Enhanced Forward Pruning

Mark H.M. Winands * H. Jaap van den Herik
Jos W.H.M. Uiterwijk Erik C.D. van der Werf

Department of Computer Science, Institute for Knowledge and Agent Technology, Universiteit Maastricht, P.O. Box 616, 6200 MD Maastricht, The Netherlands

Abstract

In this paper forward-pruning methods, such as multi-cut and null move, are tested at so-called ALL nodes. We improved the Principal Variation Search by four small but essential additions. The new PVS algorithm guarantees that forward pruning is safe at ALL nodes. Experiments show that multi-cut at ALL nodes (MC-A) when combined with other forward-pruning mechanisms give a significant reduction of the number of nodes searched. In comparison, a (more) aggressive version of the null move (variable null-move bound) gives less reduction at expected ALL nodes. Finally, it is demonstrated that the playing strength of the Lines of Action program MIA is significantly (scoring 21 per cent more winning points than the opponent) increased by MC-A.

Key words: $\alpha\beta$ search, multi-cut pruning, null move, Lines of Action

1 Introduction

For a long time, brute-force $\alpha\beta$ search was the standard procedure in games such as chess and checkers. Over the years many improvements have been proposed. An obvious improvement is the introduction of a variable-depth search, i.e., exploring promising moves more deeply (extending the search) and non-promising moves less deeply (pruning the search). The use of selective extensions provided better results right from the beginning. An example of

* Corresponding author Tel.: (+31) (0)43-38 83898; fax: (+31) (0)43-38 84897
Email addresses: m.winands@cs.unimaas.nl (Mark H.M. Winands), herik@cs.unimaas.nl (H. Jaap van den Herik), uiterwijk@cs.unimaas.nl (Jos W.H.M. Uiterwijk), e.vanderwerf@cs.unimaas.nl (Erik C.D. van der Werf).

a knowledge-independent technique which explores some continuations more deeply is singular extensions [2]. Enhancing the search with forward pruning led to mixed results until the 1990s, when the null move [3, 10, 12] was introduced. Recently, a new forward-pruning mechanism, called multi-cut [5], has been proposed. Meanwhile it has been adopted by some of the world's strongest commercial chess programs [4]. These two forward-pruning techniques have in common that, again, no explicit domain knowledge is required to make the search decision. Traditionally, multi-cut forward pruning was not applied at *expected* ALL nodes, since it was assumed that the technique was only successful at CUT nodes. However, in this paper we investigate whether it is beneficial to use forward-pruning methods at expected ALL nodes in the Principal Variation Search (PVS) framework [16]. Next to PVS, NegaScout [19] is a popular choice to apply forward-pruning techniques. The two algorithms are essentially equivalent to each other; they expand the same search tree [4]. Besides the multi-cut an aggressive version of the null move is tested at expected ALL nodes. The remainder of this paper is organised as follows. Section 2 explains the different node types in the $\alpha\beta$ search. Next, the way forward pruning works in PVS is explained in section 3. Subsequently, multi-cut is described in section 4. The method is tested at expected ALL nodes and compared to an aggressive version of the null move in section 5. Finally, section 6 gives the conclusions and future research.

2 Three Node Types

Knuth and Moore [14] identified three types of nodes in the $\alpha\beta$ minimax tree: type-one, type-two, and type-three nodes. In this paper we use the terminology introduced by Marsland and Popowich [18]. They identify the nodes as PV, CUT, and ALL nodes, respectively. The root of the tree is a PV node. At a PV node all the children have to be investigated. The best move found at a PV node leads to a successor PV node, while all the other investigated children are CUT nodes. At a CUT node the child causing a β-cutoff is an ALL node. In a perfectly ordered tree only one child of a CUT node has to be explored. At an ALL node all the children have to be explored. The successors of an ALL node are CUT nodes.

Before searching a node we do not really know what the type of the node will be. Thus, before exploring nodes we refer to them as *expected* PV nodes, *expected* CUT nodes and *expected* ALL nodes. If none of the moves causes a cutoff at an expected CUT node, the node becomes an ALL node. If one of the children at an expected ALL node turns out not to be a CUT node, the expected ALL node becomes a CUT node. When all expected CUT nodes on a path from the root to a leaf node have become ALL nodes, a new principal variation has emerged (all the nodes on the path have become in fact PV

nodes). In Figure 1 the different types of nodes are depicted in a tree. True PV, CUT and ALL nodes are denoted by P, C and A. Expected CUT and expected ALL nodes, that turn to have a different node type, are denoted by $\underline{C}$ and $\underline{A}$. Shaded nodes are not belonging to the minimal tree.

