VERIFIED NULL-MOVE PRUNING

Omid David-Tabibi [1] *Nathan S. Netanyahu* [2]

Ramat-Gan, Israel

ABSTRACT

In this article we review standard null-move pruning and introduce our extended version of it, which we call *verified null-move pruning*. 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.

Our experiments with verified null-move pruning show that on average, it constructs a smaller search tree with greater tactical strength in comparison to standard null-move pruning. Moreover, unlike standard null-move pruning, which fails badly in zugzwang positions, verified null-move pruning manages to detect most zugzwangs and in such cases conducts a re-search to obtain the correct result. In addition, verified null-move pruning is very easy to implement, and any standard null-move pruning program can use verified null-move pruning by modifying only a few lines of code.

1. INTRODUCTION

Until the mid-1970s most chess programs were trying to search the same way humans think, by generating "plausible" moves. 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 like TECH (Gillogly, 1972) and CHESS 4.X (Slate and Atkin, 1977) 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.

Most brute-force searchers of that time used no selectivity in their full-width search tree, except for some extensions, consisting mostly of check extensions and recaptures. The most successful of these brute-force programs were BELLE (Condon and Thompson, 1983a,b), DEEP THOUGHT (Hsu, Anantharaman, Campbell, and Nowatzyk, 1990), HITECH (Berliner and Ebeling, 1990; Berliner, 1987; Ebeling, 1986), and CRAY BLITZ (Hyatt, Gower, and Nelson, 1990), which for the first time managed to compete successfully against humans.

The introduction of null-move pruning (Beal, 1989; Goetsch and Campbell, 1990; Donninger, 1993) 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* (Birmingham and Kent, 1977), GAMMA (Newborn, 1975), and *marginal forward pruning* (Slagle, 1971)), 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 (van den Herik and Herschberg, 1992; Tsang and Beal, 1995; Feist, 1999). DEEP BLUE (Hammilton and Garber, 1997; Hsu, 1999) (the direct descendant of DEEP THOUGHT (Hsu *et al.*, 1990)) 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 (Feist, 1999).

In this article we introduce our new *verified null-move pruning* method, and demonstrate empirically its improved performance in comparison with standard null-move pruning. This is reflected in its reduced search

[1] Department of Computer Science, Bar-Ilan University, Ramat-Gan 52900, Israel, Email: mail@omiddavid.com, Web: http://www.omiddavid.com.

[2] Department of Computer Science, Bar-Ilan University, Ramat-Gan 52900, Israel, Email: nathan@cs.biu.ac.il, and Center for Automation Research, University of Maryland, College Park, MD 20742, USA, Email: nathan@cfar.umd.edu.

tree size, as well as its greater tactical strength. In Section 2 we review standard null-move pruning, and in Section 3 we introduce verified null-move pruning. Section 4 presents our experimental results, and Section 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 (Beal, 1989; Goetsch and Campbell, 1990; Donninger, 1993) 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.) 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 Figure 1.

```
/* the depth reduction factor */
#define R  2
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 (!in_check() && null_ok()){
      make_null_move();
      /* null-move search with minimal window around beta */
      value = -search(-beta, -beta + 1, depth - R - 1);
      if (value >= beta) /* cutoff in case of fail-high */
         return value;
   }
   /* continue regular NegaScout/PVS search */
   ...
}
```

Figure 1: Standard null-move pruning.

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 null-move search's value is an upper bound in such cases. As a result, null-move pruning is avoided in such endgame positions.

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* (Berliner, 1974). 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.

