RankCut – A Domain Independent Forward Pruning Method for Games

Yew Jin Lim and **Wee Sun Lee**
School of Computing
National University of Singapore
{limyewji,leews}@comp.nus.edu.sg

Abstract

Forward pruning, also known as selective search, is now employed in many strong game-playing programs. In this paper, we introduce RankCut – a domain independent forward pruning technique which makes use of move ordering, and prunes once no better move is likely to be available. Since game-playing programs already perform move ordering to improve the performance of $\alpha\beta$ search, this information is available at no extra cost. As RankCut uses additional information untapped by current forward pruning techniques, RankCut is a complementary forward pruning method that can be used with existing methods, and is able to achieve improvements even when conventional pruning techniques are simultaneously employed. We implemented RankCut in a modern open-source chess program, CRAFTY. RankCut reduces the game-tree size by approximately 10%-40% for search depths 8-12 while retaining tactical reliability, when implemented alongside CRAFTY's existing forward pruning techniques.

Introduction

Alpha-Beta ($\alpha\beta$) pruning (Knuth & Moore 1975) and its variants like NegaScout/Principal Variation Search (Reinefeld 1989) and MTD(f) (Plaat 1996) have become the standard methods used to search game-trees as they greatly reduce the search effort needed. Nevertheless, search effort increases exponentially with increasing search depth and, apart from theoretical exceptions where decision quality decreases with search depth (Beal 1980; Nau 1979), it is generally accepted that searching deeper will result in higher move decision quality (Junghanns *et al.* 1997). To further reduce the number of nodes searched, game-playing programs perform forward pruning (Marsland 1986), where a node is discarded without searching beyond that node if it is believed that the node is unlikely to affect the final minimax value of the node.

In this paper, we propose a new method, RankCut, which estimates the probability of discovering a better move later in the search by using the relative frequency of such cases for various states during search. These probabilities are precomputed off-line using several self-play games. RankCut can then reduce search effort by performing a shallow search when the probability of a better move appearing is below a certain threshold. RankCut implicitly requires good move ordering to work well. However, game-playing programs already perform move ordering as it improves the efficiency of $\alpha\beta$ search, and thus good move ordering is available at no performance cost. We implemented RankCut in the open source chess program, CRAFTY, and show its effectiveness with test suites and matches.

Background

We assume that the reader is familiar with $\alpha\beta$ search and its relationship to the minimax algorithm. However, we recall some properties of the algorithms and several definitions which will be relevant to this paper. $\alpha\beta$ search reduces, on average, the effective branching factor by two compared to the minimax algorithm, and does better when the nodes are evaluated in an optimal or near optimal order (Knuth & Moore 1975). A reduced-depth search is a search with depth $d - r$, where d is the original search depth and r is the amount of depth reduction. The higher the value of r, the more errors the reduced-depth search makes due to the horizon effect (Berliner 1974). However, a reduced-depth search will result in a smaller search tree and therefore a compromise has to made between reduced search effort and the risk of making more search errors.

Although it is technically more correct to refer to the value of a node (and its corresponding position) in the minimax framework, we will use the "value of a move" to refer to the value of the node as result of that move as it is conceptually easier to describe how to use move order information during search in this manner.

Current Forward Pruning Techniques

We briefly describe current forward pruning techniques in this section for the purpose of showing how they are based on a similar principle. For more extensive readings, we suggest (Marsland 1986; Buro 1999; Heinz 2000; Björnsson & Marsland 2000b; Tabibi & Netanyahu 2002).

Null Move Pruning Null-move pruning (Beal 1989; Goetsch & Campbell 1990; Donninger 1993), which was introduced in the 1990s, enables programs to search deeper with little tactical risk. Null move pruning assumes that not making any move and passing (also known as making a *null*

move), even if passing is illegal in the game, is always bad for the player to move. So when searching to depth d, the heuristic makes a null move by simply changing the turn to move to the opponent and performs a reduced-depth search. If the reduced-depth search returns a score greater than β, then the node is likely to be a strong position for the player to move and will have a score greater than β when searched to depth d without the null move. The program should therefore fail-high and return β.