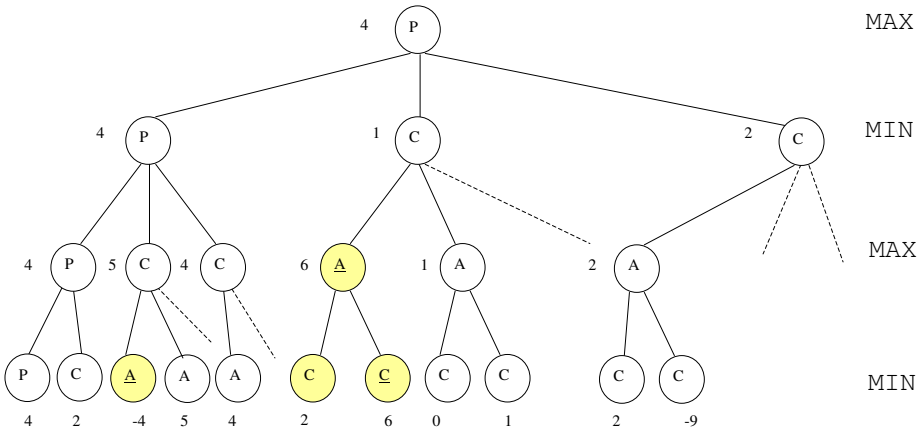

Fig. 1. An example of $\alpha\beta$ tree with different types of nodes

The Principal Variation Search (PVS) framework [16] is able to determine the expected and true type of a node. This algorithm is in general more efficient than the original $\alpha\beta$. The algorithm considers the first node explored at the root (and at subsequent PV nodes) to be a PV node. The value of that node is therefore treated as best value, and all the siblings are searched using a closed $\alpha\beta$-window (i.e., $\beta = \alpha + 1$) to prove that they are inferior. This part of the search is called zero-width-window search [17] or null-window search (NWS) [8]. In the NWS, nodes are assumed not to be on the principal variation and therefore are expected to be alternately CUT and ALL nodes. If the NWS returns a score less than or equal to α, then that particular sibling has been proved inferior. Sometimes, the NWS returns a better score and a re-search has to be done. In that case the $\alpha\beta$-window is opened and the child node under consideration is regarded as a PV node.

3 Forward Pruning in the Null-Window Search

Recently, it has become practice to use forward-pruning methods only in the NWS part of the PVS framework [8, 11, 17]. The idea is that it is too risky to prune forward at an expected PV node, because a possible mistake causes an immediate change of the principal variation. If we erroneously prune forward at some CUT node and the resulting score is backed up all the way to a PV node, we obtain a fail low (the subtree seemingly has been proved inferior). The result of the mistake is that a possible new principal variation is overlooked in this position. Therefore, forward pruning at a CUT node without further

provisions is dangerous. However, by the large savings achieved it has proven worthwhile to implement further provisions making forward pruning at CUT nodes more safe (and thus acceptable). If we erroneously forward prune at some ALL node and the resulting score is backed up all the way to a PV node, we obtain a fail high and a re-search will be performed. The algorithm is then searching for a new principal variation and regards the subsequent nodes as PV nodes. Because no forward pruning is done at PV nodes, it is possible to find out whether there exists a new PV or not. In this case the result of the mistake will be that an extra amount of nodes has been searched. In principle, forward pruning at an expected ALL node is not dangerous but can trigger unnecessary re-searches.

To ensure that a re-search is performed which is able to correct the value, some small changes in the PVS algorithm have to be made. The following four additions are performed.

1) To prevent that a backed-up value of a forward-pruned ALL node causes a β-cutoff at the PV node lying above, the forward-pruning mechanism should return a value equal to β in case of a cut-off (which is equal to $\alpha + 1$ at an ALL node).

2) If the window of the PV node was already closed and the NWS should return a value equal to β ($\alpha + 1$), we still have to do a re-search.

3) If we do a re-search and the returned value of the NWS equals $\alpha + 1$, we should do a re-search with α as lower bound.

4) CUT nodes where a fail-low has occurred with a value equal to α are not stored in the transposition table because their values are uncertain.

Pseudo code for this enhanced PVS algorithm is given in Figure 2.

4 Multi-cut

Multi-cut pruning is a new forward-pruning method [5]. Before examining an expected CUT node to full depth, the first M child nodes are searched to a depth reduced with a factor R. If at least C child nodes return a value larger than or equal to β, a cutoff occurs. However, if the pruning condition is not satisfied, the search continues as usual, re-exploring the node under consideration to a full depth d. The multi-cut code is given in Figure 3.