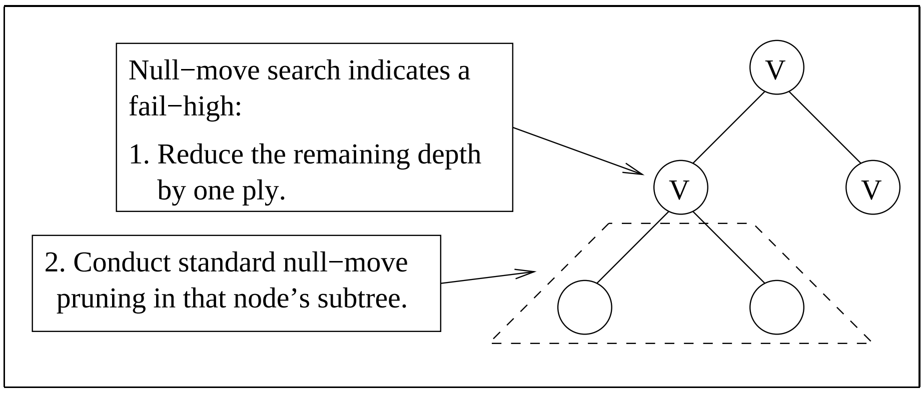
In the early days of null-move pruning, most programs used $R = 1$, which ensures the least tactical risk, but offers the least saving in comparison with other R values. Other reduction factors that were experimented with were $R = 2$ and $R = 3$. Research conducted over the years, most extensively by Heinz (1999), showed that overall, $R = 2$ performs better than the too conservative $R = 1$ and the too aggressive $R = 3$. Today, almost all null-move pruning programs, use at least $R = 2$ (Feist, 1999). However, using $R = 3$ is tempting, considering the reduced search effort resulting from shallower null-move searches. (This will be demonstrated in Section 4.) Donninger (1993) was the first to suggest an adaptive rather than a fixed value for R. Experiments conducted by Heinz (1999), in his article on adaptive null-move pruning, suggest that using $R = 3$ in upper parts of the search tree and $R = 2$ in its lower parts can save 10 to 30 percent of the search effort in comparison with a fixed $R = 2$, while maintaining overall tactical strength.

In the next section we present a new null-move pruning method which allows the use of $R = 3$ in all parts of the search tree, while alleviating to a significant extent the main disadvantage of standard null-move pruning.

3. VERIFIED NULL-MOVE PRUNING

Cutoffs based on a shallow null-move search can be too risky at some points, especially in zugzwang positions. Goetsch and Campbell (1990) hinted at continuing the search with reduced depth, in case the null-move search indicates a fail-high, in order to substantiate that the value returned from the null-move search is indeed a lower bound on the position. Plenkner (1995) showed that this idea can help prevent errors due to zugzwangs. However, verifying the search in the middle game seems wasteful, as it appears to undermine the basic benefit of null-move pruning, namely that a cutoff is determined by a shallow null-move search.

In addition to helping in detecting zugzwangs, the idea of not immediately pruning the search tree (based on the value returned from the shallow null-move search) can also help to reduce 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 has failed to detect. Based on these ideas, we developed our own reformulation, which we call *verified null-move pruning*. At each node, we conduct a null-move search with a depth reduction of $R = 3$. If the returned value from that null-move search indicates a fail-high (i.e., $value \geq \beta$), we then reduce the depth by one ply and continue the search in order to verify the cutoff. However, for that node's subtree, we use standard null-move pruning (cutoff takes place upon fail-highs). See Figure 2, for an illustration.


Null−move search indicates a fail−high:
1. Reduce the remaining depth by one ply.
2. Conduct standard null−move pruning in that node's subtree.
V
V
V
Figure 2: Illustration of verified null-move pruning.

The basic idea behind verified null-move pruning is that null-move search with $R = 3$ constructs a considerably smaller search tree. However, because of its tactical deficiencies, a cutoff based on it is too risky. So upon a fail-high, we reduce the depth and continue the search, using standard null-move pruning (with $R = 3$) in that node's subtree. The search at a node is thus cut off (based on its null-move search) only if there has been another null-move search fail-high indication in one of the node's ancestors (see Figure 2). As the experimental results in the next section show, verified null-move pruning constructs a search tree which is close in size to that of standard null-move pruning with $R = 3$, and whose tactical strength is greater on average than that of standard null-move pruning with $R = 2$. This is a smaller search tree with greater tactical strength, in comparison with standard null-move pruning with $R = 2$, which is commonly used nowadays.

Since upon a fail-high indication we do not cut off the search at once, we have the ability to check whether the returned value is indeed a lower bound on the position. If the null-move search indicates a cutoff, but the search shows that the best value is smaller than β, this implies that the position is a zugzwang, as the value from the null move is greater than or equal to the value from the best move. In such cases, we restore the original depth (which was reduced by one ply after the fail-high indication), and conduct a re-search to obtain the correct value.