However, there is a class of positions known as *zugzwang* positions where making the null move is actually good for the player to move. This violates the assumption that null-move pruning makes and causes search errors. As a result, null-move pruning is generally avoided in endgame positions.

ProbCut The ProbCut heuristic (Buro 1995) prunes nodes which are likely to fail outside the $\alpha\beta$ bounds during search by using simple statistical tools. ProbCut first requires off-line computations to record results of both shallow and deep searches of the same positions. The results of both searches are then correlated using linear regression. ProbCut is therefore able to form a confidence interval from the result of a shallow search to obtain an estimated range in which a deep search will return. If this confidence interval falls outside the $\alpha\beta$ bounds, the program should fail high or low appropriately. ProbCut was successfully applied in Logistello, an Othello program that beat the then World Human Othello Champion 6-0 under standard time controls (Buro 1997).

Multi-ProbCut (Buro 1999) is an enhancement of ProbCut which uses different regression parameters and pruning thresholds for different stages of the game and multiple depth pairings. Preliminary experiments show that Multi-ProbCut can be successfully applied to chess (Jiang & Buro 2003), but requires a fair amount of work to get good results.

Futility Pruning Futility pruning applies to more restrictive situations but has been successfully used in chess programs. This heuristic was first introduced by Schaeffer (Schaeffer 1986) and is based on the observation that evaluation functions can usually be separated into major and minor components. For example, in chess, the major component would include material count and the minor component would include positional evaluations. As the minor component of the evaluation is composed solely of smaller adjustments to the score, it can typically be bounded. At frontier nodes, or nodes that are 1-ply from the leaf nodes, if the evaluation of major components falls greatly outside $\alpha\beta$ bounds and the bounds on the evaluation of the minor component cannot possibly adjust the score to within the $\alpha\beta$ bound, the frontier node can be safely pruned.

Heinz improved on futility pruning by applying futility pruning to pre- and pre-pre-frontier nodes and called it "Extended Futility Pruning" (Heinz 1998b). Experiments with well-known test suites and the chess program DARKTHOUGHT (Heinz 1998a) showed improvements of $10\% - 30\%$ in fixed search depths of 8-12 plies.

Is There Anything Better?

Most forward pruning techniques can be seen as using the same principle of locally testing the probability of a node scoring outside the α or β bound, and pruning the nodes which have a high probability of failing high or low. In other words, for a node n with k moves, $m_1 \dots m_k$, current pruning techniques test $p(v(m_i) \geq \beta)$ and $p(v(m_i) \leq \alpha)$, where $v(m_i)$ is the minimax evaluation of the board position after making move i, and prune if either of the probabilities is high. For example, the null move heuristic decides to prune m_k if the search returns a score greater than β after making a null move, as it assumes that doing nothing is generally bad for the player. This is equivalent to saying that $p(v(m_k) \geq \beta)$ is high and the program can prune the node while taking little risk. ProbCut extends this by also considering that if $p(v(m_k) \leq \alpha)$ is high, the program should fail low and return α. However, the fact that m_k is the k^{th} move considered is not used by existing techniques.

Information on move ordering can be beneficial when making forward pruning decisions. For example, consider the same scenario where we are deciding whether or not to prune m_k, and the current best move has remained the first move m_1, while $v(m_1) > v(m_2) > v(m_3) \dots > v(m_{k-2}) > v(m_{k-1})$. Furthermore, if we know that, under similar conditions in representative games, no moves ranked k or higher have ever been found to be better, then it is unlikely that m_k will return a score better than the score of the current best move. This means m_k can be be pruned with a small risk of error based on move order information.