In general the behaviour of multi-cut is as follows. The higher M and R are and the lower C is, the higher the number of prunings is.

```
CUT_NODE = 1, ALL_NODE = -1, PV_NODE = 0;

PVS(node, alpha, beta, depth, node_type){
    //Transposition table lookup, omitted
    ..................................
    if(depth == 0)
      return evaluate(pos);
    if(node_type != PV_NODE){
      //Forward-pruning code, omitted
      ..................................
      if(forward_pruning condition holds) return beta; //Addition 1
    }
    next = firstSuccessor(node);
    best = -PVS(next, -beta, -alpha, depth-1, -node_type);
    if(best >= beta) goto Done;
    next = nextSibling(next);
    while(next != null){
      alpha = max(alpha, best);
      //Null-Window Search Part
      value = -PVS(next, -alpha-1, -alpha, depth-1,
                  (node_type == CUT_NODE)?ALL_NODE:CUT_NODE);
      //Re-search
      if((value > alpha && value < beta) ||
        //Addition 2
        (node_type == PV_NODE && value == beta && beta == alpha+1)){
        //Value is not a real lower bound
        if(value == alpha+1) //Addition 3
           value = alpha;
        value = -PVS(next, -beta, -value, depth-1,  node_type);
      }
      if(value > best){
        best = value;
        if(best >= beta) goto Done;
      }
      next = nextSibling(next);
    }
    if(node_type == CUT_NODE && best == alpha) return best; //Addition 4

    Done: //Store in Transposition table, omitted
    ..................................
}
```

Fig. 2. Pseudo code for enhanced PVS

```
.....................................................
//Forward-pruning code
 if(node_type == CUT_NODE && depth > 2){
   next = firstSuccessor(node);
   c = 0, m = 0;
   while(next != null && m < M){
     value = -PVS(next, -beta, -alpha, depth-1-R, -node_type);
     if(value >= beta){
       c++;
       //Keep track of the moves causing a cut-off at d-R
       storeCutOffNode(next);
       if(value >= WIN_SCORE) //Addition 1
         return value;
       else if(c >= C)
         return beta;
     }
     m++;
     next = nextSibling(next);
   }
   //Re-order moves
   putCutOffNodesInFront(); //Addition 2
}
.....................................................
```

Fig. 3. Pseudo code for multi-cut

In our opinion, we made two small improvements in the original algorithm of [5] (indicated in Figure 3). First, when at a reduced depth a winning value is found, the search is stopped and the winning value is returned. Second, if the multi-cut does not succeed in causing a cutoff, the moves causing a β-cutoff at the reduced depth are tried first in the normal search. The remaining question is whether multi-cut is also useful at ALL nodes. The inventors of the multi-cut algorithm anticipated that it would not be successful elsewhere [6] and therefore did not test it at *expected* ALL nodes. Henceforth, we call multi-cut at expected CUT nodes MC-C and at expected ALL nodes MC-A. In the following section, some experiments are reported on testing whether MC-A is beneficial.

5 Experiments

In this section experiments are described with multi-cut at expected ALL nodes. First, we briefly explain the game of Lines of Action (LOA) and the search engine MIA. Next, MC-A is tested under different parameter settings and different combinations of forward-pruning methods are investigated. Sub-

sequently, variable null-move bound is tested and compared to MC-A. Finally, the increase in playing strength is tested.

5.1 Lines Of Action

Lines of Action (LOA) [20] is a two-person zero-sum chess-like connection game with perfect information. It is played on an 8×8 board by two sides, Black and White. Each side has twelve pieces at its disposal. The starting position is given in Figure 4a. The players alternately move a piece, starting with Black. A move takes place in a straight line, exactly as many squares as there are pieces of either colour anywhere along the line of movement (see Figure 4b). A player may jump over its own pieces. A player may not jump over the opponent's pieces, but can capture them by landing on them. The goal of a player is to be the first to create a configuration on the board in which all own pieces are connected in one unit (see Figure 4c). In the case of simultaneous connection, the game is drawn. The connections within the unit may be either orthogonal or diagonal. If a player cannot move, this player has to pass. If a position with the same player to move occurs for the third time, the game is drawn.