Implementation of verified null-move search is a matter of adding a few lines of code to standard null-move search, as shown in Figure 3. Regarding the pseudo-code presented, when the search starts at the root level, the flag `verify` is initialized to `true`. When the null-move search indicates a fail-high, the remaining depth is reduced by one ply, and `verify` is given the value `false`, which will be passed to the children of the current node, indicating that standard null-move pruning will be conducted with respect to the children. Upon a fail-high indication due to the standard null-move search of these children's subtrees, cutoff takes place immediately.

```
#define R  3 /* the depth reduction factor */
/* at the root level, verify = true */
int search (alpha, beta, depth, verify) {
   if (depth <= 0)
      return evaluate(); /* in practice, quiescence() is called here */
   /* if verify = true, and depth = 1, null-move search is not conducted, since
    * verification will not be possible */
   if (!in_check() && null_ok() && (!verify || depth > 1)) {
      make_null_move();
      /* null-move search with minimal window around beta */
      value = -search(-beta, -beta + 1, depth - R - 1,
                         verify);
      if (value >= beta) { /* fail-high */
         if (verify) {
            depth--; /* reduce the depth by one ply */
            /* turn verification off for the sub-tree */
            verify = false;
            /* mark a fail-high flag, to detect zugzwangs later*/
            fail_high = true;
         }
         else /* cutoff in a sub-tree with fail-high report */
            return value;
      }
   }
re_search: /* if a zugzwang is detected, return here for re-search */
   /* do regular NegaScout/PVS search */
   /* search() is called with current value of "verify" */
   ...
   /* if there is a fail-high report, but no cutoff was found, the position
    * is a zugzwang and has to be re-searched with the original depth */
   if(fail_high && best < beta) {
      depth++;
      fail_high = false;
      verify = true;
      goto re_search;
   }
}
```

Figure 3: Verified null-move pruning.

4. EXPERIMENTAL RESULTS

In this section we examine the performance of verified null-move pruning, focusing on its tactical strength and smaller search-tree size in comparison with standard null-move pruning. We conducted our experiments using the GENESIS[3] engine. GENESIS is designed especially for research, emphasizing accurate implementation of algorithms and detailed statistics. For our experiments we used the NEGASCOUT/PVS (Campbell and Marsland, 1983; Reinefeld, 1983) search algorithm, with history heuristic (Schaeffer, 1983, 1989) and transposition table (Slate and Atkin, 1977; Nelson, 1985). To demonstrate the tactical strength differences between the different methods even better, we used one-ply check extensions on leaf nodes; the quiescence search consisted only of captures/recaptures. In all test suites used, we discarded positions in which at least one side had no more than King and Pawns. This was done to avoid dealing with zugzwang positions, for which verified null-move pruning obviously fares much better tactically, as explained before.

In order to obtain an estimate of the search tree, we searched 138 test positions from *Test Your Tactical Ability* by Yakov Neishtadt (see the Appendix) to depths of 9 and 10 plies, using standard $R = 1$, $R = 2$, $R = 3$, and verified $R = 3$. Table 1 gives the total node count for each method and the size of the tree in comparison with verified $R = 3$. Table 2 gives the number of positions that each method solved correctly (i.e., found the correct variation for). Later we will further examine the tactical strength, using additional test suites.

Depth	Std $R = 1$	Std $R = 2$	Std $R = 3$	Vrfd $R = 3$
9	1,652,668,804 (+267.46%)	603,549,661 (+34.19%)	267,208,422 (-40.58%)	449,744,588 -
10	11,040,766,367 (+661.64%)	1,892,829,685 (+30.57%)	862,153,828 (-40.52%)	1,449,589,289 -

Table 1: Total node count of standard $R = 1, 2, 3$ and verified $R = 3$ at depths 9 and 10, for 138 Neishtadt test positions.

The results in Tables 1 and 2 reveal that the size of the tree constructed by verified null-move pruning is between those of standard $R = 2$ and $R = 3$, and that its tactical strength is greater on average than that of standard $R = 2$. These results also show that the use of $R = 1$ is impractical due to its large tree size in comparison with other depth-reduction values. Focusing on the practical alternatives (i.e., standard $R = 2$ and $R = 3$, and verified $R = 3$), we would like to examine the behavior of verified $R = 3$ and find out whether its tree size remains between the tree sizes associated with $R = 2$ and $R = 3$, or whether it approaches the size of one of these trees. We therefore conducted a search to a depth of 11 plies, using 869 positions from the *Encyclopedia of Chess Middlegames* (ECM)[4]. Table 3 provides the total node counts at depths 9, 10, and 11, using standard $R = 2$, $R = 3$, and verified $R = 3$. See also Figure 4.