One pruning method that utilizes ordering is N-Best Selective Search. In N-Best Selective Search, only the N best moves, as ranked by an evaluation function, are actually searched. This heuristic requires the move ordering to be able to rank the best move consistently within the top N moves. However, the simple N-best selective search either introduces too many errors or prunes too little, depending on how N is set. For example, consider a node where the best move changes every time a new move is searched. Evidently, move ordering is not performing well and pruning after a small number of moves is likely to introduce errors. One easy, but wrong, solution would be to increase N to a high value where such occurrences are rare. However if we now consider a node where the best move has not changed since the first move, and the scores of the subsequent moves are decreasing in a monotonic fashion, then it is likely that a high value of N is too conservative for this particular case, and forward pruning opportunities have been lost.

A better way of factoring in move order when pruning is by considering the probability $\Pi_x(\vec{f_i}) = p(v(m_i) > x \mid \vec{f_i})$, where $\vec{f_i} = (f(m_1), \dots, f(m_{i-1}))$ are the salient features of the previous moves considered, and x is an adaptive bound. In our experiments, x is set to v_{best}, the score of the current best move. However, x can also be set to a value like α, or a variable bound so as to minimize the risk of error. For brevity, we use $\Pi(\vec{f_i})$ to represent $\Pi_{v_{best}}(\vec{f_i})$. Good features allow $\Pi(\vec{f_i})$ to identify when moves are unlikely to affect the final score, and examples include the current move number and the scores of prior moves. So when performing

forward pruning, the probability $\Pi(\vec{f_i})$ gives the likelihood of m_i returning a score better than the current best move, and if it is below a certain threshold, we can prune this move as it is unlikely to affect the final score. This approach is more adaptive than the static N-best selection search.

Since good move ordering is essential to achieving more cutoffs when using $\alpha\beta$ search (Knuth & Moore 1975), heuristics like the History Heuristic (Schaeffer 1989) and the Killer Heuristic (Akl & Newborn 1977) together with domain dependent knowledge are used to improve move ordering. Good move ordering is therefore usually available in practical implementations, and is another good reason to consider move order in any forward pruning decision. This observation is not new – Björnsson and Marsland (Björnsson & Marsland 2000a) mention this insight, but still restrict the application to only the possibility of failing high, or $p(v(m_i) > \beta \mid v(m_1) < \beta, \ldots, v(m_{i-1}) < \beta)$. Moriarty and Miikkulainen (Moriarty & Miikkulainen 1994) and Kocsis (Kocsis 2003, Chapter 4) also considered pruning nodes while taking move ordering into consideration by using machine learning techniques like neural networks to estimate the probability. However, the results of these experiments have not been conclusive and more research in their effectiveness is needed.

RankCut

In this section, we introduce a new domain independent forward pruning method called RankCut and show its effectiveness by implementing it in an open source chess program, CRAFTY[1].

Concept

In the previous section we suggested considering the probability $\Pi(\vec{f_i})$ when forward pruning. However, if the moves are ordered and $\Pi(\vec{f_i})$ is low, then the remaining moves m_j, where $j > i$, should also have low probabilities $\Pi(\vec{f_j})$. Testing each probability $\Pi(\vec{f_i})$ is thus often redundant and RankCut considers instead the value of $\Pi'(\vec{f_i}) = p(\max\{v(m_i), v(m_{i+1}), \ldots, v(m_k)\} > v_{best} \mid \vec{f_i})$, where k is the total number of legal moves of the current node. RankCut can be thought of as asking the question "Is it likely that any remaining move is better than the current best move?". These probabilities are estimated off-line by using the relative frequency of a better move appearing and can be represented by x/y where x is the number of times a move m_j, where $j \geq i$, returns a score better than the current best when in the state $\vec{f_i}$ and y is the total number of instances of the state $\vec{f_i}$, regardless of whether the best move changes. This off-line procedure of collecting the statistics requires modifying the game-playing program to store the counts, and then playing a number of games under the same (or longer) time controls expected.

RankCut tests $\Pi'(f_{m_i}) < t$ for each move, where $t \in (0, 1)$ is user-defined. If true, RankCut does not prune the move but instead does a reduced-depth search and returns the score of that shallow search. The full-width nature of the reduced-depth search helps to retain tactical reliability while reducing search effort. RankCut is domain independent as it does not require any game logic and is easily added to an $\alpha\beta$ search as shown in Pseudocode 1.