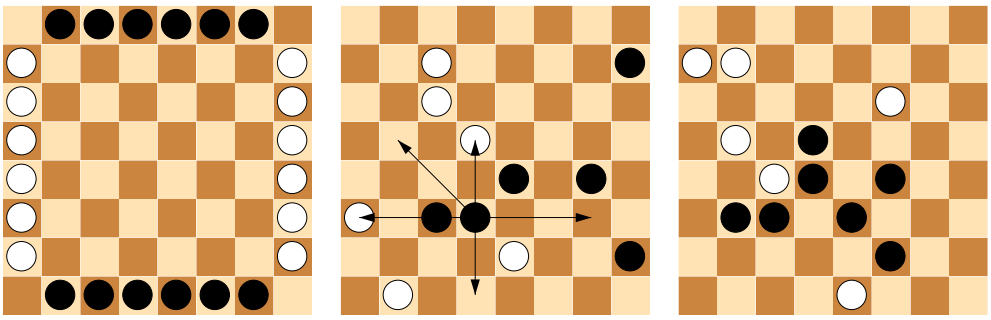

Fig. 4. (a) The initial position of LOA (b) An example of possible moves in a LOA game (c) A terminal LOA position

Recently, LOA is used as domain to test several search techniques [15, 24]. In particular, multi-cut is successfully applied to the game of LOA [4].

5.2 LOA program MIA

All experiments have been performed in the framework of the tournament program MIA (Maastricht In Action).[1] The program has been written in

[1] The program and the test sets can be found at the website:
http://www.cs.unimaas.nl/m.winands/loa/

Java and can easily be ported to all platforms supporting Java.

MIA performs an $\alpha\beta$ depth-first iterative-deepening search in the PVS framework. A *two-deep* transposition table [9] is applied to prune a subtree or to narrow the $\alpha\beta$ window. At all interior nodes which are more than 2 ply away from the leaves, we generate all the moves to perform the Enhanced Transposition Cutoffs (ETC) scheme [22]. Next, a null move [10] is performed before any other move and it is searched to a lower depth (reduced by R) than we would do for other moves. The null move is done at CUT nodes *and* at ALL nodes. At a CUT node a variable scheme, called adaptive null move [13], is used to set R. If the remaining depth is more than 6, R is set to 3. When the number of pieces of the side to move is lower than 5 the remaining depth has to be more than 8 for setting R to 3. In all other cases R is set to 2. For ALL nodes $R = 3$ is used. Then, MC-C is performed with C= 3, $M = 10$ and $R = 3$. For move ordering, the move stored in the transposition table, if applicable, is always tried first. Next, two killer moves [1] are tried. These are the last two moves, which were best or at least caused a pruning at the given depth. Thereafter follow: (1) capture moves going to the inner area (the central 4×4 board) and (2) capture moves going to the middle area (the 6×6 rim). All the other moves are ordered decreasingly according to their scores in the history table [21]. In the leaf nodes of the tree a quiescence search is performed. This quiescence search looks at capture moves, which form or destroy connections [24] and at capture moves going to the central 4×4 board. The evaluation function used is described in [23, 24].

5.3 Parameter Tuning in MC-A

In the following series of experiments we tried to find the parameter setting (C, M, R) of MC-A, which gave the largest node reduction. Using a set of 171 LOA positions, the program was tested at depth 10 using its normal enhancements described in the previous subsection. The following pairs of C and R were investigated: (1,2), (2,2), (3,2), (4,2), (1,3), (2,3), (3,3), (4,3), and (4,4). For each pair the parameter M took the values 2, 4, 6, 8, 10, 12, 14, and 16 except when $M < C$. In the case of $C = 3$ we also investigated $M = 3$. In Figure 5 the total number of nodes searched for each set of parameters is given. The results of the search without MC-A (default) are showed for comparison. In the process of parameter tuning we found that MC-A (2,10,2) is the most efficient. However, the difference with several other parameter configurations is not significant (e.g., (2,6,3)). It is clear that configurations with $C = 1$ have a moderate success. Probably, many re-searches had to be executed because of its aggressive nature (i.e., too much pruning). Configurations with $C = 4$ do not give good results as well, especially not when R and M are chosen too low. The parameter $C = 4$ is too conservative (i.e., not much pruning) and

can be made more aggressive by increasing R and M. It appears that $C = 2$ and $C = 3$ are close to each other if M is sufficiently high.