As Figure 4 clearly indicates, for depth 11 the size of the tree constructed by verified null-move pruning with $R = 3$ is closer to standard null-move pruning with $R = 3$. This implies that the saving from verified null-move pruning will be greater as we search more deeply. This can be explained by the fact that the saving from the use of $R = 3$ in the shallow null-move search far exceeds the verification cost of verified null-move pruning.

Having studied the effect of verified null-move pruning on the search tree size, we now take a closer look at the resulting tactical strength in comparison with standard null-move pruning with different depth reductions. For this purpose we used 999 positions from the *Winning Chess Sacrifices* (WCS) test suite, and 434 positions of “mate in 4” and 353 positions of “mate in 5” from the test suites of the *Chess Analysis Project* (CAP); see the Appendix. The WCS positions were searched to depths of 8, 9, and 10 plies, using standard $R = 2$, $R = 3$, and verified $R = 3$. Table 4 provides the total node counts, and Table 5 gives the number of correctly solved positions for the WCS test suite. For each position of “mate in 4” we conducted a search to a depth of 8 plies, and for each “mate in 5” position a search to a depth of 10 plies. The search was conducted using standard

[3]http://www.omiddavid.com/genesis

[4]Because of the large number of errors in ECM’s suggested best moves, we did not check here for number of solved positions.

Depth	Std $R = 1$	Std $R = 2$	Std $R = 3$	Vrfd $R = 3$
9	64	62	53	60
10	71	66	65	71

Table 2: Number of solved positions using standard $R = 1, 2, 3$ and verified $R = 3$ at depths 9 and 10, for 138 Neishtadt test positions.

Depth	Std $R = 2$	Std $R = 3$	Vrfd $R = 3$
9	5,374,275,763 (+10.84%)	2,483,951,601 (-48.76%)	4,848,596,820 -
10	16,952,333,579 (+17.40%)	7,920,812,800 (-45.14%)	14,439,185,304 -
11	105,488,197,524 (+106.51%)	24,644,668,194 (-51.75%)	51,080,338,048 -

Table 3: Total node count of standard $R = 2$, $R = 3$, and verified $R = 3$ at depths 9, 10, and 11, for 869 ECM test positions.

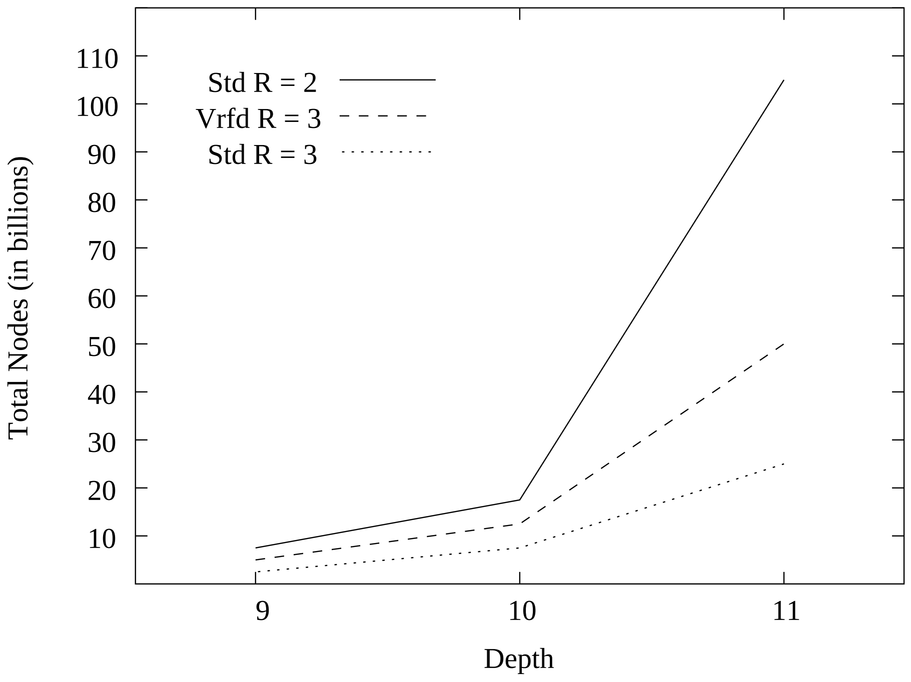


Figure 4: Tree sizes of standard $R = 2$, $R = 3$, and verified $R = 3$ at depths 9, 10, and 11, for 869 ECM test positions.