Pseudocode 1 RankCut(α, β, depth)

```
if depth == 0 then
  return LeafEvaluate()
pruneRest ← false
for move ← NextMove() do
  r ← 0
  Compute(f_i)
  if (pruneRest || Π'(f_i) < t) then
    r ← DepthReduction()
    pruneRest ← true
  score ← −RankCut(−β, −α, depth − 1 − r)
  if α < score then
    pruneRest ← false
    α ← score
  if score ≥ β then
    return score
return score
```

The main modifications to the $\alpha\beta$ algorithm are the test of $\Pi'(\vec{f_i})$ and computations of $\vec{f_i}$. Prior to making a move, RankCut tests $\Pi'(\vec{f_i}) < t$ and if true, makes a reduced-depth search. Otherwise, the usual $\alpha\beta$ algorithm is executed.

One potential problem is that RankCut assumes the statistics of $\Pi(\vec{f_i})$ collected without forward pruning remain the same when forward pruning. While our experiments indicate that this assumption is reasonable for practical purposes, one possible solution is to recollect the statistics with forward pruning until the probabilities stabilize. However, more experiments are needed to verify this approach.

Implementation in CRAFTY

CRAFTY is a very strong open-source chess engine and its rating is about 2600 on a 1.2 GHz Athlon with 256MB of RAM when tested independently by the Swedish Chess Computer Association, or SSDF[2]. CRAFTY uses modern computer chess techniques such as bitboards, 64-bit data structures, negascout search, killer move heuristics, static exchange evaluation, quiescence search, and selective extensions. We incorporated RankCut into CRAFTY Version 19.19, which features null move pruning and futility pruning, and ran all experiments on a PowerMac 1.8GHz. We will differentiate between the two versions of CRAFTY when needed in our discussions by calling them ORIGINAL CRAFTY and RANKCUT CRAFTY.

CRAFTY has 5 sequential phases of move generation – (1) Principal variation from the previous search depth during iterative deepening, (2) capture moves sorted based on the expected gain of material, (3) Killer moves, (4) at most 3 History moves, and (5) the rest of the moves. We modified CRAFTY so that at phase 4 or the History move phase, it

[1] Available at ftp://ftp.cis.uab.edu/pub/hyatt

[2] Details at http://web.telia.com/∼u85924109/ssdf/

would continue sorting remaining moves according to the History heuristic until no more suitable candidate was found.

During testing, we discovered that the probability of a better move appearing for moves generated in phases before the History Move phase is always too high to trigger a pruning decision. RANKCUT CRAFTY saves computational effort by only starting to forward prune when move generation is in History move phase.

The probabilities $\Pi'(\vec{f_i})$ were calculated by collecting the statistics from 50 self-play games, each with a randomly-chosen opening, where CRAFTY played against itself in a time control of 80 minutes per 40 moves. The pruning threshold t and the amount of depth reduction in the shallow search were conservatively set at 0.75% and 1 ply, respectively. As we calculated the relative frequencies with a small set of 50 games, we use the probabilities $\Pi'(\vec{f_i})$ only if 1,000 or more instances of $\vec{f_i}$ were seen to ensure that the statistics are reliable. The following features $\vec{f_i}$ were used:

1. Current depth of search
2. Whether or not player is in check
3. Current move number
4. Number of times the best move has changed
5. Difference between the score of the current best move from the given α bound (discretised to 7 intervals)
6. Difference between the score of the last move from the current best move (discretised to 7 intervals)
7. Phase of the move generation (History Moves or Remaining Moves)

CRAFTY uses search extensions to explore promising positions more deeply. RANKCUT CRAFTY therefore does not reduce the search depth even when $\Pi'(\vec{f_i}) < t$ if CRAFTY extends the search depth as CRAFTY has signaled that the current node needs more examination. RANKCUT CRAFTY is also set to forward prune only nodes that have search depth, defined as the length of a path from the current node to the leaf nodes, greater or equal to 7. This is because move ordering tends to be less reliable when iterative deepening is initially searching the first few depths. Furthermore, when searching higher search depths, move ordering also becomes less reliable when the search is further from the root.