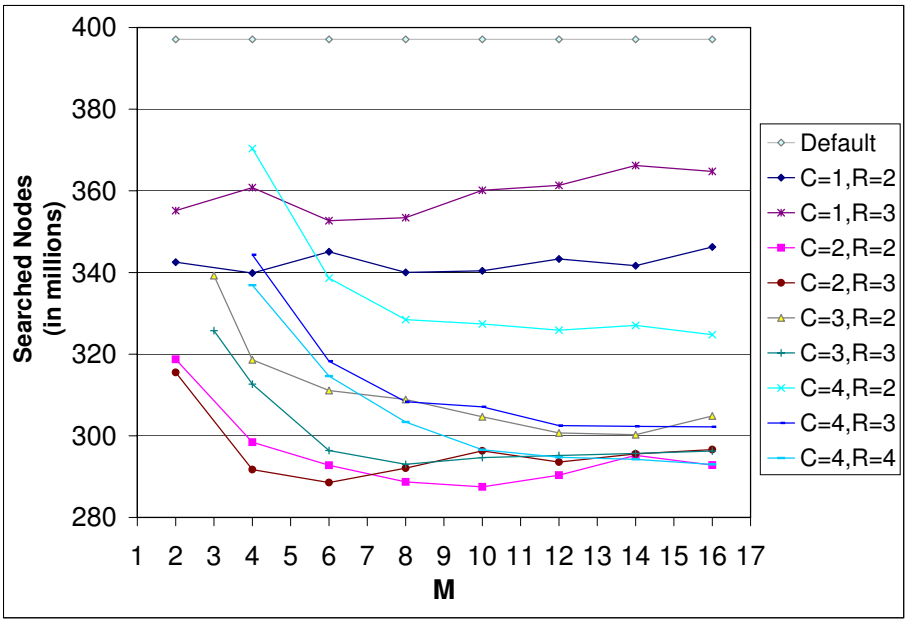

Fig. 5. Tree sizes for different C, M and R

In Figure 5 we see that MC-A with an optimal parameter setting gives a reduction in nodes of 28 per cent compared to the default search engine. If we compare the (aggressive) optimal setting of parameters (i.e., (2,10,2)) to the ones in MC-C (i.e., (3,10,2)) we see a difference of 6 per cent. We would like to remark that MC-C parameters of MIA were already chosen more aggressively than the ones in the LOA-playing program YL (i.e., (3,3,2)) [4]. The last data point gives less reduction according to our experiments (see Figure 5).

A possible enhancement is making the values for C, M, and R variable instead of constant for MC-A. An attractive scheme is to set R to 2 or 3 dependent of the search depth and static board properties (this idea is used in adaptive null-move pruning). Unfortunately, preliminary experiments showed no extra reduction of nodes, so we did not pursue this idea any further. Another idea is to forward prune immediately when the reduced search returns a value greater than $\beta+\delta$ with a sufficiently large δ. Using the optimal parameter setting, the optimal δ value (525) gives a small reduction of 1.5 per cent of nodes searched at depth 10 for the given test set. This heuristic is included in the following experiments. Finally, we remark that if we would not include the re-ordering of moves after an unsuccessful MC-A the number of nodes searched increases with 2 per cent at depth 10.

5.4 Evaluation of MC-A

In the next series of experiments we tested the added value of MC-A, using the optimal parameter setting of the previous subsection, against the same set of 171 positions at several depths in our original search engine. We also looked at the performance of MC-A in combination with the already included pruning methods (i.e., MC-C and null move). The results are given in Table 1.

Table 1
Added value of MC-A

	No Forward pruning			Only Null move		
d	No MC-A	MC-A	%	No MC-A	MC-A	%
5	5,071,689	4,995,845	98.5	3,504,759	3,404,882	97.2
6	19,896,101	19,286,868	96.9	10,109,533	9,518,082	94.1
7	113,653,808	110,663,056	97.4	36,265,257	34,671,647	95.6
8	416,549,038	406,489,302	97.6	92,749,650	89,483,140	96.5
9	2,427,406,280	2,395,844,102	98.7	314,507,126	303,466,596	96.5
10	9,635,185,102	9,460,591,510	98.2	891,348,022	813,032,326	91.2
11	-	-	-	2,930,142,106	2,599,157,486	88.7
12	-	-	-	8,362,297,395	7,080,475,905	84.7
	Only MC-C			Null move and MC-C		
d	No MC-A	MC-A	%	No MC-A	MC-A	%
5	2,097,908	1,955,564	93.2	2,012,835	1,897,600	94.3
6	7,314,731	5,549,772	75.9	6,083,136	5,122,496	84.2
7	28,656,432	20,221,202	70.6	20,491,711	17,423,109	85.0
8	85,103,638	50,333,688	59.1	50,018,470	42,242,144	84.5
9	297,239,554	149,671,128	50.4	142,182,834	116,784,068	82.1
10	1,286,515,396	393,490,307	30.6	397,092,800	283,350,391	71.4
11	4,860,474,957	1,358,658,246	28.0	1,223,918,717	846,066,886	69.1
12	23,806,355,059	3,536,842,482	14.9	3,328,838,963	2,162,692,924	65.0
13	-	-	-	9,869,101,893	6,289,563,990	63.7
14	-	-	-	30,087,791,323	17,578,589,423	58.4