Depth	Std $R = 2$	Std $R = 3$	Vrfd $R = 3$
8	783,461,647 (-13.55%)	533,282,695 (-41.15%)	906,225,552 -
9	3,742,064,688 (+47.38%)	1,316,719,980 (-48.14%)	2,539,057,043 -
10	11,578,143,939 (+46.75%)	4,871,295,877 (-38.26%)	7,889,544,754 -

Table 4: Total node count of standard $R = 2$, $R = 3$ and verified $R = 3$ at depths 8, 9, and 10, for 999 WCS test positions.

Depth	Std $R = 2$	Std $R = 3$	Vrfd $R = 3$
8	762	760	782
9	838	812	838
10	850	849	866

Table 5: Number of solved positions using $R = 2$, $R = 3$ and verified $R = 3$ at depths 8, 9, and 10 for 999 WCS test positions.

$R = 1$, $R = 2$, $R = 3$, and verified $R = 3$. Table 6 provides the number of positions that each method solved (i.e., found the checkmating sequence).

Test Suite	Std $R = 1$	Std $R = 2$	Std $R = 3$	Vrfd $R = 3$
"Mate in 4" Depth 8 plies	433	385	379	431
"Mate in 5" Depth 10 plies	347	292	286	340

Table 6: Numbers of solved positions using standard $R = 1, 2, 3$ and verified $R = 3$ for 434 "mate in 4" and 353 "mate in 5" test suites.

The results in Tables 5 and 6 indicate that verified null-move pruning solved far more positions than standard null-move pruning with depth reductions of $R = 2$ and $R = 3$. This demonstrates that not only does verified null-move pruning result in a reduced search effort (the constructed search tree is closer in size to that of standard $R = 3$), but its tactical strength is greater than that of standard $R = 2$, which is the common depth reduction value.

Finally, to study the overall advantage of verified null-move pruning over standard null-move pruning in practice, we conducted 100 self-play games, using two versions of the GENESIS engine, one with verified $R = 3$ and the other with standard $R = 2$. The time control was set to 60 minutes per game. The version using verified $R = 3$ scored 68.5 out of 100 (see the Appendix), which demonstrates the superiority of verified null-move pruning over the standard version.

5. CONCLUSION

In this article we introduced a new null-move pruning method which outperforms standard null-move pruning techniques, in terms of reducing the search tree size as well as gaining greater tactical strength. The idea of not cutting off the search as soon as the shallow null-move search indicates a fail-high allows verification of the cutoff, which results in greater tactical accuracy and prevents errors due to zugzwangs. We showed empirically that verified null-move pruning with a depth reduction of $R = 3$ constructs a search tree which is closer in size to that of the tree constructed by standard $R = 3$, and that the saving from the reduced search effort in comparison with standard $R = 2$ becomes greater as we search more deeply. We also showed that on average, the tactical strength of verified null-move pruning is greater than that of standard null-move pruning with $R = 2$. Moreover, verified null-move pruning can be implemented within any standard null-move pruning framework by merely adding a few lines of code.

We considered a number of variants of standard null-move pruning. The first variant was not to cut off at all upon fail-high reports, but rather reduce the depth by 2 plies. We obtained good results with this idea, but its tactical strength was sometimes smaller than that of standard $R = 2$. We concluded that in order to improve the results, the depth should not be reduced by more than one ply at a time upon fail-high reports. An additional variant was not to cut off at any node, not even in the subtree of a node with a fail-high report, but merely to reduce the depth by one ply upon a fail-high report. Unfortunately, the size of the resulting search tree exceeded the size of the tree constructed by standard $R = 2$. Still, another variant was to reduce the depth by one ply upon fail-high reports, and to reduce the depth by two plies upon fail-high reports in that node's subtree, rather than cutting off.

Our empirical studies showed that cutting off the search at the subtree of a fail-high reported node does not decrease tactical strength. Indeed, this is the verified null-move pruning version that we studied in this article. In contrast to the standard approach which advocates the use of immediate cutoff, the novel approach taken here uses depth reduction, and delays cutting off the search until further verification. This yields greater tactical strength and a smaller search tree.

6. REFERENCES

Beal, D.F. (1989). Experiments with the null move. *Advances in Computer Chess 5,* (Ed. D.F. Beal) , pp. 65–79. Elsevier Science Publishers, Amsterdam, The Netherlands. ISBN 0-444-87-159-4.