Test Suites We tested RANKCUT CRAFTY with all 2,180 positions from the tactical chess test suites ECM, WAC and WCS (See Appendix) by searching to fixed depths of 8, 10, and 12 plies respectively. Table 1 shows the results of these results and compares them with the results of the ORIGINAL CRAFTY. The last three rows of Table 1 also show the combined results of the three test suites.

The absolute standard error (Heinz 1999a) of n test positions with k correct solutions is $SE = n \times \sqrt{p \times (1-p)/n}$ where $p = k/n$. The standard error allows us to ascertain whether the errors introduced by the pruning method is within 'statistical' error bounds. The combined results of all three suites for ORIGINAL CRAFTY in Table 1 shows that the standard error for "Sum-08", "Sum-10" and "Sum-12" are $SE_8 = 19.6$, $SE_{10} = 18.1$ and $SE_{12} = 16.7$, respectively. RANKCUT CRAFTY, however, solves only 3, 6 and 16 fewer test positions while searching less nodes (with the difference from ORIGINAL CRAFTY denoted by Δ columns). Hence all the results of RANKCUT CRAFTY are within one standard error of the results of the results of ORIGINAL CRAFTY.

We also tested using the LCT II test (See Appendix), a set of 35 positions divided into positional, tactical and end-game positions. The LCT II estimates an ELO rating for the program based on the solution times. Both ORIGINAL CRAFTY and RANKCUT CRAFTY solved the same 28 out of 35 problems, but due to faster solutions, RANKCUT CRAFTY obtained a rating 2635 ELO whereas ORIGINAL CRAFTY was estimated at 2575 ELO. During the test, ORIGINAL CRAFTY searched to an average of 15.7 plies whereas RANKCUT CRAFTY was able to search to an average of 16.5 plies, almost 1 ply deeper on average.

Match Games against Original We played RANKCUT CRAFTY against ORIGINAL CRAFTY with a set of 62-openings, consisting of 32 openings used in (Jiang & Buro 2003), 10 Nunn positions, and 20 Nunn-II positions (See Appendix) under the time control of 40 moves/40 minutes. This 124-game match resulted in a decisive result of +40 -10 =74 or 62.0% in favor of RANKCUT CRAFTY. We will outline the statistical tools used in (Heinz 1999b) to show that this result is statistically significant.

The standard error of a scoring rate $w = x/n$ is $s(w) = \sqrt{w \times (1-w)/n}$, where $x \leq n$ is the number of points scored in a match of n games. Let $z_{\%}$ denote the upper critical value of the standard $N(0,1)$ normal distribution for a desired $\%$-level statistical confidence, where $z_{90\%} = 1.645$ and $z_{95\%} = 1.96$. Then $w \pm z_{\%} \times s(w)$ is the $\%$-level confident interval on the real winning probability of a player with scoring rate w.

We can now derive the $\%$-level confident lower bound on the difference in real winning probability between two players of scoring rates $w_1 = x_1/n_1$, and $w_2 = x_2/n_2$, where $w_1 \geq w_2$. Let $l_{\%} = (w_1 - z_{\%} \times s(w_1)) - (w_2 + z_{\%} \times s(w_2))$. If $l_{\%} > 0$, we are $\%$-level confident that the player with the higher scoring rate is indeed stronger than the other.

From above, $l_{95\%} \approx 0.099$ and therefore we can claim that RANKCUT CRAFTY is indeed stronger than ORIGINAL CRAFTY with 95% confidence.