If MC-A is added in a search engine where no forward pruning is used, we see that there is hardly an improvement of performance. The decrease in the

number of nodes searched, is only some 2 per cent. Apparently, when no forward pruning is used at all, multi-cut is useless at expected ALL nodes. This is in accordance with the claim of [6]. However, if some forward-pruning method is used we see that MC-A gives significant improvements. In the case of using only null move adding MC-A gives a further reduction increasing to 15 per cent at depth 12. If only MC-C was used as forward-pruning method, introducing MC-A to the search brings a further reduction increasing to 85 per cent at depth 12. This saving seems enormous but we have to consider that in the previous experiment null-move pruning was already performed at ALL nodes. In the case of using only MC-C there is no forward pruning at expected ALL nodes, which therefore results in a large saving when MC-A is added. We remark that the tree sizes of the combination MC-C and MC-A are approximately the same as the combination null move and MC-C. When null move and MC-C are available in the engine (the original situation), MC-A reduces the search tree with another 42 per cent at depth 14.

Since we have tuned MC-A at this particular test set (subsection 5.3), we performed similar experiments on a set consisting of 156 different positions to validate the result. In the first column of Table 2 we see the relative performance of MC-A on the the original test set. In the second column the relative performance of MC-A is given for the validation set. If we compare those two columns with each other, we see that there is not much difference between them and similar results are achieved.

Table 2
Relative performance of MC-A in combination with null move and MC-C.

Depth	Original Set (171 positions)	Validation Set (156 positions)
5	94.3	94.7
6	84.2	86.8
7	85.0	83.7
8	84.5	82.0
9	82.1	79.8
10	71.4	73.9
11	69.1	72.0
12	65.0	68.5
13	63.7	64.8
14	58.4	61.4

In summary, our experiments have showed that if forward pruning is already used additional forward pruning at ALL nodes gives a significant improve-

ment. Of course, it is possible that another forward-pruning mechanism might achieve the same results. For example, making the null-move mechanism at ALL nodes more aggressive is a possibility to gain reductions. In our experiments we have made the null move more aggressive by setting the reduction factor at 3 when searching an expected ALL node. In the next subsection we test the variable null-move bound.

5.5 Variable null-move bound

Campbell and Goetsch [12] introduced the idea of variable null-move bound, which was implemented and tested by Björnsson and Marsland [7]. The idea is that a null-move cutoff can be forced if the returned null-move search value is larger than or equal to $\beta - t$, where t is the minimal value of a tempo. The value t is dependent on the evaluation function. This allows a larger part of the null-move searches to cause cut-offs. The pseudo code is given in Figure 6. We remark that the null move is considered as a regular move. Therefore, if a null move originates from a CUT node, its successor is an ALL node, and *vice versa*.

```
.....................................................
//Forward-pruning code
 if(node_type != PV_NODE && depth > 2){
   next = swapSides(node);
   if(node_type == ALL_NODE) bound = beta - t else bound = beta;
   value = -PVS(next, -bound, -bound+1, depth-1-R, -node_type);
   if(value >= bound) return beta;
 }
.....................................................
```

Fig. 6. Pseudo code for variable null-move bound

To compare variable null-move bound (VNMB) with MC-A we investigated in the following series of experiments whether VNMB at expected ALL nodes gives a reduction of nodes. The heuristic was tested against the same set of 171 positions with 10-ply searches. The search was not enhanced with MC-A. The parameter t was increased with a step size of 25 to find a value which gives the most reduction. The results are given in Figure 7. For comparison the results of the default search and the search enhanced with MC-A are also given.

We see that the best result for t is achieved when t lies between 200 and 350. A reduction of 16 per cent is achieved compared to the original one (of course with the null-move and MC-C). But the search enhanced with MC-A is still 15 per cent faster. When t is larger than 500, the search enhanced

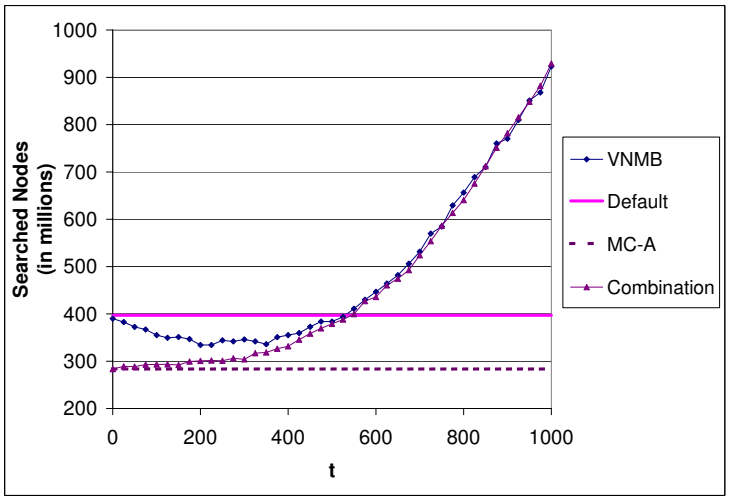

Fig. 7. Variable null-move bound

with variable null-move bound searches more nodes than the original one. The forward pruning becomes too aggressive resulting in too many re-searches.

