 5th Annual ACM Computer Science Conference, pp. 466–473. ACM Press, Seattle (1977)
2. Anantharaman, T.S.: Extension heuristics. *ICCA Journal* 14(2), 47–65 (1991)
3. Beal, D.F.: Experiments with the null move. In: Beal, D.F. (ed.) *Advances in Computer Chess* 5, pp. 65–79. Elsevier Science Publishers, Amsterdam (1989)
4. Beal, D.F., Smith, M.C.: Quantification of search extension benefits. *ICCA Journal* 18(4), 205–218 (1995)
5. Berliner, H.J.: *Chess as Problem Solving: The Development of a Tactics Analyzer*. Ph.D. thesis, Carnegie-Mellon University, Pittsburgh, PA (1974)
6. Birmingham, J.A., Kent, P.: Tree-searching and tree-pruning techniques. In: Clarke, M.R.B. (ed.) *Advances in Computer Chess* 1, pp. 89–107. Edinburgh University Press, Edinburgh (1977)
7. Björnsson, Y., Marsland, T.: Multi-cut pruning in alpha-beta search. In: Proceedings of the 1st International Conference on Computers and Games, pp. 15–24 (1998)
8. Björnsson, Y., Marsland, T.: Multi-cut alpha-beta-pruning in game-tree search. *Theoretical Computer Science* 252(1-2), 177–196 (2001)

9. Campbell, M.S., Marsland, T.A.: A comparison of minimax tree search algorithms. *Artificial Intelligence* 20(4), 347–367 (1983)
10. David-Tabibi, O., Netanyahu, N.S.: Verified null-move pruning. *ICGA Journal* 25(3), 153–161 (2002)
11. David-Tabibi, O., Felner, A., Netanyahu, N.S.: Blockage detection in pawn endings. In: van den Herik, H.J., Björnsson, Y., Netanyahu, N.S. (eds.) *CG 2004. LNCS*, vol. 3846, pp. 187–201. Springer, Heidelberg (2006)
12. Diepeveen, V.: Private communication (2008)
13. Donninger, C.: Null move and deep search: Selective search heuristics for obtuse chess programs. *ICCA Journal* 16(3), 137–143 (1993)
14. Feist, M.: The 9th World Computer-Chess Championship: Report on the tournament. *ICCA Journal* 22(3), 155–164 (1999)
15. Feldmann, R.: Fail high reductions. In: van den Herik, H.J., Uiterwijk, J.W.H.M. (eds.) *Advances in Computer Chess 8*, pp. 111–128. Universiteit Maastricht (1996)
16. Goetsch, G., Campbell, M.S.: Experiments with the null-move heuristic. In: Marsland, T.A., Schaeffer, J. (eds.) *Computers, Chess, and Cognition*, pp. 159–168. Springer, New York (1990)
17. Gillogly, J.J.: The technology chess program. *Artificial Intelligence* 3(1-3), 145–163 (1972)
18. Hamilton, S., Garber, L.: Deep Blue’s hardware-software synergy. *IEEE Computer* 30(10), 29–35 (1997)
19. Heinz, E.A.: Extended futility pruning. *ICCA Journal* 21(2), 75–83 (1998)
20. Heinz, E.A.: Adaptive null-move pruning. *ICCA Journal* 22(3), 123–132 (1999)
21. Hsu, F.-h.: IBM’s DEEP BLUE chess grandmaster chips. *IEEE Micro* 19(2), 70–80 (1999)
22. Nelson, H.L.: Hash tables in CRAY BLITZ. *ICCA Journal* 8(1), 3–13 (1985)
23. Newborn, M.M.: *Computer Chess*. Academic Press, New York (1975)
24. Plenkner, S.: A null-move technique impervious to zugzwang. *ICCA Journal* 18(2), 82–84 (1995)
25. Reinefeld, A.: An improvement to the SCOUT tree-search algorithm. *ICCA Journal* 6(4), 4–14 (1983)
26. Schaeffer, J.: The history heuristic. *ICCA Journal* 6(3), 16–19 (1983)
27. Scott, J.J.: A chess-playing program. In: Meltzer, B., Michie, D. (eds.) *Machine Intelligence 4*, pp. 255–265. Edinburgh University Press, Edinburgh (1969)
28. Slagle, J.R.: *Artificial Intelligence: The Heuristic Programming Approach*. McGraw-Hill, New York (1971)
29. Slate, D.J., Atkin, L.R.: Chess 4.5 – The Northwestern University chess program. In: Frey, P.W. (ed.) *Chess Skill in Man and Machine*, 2nd edn., pp. 82–118. Springer, New York (1983)

Appendix

A Experimental Setup

Our experimental setup consisted of the following resources:

- 879 positions from *Encyclopedia of Chess Middlegames* (ECM).
- FALCON, JUNIOR 10, FRITZ 8, SHREDDER 10, HIARCS 9, and CRAFTY 19 chess engines, running on AMD 3200+ with 1 GB RAM and Windows XP operating system.

- Fritz 8 interface for automatic running of test suites and self-play matches (FALCON was run as a UCI engine).
- CRAFTY benchmark for fixed depth search, consisting of the following six positions:

```

D=11: 3r1k2/4npp1/1ppr3p/p6P/P2PPPP1/1NR5/5K2/2R5 w - - 0 1 D=11:
rnbqkb1r/p3pppp/1p6/2ppP3/3N4/2P5/PPP1QPPP/R1B1KB1R w KQkq - 0 1
D=14: 4b3/p3kp2/6p1/3pP2p/2pP1P2/4K1P1/P3N2P/8 w - - 0 1 D=11:
r3r1k1/ppqb1ppp/8/4p1NQ/8/2P5/PP3PPP/R3R1K1 b - - 0 1 D=12:
2r2rk1/1bqnbpp1/1p1ppn1p/pP6/N1P1P3/P2B1N1P/1B2QPP1/R2R2K1 b - - 0 1
D=11: r1bqk2r/pp2bppp/2p5/3pP3/P2Q1P2/2N1B3/1PP3PP/R4RK1 b kq - 0 1

```

B Elo Rating System

The Elo rating system, developed by Prof. Arpad Elo, is the official system for calculating the relative skill levels of players in chess. Given the rating difference (RD) of two players, the following formula calculates the expected winning rate (W , between 0 and 1) of the player:

$$W = \frac{1}{10^{-RD/400} + 1}$$

Given the winning rate of a player, as is the case in our experiments, the expected rating difference can be derived from the above formula:

$$RD = -400 \log_{10}\left(\frac{1}{W} - 1\right)$$