Match Games against FRUIT 2.1 FRUIT is one of the strongest chess programs in the world. FRUIT 2.2.1 finished in second place at the 2005 World Computer Chess Championships (Björnsson & van den Herik 2005) and obtained a rating of more than 2800 when tested by SSDF. FRUIT was an open-source chess engine until Version 2.1[3]. Due to a lack of time, we could only test the ORIGINAL CRAFTY and RANKCUT CRAFTY against FRUIT 2.1 with the 32-openings used in (Jiang & Buro 2003) under blitz time controls of 2 min + 10 sec/move. ORIGINAL CRAFTY lost to Fruit by +11 -39 =14 or 28.1% and RANKCUT CRAFTY lost to Fruit by +15 -40 =9 or 30.5%. We are unable to draw any conclusion based on such a small sample of games

[3]Source code is available at http://arctrix.com/nas/fruit/

	Original			RankCut				
Test Suite	#Nodes	Avg Time (s)	#Solved	ΔNodes	Δ%	ΔTime	Δ%	ΔSolved
ECM-08	1,170,566,012	1.68	549	-104,561,591	-8.9%	-0.03	-1.98%	-2
ECM-10	11,641,894,779	15.43	620	-3,636,256,257	-31.2%	-3.98	-25.80%	-7
ECM-12	88,924,232,803	120.92	678	-36,761,238,388	-41.3%	-47.34	-39.15%	-13
WAC-08	160,052,139	0.63	289	-11,554,559	-7.2%	-0.02	-2.87%	-2
WAC-10	1,961,241,110	6.85	294	-667,242,604	-34.0%	-1.26	-18.32%	0
WAC-12	13,361,000,868	66.92	297	-4,763,207,469	-35.7%	-35.95	-53.72%	-1
WCS-08	914,031,896	1.11	840	-96,470,205	-10.6%	-0.06	-4.96%	1
WCS-10	9,534,443,036	10.75	863	-2,234,703,349	-23.4%	-2.32	-21.60%	1
WCS-12	77,536,489,398	87.36	873	-212,586,355	-36.1%	-27.36	-31.31%	-2
Sum-08	2,244,650,047	1.27	1678	-212,586,355	-9.5%	-0.04	-3.23%	-3
Sum-10	23,137,578,925	12.10	1777	-6,538,202,210	-28.3%	-2.84	-23.50%	-6
Sum-12	179,821,723,069	98.08	1848	-69,528,669,092	-38.7%	-36.60	-37.31%	-16

Table 1: Comparison of performance in test suites with fixed depths

under blitz time controls, but the results suggests that the performance gains of RANKCUT CRAFTY extend to games against other chess engines.

Discussion

The results on the test suites and match against ORIGINAL CRAFTY provide strong evidence that RankCut improves the performance of CRAFTY. This is significant as CRAFTY already implements null-move pruning and futility pruning.

We have done some preliminary testing to compare the performance of Multi-ProbCut (MPC) in CRAFTY (Jiang & Buro 2003) to RANKCUT CRAFTY, and we are able to show that RANKCUT CRAFTY outperforms MPC CRAFTY in a 64-game match with a 95%-level confidence. However, as MPC CRAFTY was based on CRAFTY 18.18 and RANKCUT CRAFTY is based on CRAFTY 19.19, we are unable to draw the conclusion that RankCut is a better forward-pruning method than Multi-ProbCut in chess. While this research is necessary to ascertain the effectiveness of each forward pruning method, we believe that RankCut complements existing forward pruning methods and should be viewed as a technique to allow game-playing programs to factor in move order when forward pruning.

RankCut is able to identify when moves generated beyond a certain point are not likely to affect the final score, and therefore we believe that RankCut will benefit game-playing programs most in games with large branching factor and where good or bad moves are easily identifiable. One candidate is the game of Go where the number of legal moves ranges from 100–360 for the opening phase and most of the middle-game. Another example is the game of Abalone, a two-player game invented in 1990 by Laurent Levi and Michel Lalet. Abalone has an average branching factor of 80 (Aichholzer, Aurenhammer, & Werner 2002) which places the game-tree complexity of Abalone between those of chess and Go.