In the next experiment the combination of MC-A and variable null-move was tested with different t. There was no setting of t for which the combination was better than the MC-A enhanced search. When t became sufficiently large enough the combination behaved the same as the variable null-move. In Figure 8 the relative performance of MC-A is compared to variable null-move bound with the optimal parameter $t = 200$ for several search depths. We see that the search enhanced with MC-A searches fewer nodes than the one enhanced with VNMB. This difference increases with depth to some 22 per cent.

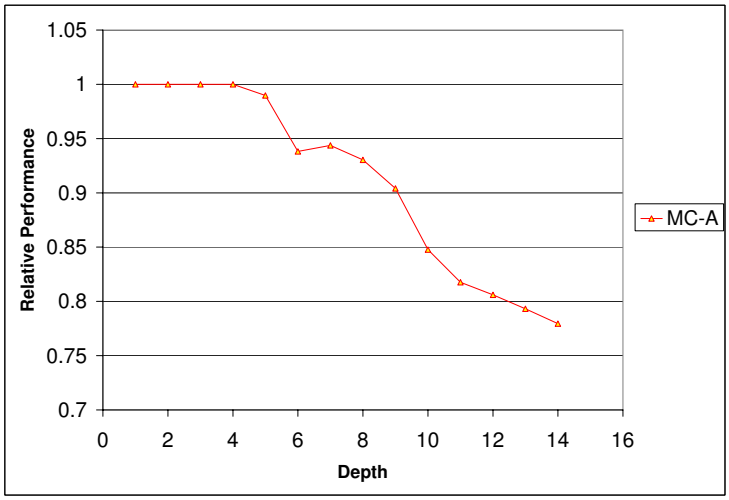

Fig. 8. MC-A compared to variable null-move bound

These experiments suggest that MC-A is not surpassed by aggressive null-move pruning at expected ALL nodes. We believe that MC-A is a significant enhancement in a PVS search where forward pruning is used. In general these

experiments suggest that extra forward pruning at ALL nodes gives a safe reduction of the number of nodes searched.

5.6 Performance enhancement with MC-A

In the last experiment we tested the possible increase of playing strength by using MC-A with the optimal parameter setting (of subsection 5.3). Two versions of MIA were matched against each other, one using MC-A and the other without MC-A. These versions used all the enhancements described in subsection 5.2. Both programs played 1000 games, switching sides halfway. The thinking time was limited to 60 seconds per move, simulating tournament conditions. To prevent that programs played the games over and over again, a random factor was included in the evaluation function. The results are given in Table 3.

Table 3
1000-game match results.

	Score	Winning ratio
MC-A vs. Default	549-451	1.21

We observe that the modified version outplayed the original version with a winning ratio of 1.21 (i.e., scoring 21% more winning points than the opponent). This result reveals that MC-A improves the playing strength of MIA significantly.

6 Conclusions and Future Research

This paper showed that forward pruning at expected ALL nodes is safe and beneficial. Multi-cut at expected ALL nodes gives a safe reduction of approximately 40 per cent of the number of nodes searched in combination with null move and the regular multi-cut MC-C. Experiments suggested that parameters more aggressively chosen than MC-C lead to an additional improvement. We observed that MC-A still searches 22 per cent less nodes than variable null-move bound at expected ALL nodes. Moreover, MC-A was able to increase significantly the playing strength of the program MIA.

As future research, experiments are envisaged in other games to test the performance of MC-A. Whether MC-A surpasses an aggressive version of null move in other games also has to be tested. Finally, the combination of MC-A and variable null-move bound has to be tuned with different settings of C, M, R and t.

Acknowledgements

The authors would like to thank the members of the Maastricht Search & Games Group and the referees for their valuable comments. We gratefully acknowledge financial support by the Universiteitsfonds Limburg / SWOL.

References

[1] S.G. Akl and M.M. Newborn. The principal continuation and the killer heuristic. In *1977 ACM Annual Conference Proceedings*, pages 466–473. ACM, Seattle, 1977.