Berliner, H.J. (1974). *Chess as Problem Solving: The Development of a Tactics Analyzer*. Ph.D. thesis, Carnegie-Mellon University, Pittsburgh, PA.

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

Berliner, H.J. and Ebeling, C. (1990). HITECH. *Computers, Chess and Cognition* (Eds. T.A. Marsland and J. Schaeffer), pp. 79–109. Springer-Verlag, New York, N.Y. ISBN 0-387-97415-6/3-540-97415-6.

Birmingham, J.A. and Kent, P. (1977). Tree-searching and tree-pruning techniques. *Advances in Computer Chess 1,* (Ed. M.R.B. Clarke), pp. 89–107. Edinburgh University Press, Edinburgh. ISBN 0-852-24292-1.

Campbell, M.S. and Marsland, T.A. (1983). A comparison of minimax tree search algorithms. *Artificial Intelligence,* Vol. 20, No. 4, pp. 347–367. ISSN 0004-3702.

Condon, J.H. and Thompson, K. (1983a). BELLE. *Chess Skill in Man and Machine*, (Ed. P.W. Frey), pp. 201–210. Springer-Verlag, New York, N.Y., 2nd ed., ISBN 0-387-90790-4/3-540-90790-4.

Condon, J.H. and Thompson, K. (1983b), BELLE chess hardware. *Advances in Computer Chess 3*, (Ed. M.R.B. Clarke), pp. 45–54. Pergamon Press, Oxford, ISBN 0-080-26898-6.

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

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

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

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

Gillogly, J.J. (1972). The technology chess program. *Artificial Intelligence*, Vol. 3, No. 1-3, pp. 145–163. ISSN 0004-3702.

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

Heinz, E.A. (1999). Adaptive null-move pruning, *ICCA Journal*, Vol. 22, No. 3, pp. 123–132.

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

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

Hsu, F.-h., Anantharaman, T.S., Campbell, M.S., and Nowatzyk, A. (1990). DEEP THOUGHT. *Computers, Chess, and Cognition,* (Eds. T.A. Marsland and J. Schaeffer), pp. 55–78. Springer-Verlag, New York, N.Y. ISBN 0-387-97415-6/3-540-97415-6.

Hyatt, R.M., Gower, A.E., and Nelson, H.L. (1990). CRAY BLITZ, *Computers, Chess, and Cognition*, (Eds. T.A. Marsland and J. Schaeffer), pp. 111–130. Springer-Verlag, New York, N.Y. ISBN 0-387-97415-6/3-540-97415-6.

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

Newborn, M.M. (1975). *Computer Chess*. Academic Press. New York, N.Y. ISBN 0-125-17250-8.

Plenkner, S. (1995). A null-move technique impervious to zugzwang. *ICCA Journal*, Vol. 18, No. 2, pp. 82–84.

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

Sc haeffer, J. (1983). The history heuristic. *ICCA Journal*, Vol. 6, No. 3, pp. 16–19.

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. ISSN 0162-8828.

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

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

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, pp. 93–101.

7. ACKNOWLEDGEMENTS

We would like to thank Shay Bushinsky for his interest in our research, and for promoting the discipline of Computer Chess in our department. We would also like to thank Dann Corbit for providing the CAP test positions for our empirical studies, and Azriel Rosenfeld for his editorial comments. Finally, we are indebted to Jonathan Schaeffer and Christian Donninger for their enlightening remarks and suggestions.

8. APPENDIX

EXPERIMENTAL SETUP

Our experimental setup consisted of the following resources:

- 138 positions (Diagrams 241 to 378) from: Yakov Neishtadt (1993). *Test Your Tactical Ability*, pp. 110–135. Batsford, ISBN 0-7134-4013-9.
- 869 positions from *Encyclopedia of Chess Middlegames*, and 999 positions from *Winning Chess Sacrifices*, as available on the Internet.
- 434 "Mate in 4" and 353 "Mate in 5" positions from *Chess Analysis Project*, available at ftp://cap.connx.com/
- GENESIS chess engine, with 2^{22} transposition table entries (64MB), running on a 733 MHz Pentium III with 256MB RAM, with the Windows 98 operating system.

The webpage http://www.omiddavid.com/pubs.html contains additional information about the test suites, move lists of self-play games, and detailed experimental results.