[2] T.S. Anantharaman, M. Campbell, and F.-h. Hsu. Singular extensions: Adding selectivity to brute-force searching. *ICCA Journal*, 11(4):135–143, 1988. Also published in *Artificial Intelligence*, 43(1):99-109, 1990.

[3] D.F. Beal. Experiments with the null move. In D.F. Beal, editor, *Advances in Computer Chess 5*, pages 65–79. Elsevier Science Publishers, Amsterdam, The Netherlands, 1989.

[4] Y. Björnsson. *Selective Depth-First Game-Tree Search.* PhD thesis, University of Alberta, Edmonton, Canada, 2002.

[5] Y. Björnsson and T.A. Marsland. Multi-cut alpha-beta pruning. In H.J. van den Herik and H. Iida, editors, *Computers and Games, Lecture Notes in Computing Science 1558*, pages 15–24. Springer-Verlag, Heidelberg, Germany, 1999.

[6] Y. Björnsson and T.A. Marsland. Multi-cut alpha-beta pruning in game-tree search. *Theoretical Computer Science*, 252(1-2):177–196, 2001.

[7] Y. Björnsson and T.A. Marsland. Risk managament in game-tree pruning. *Information Sciences*, 122(1):23–41, 2001.

[8] Y. Björnsson, T.A. Marsland, J. Schaeffer, and A. Junghanns. Searching with uncertainty cut-offs. *ICCA Journal*, 20(1):29–37, 1997.

[9] D.M. Breuker, J.W.H.M. Uiterwijk, and H.J. van den Herik. Replacement schemes and two-level tables. *ICCA Journal*, 19(3):175–180, 1996.

[10] C. Donninger. Null move and deep search: Selective-search heuristics for obtuse chess programs. *ICCA Journal*, 16(3):137–143, 1993.

[11] R. Feldmann. Fail-high reductions. In H.J. van den Herik and J.W.H.M Uiterwijk, editors, *Advances in Computer Chess 8*, pages 111–127. Universiteit Maastricht, Maastricht, The Netherlands, 1997.

[12] G. Goetsch and M.S. Campbell. Experiments with the null-move heuristic. In T.A. Marsland and J. Schaeffer, editors, *Computers, Chess, and Cognition*, pages 159–168. Springer-Verlag, New York, 1990.

[13] E.A. Heinz. Adaptive null-move pruning. *ICCA Journal*, 22(3):123–132, 1999.

[14] D.E. Knuth and R.W. Moore. An analysis of alpha-beta pruning. *Artificial Intelligence*, 6(4):293–326, 1975.

[15] L. Kocsis, J.W.H.M. Uiterwijk, and H.J. van den Herik. Move ordering using neural networks. In L. Montosori, J. Váncza, and M. Ali, editors, *Engineering*

of Intelligent Systems, Lecture Notes in Artificial Intelligence, Vol. 2070, pages 45–50. Springer-Verlag, Berlin Heidelberg, 2001.

[16] T.A. Marsland. Relative efficiency of alpha-beta implementations. In *Proceedings of the 8th International Joint Conference on Artificial Intelligence (IJCAI-83)*, pages 763–766. Karlsruhe, Germany, 1983.

[17] T.A. Marsland and Y. Björnsson. Variable-depth search. In H.J. van den Herik and B. Monien, editors, *Advances in Computer Games 9*, pages 9–24. Universiteit Maastricht, Maastricht, The Netherlands, 2001.

[18] T.A. Marsland and F. Popowich. Parallel game-tree search. *IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI*, 7(4):442–452, 1985.

[19] A. Reinefeld. An improvement to the Scout search tree algorithm. *ICCA Journal*, 6(4):4–14, 1983.

[20] S. Sackson. *A Gamut of Games*. Random House, New York, NY, USA, 1969.

[21] J. Schaeffer. The history heuristic. *ICCA Journal*, 6(3):16–19, 1983.

[22] J. Schaeffer and A. Plaat. New advances in alpha-beta searching. In *Proceedings of the 1996 ACM 24th annual conference on Computer science*, pages 124–130. ACM Press, New York, NY, USA, 1996.

[23] M.H.M. Winands, L. Kocsis, J.W.H.M. Uiterwijk, and H.J. van den Herik. Temporal difference learning and the neural movemap heuristic in the game of Lines of Action. In Quasim Mehdi, Norman Gough, and Marc Cavazza, editors, *GAME-ON 2002 3rd International Conference on Intelligent Games and Simulation*, pages 99–103. SCS Europe Bvba, 2002.

[24] M.H.M. Winands, J.W.H.M. Uiterwijk, and H.J. van den Herik. The quad heuristic in Lines of Action. *ICGA Journal*, 24(1):3–15, 2001.