We have done some initial experiments in the game of Abalone by training a neural network as an leaf evaluation function and implemented RankCut together with $\alpha\beta$ search. Due to the large branching factor of Abalone, $\alpha\beta$ search without forward pruning could only search up to an average of 4 plies in 1 minute, but is able to search up to an average of 8 plies in 1 minute with RankCut. Under fixed time limits of 1 minute per move, $\alpha\beta$ search with RankCut was able to beat the commercial version of ABA-PRO (Aichholzer, Aurenhammer, & Werner 2002), arguably the strongest Abalone-playing entity, at a fixed playing level of 9 by +12 -5 =3 in a 20-game series, whereas $\alpha\beta$ search without RankCut lost handily by +2 -15 =5 to ABA-PRO. However, as this match was not under tournament conditions, we are unable to draw any concrete conclusion on the playing strength of our Abalone program.

Conclusion

In this paper, we introduce RankCut, a domain-independent forward pruning technique that exploits the move ordering that current game-playing programs typically perform for efficiency in $\alpha\beta$ search. RankCut can be implemented within an existing $\alpha\beta$ search, and we successfully implemented RankCut in CRAFTY, an open-source chess-playing program. Compared to the unmodified version of CRAFTY, RankCut reduces the game-tree size by 10%-40% for search depths 8-12 while retaining tactical reliability. After playing a 124-game match against the original CRAFTY, RankCut CRAFTY had a winning percentage of 62%. This is despite the fact that CRAFTY also features null-move pruning and futility pruning. The simplicity of RankCut makes implementation in various games easy, and it can even be implemented on top of existing forward pruning techniques.

Acknowledgement

This work is supported in part by an A*Star-NUS Graduate Fellowship to Yew Jin Lim. We thank Jürg Nievergelt, Oon Wee Chong and the anonymous referees for their helpful comments.

Appendix – Experimental Setup

- Test suites "Encyclopedia of Chess Middlegames" (ECM, 879 positions), "Win at Chess" (WAC, 300 positions), and "1001 Winning Chess Sacrifices" (WCS, 1001 positions).

- LCT II Test. http://perso.wanadoo.fr/lefouduroi/test_lct_native.htm
- Nunn Positions. http://www.chessbaseusa.com/fritz5/nunnmtch.htm
- Nunn-II Positions. http://www.computerschach.de/index.php?option=com_remository&Itemid=52&func=fileinfo&id=3
- The hardware used was a Apple PowerMac Dual 1.8 GHz PowerPC G5 with 2.25 GB Ram.
- All versions of CRAFTY used the default settings of 3 MB for hash table, and 768KB for pawn hash table. No endgame database was used.
- Pondering was turned off. CPU time was used during test suites and elapsed time was used during matches.

References

Aichholzer, O.; Aurenhammer, F.; and Werner, T. 2002. Algorithmic fun - abalone. *Special Issue on Foundations of Information Processing of TELEMATIK* 1:4–6.

Akl, S. G., and Newborn, M. M. 1977. The principle continuation and the killer heuristic. In *ACM Annual Conference*, 466–473.

Beal, D. F. 1980. An analysis of minimax. In *Advances in Computer Chess 2*, 103–109. Edinburgh University Press.

Beal, D. F. 1989. Experiments with the null move. In *Advances in Computer Chess 5*. Elsevier Science. 65–79.

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

Björnsson, Y., and Marsland, T. A. 2000a. Risk management in game-tree pruning. *Inf. Sci* 122(1):23–41.

Björnsson, Y., and Marsland, T. A. 2000b. Selective depth-first search methods. In van den Herik, H. J., and Iida, H., eds., *Games in AI Research*, 31–46.

Björnsson, Y., and van den Herik, H. J. 2005. The 13^{th} world computer-chess championship. *International Computer Games Association Journal* 28(3).

Buro, M. 1995. ProbCut: An effective selective extension of the $\alpha - \beta$ algorithm. *International Computer Chess Association Journal* 18(2):71–76.

Buro, M. 1997. The othello match of the year: Takeshi murakami vs. logistello. *International Computer Chess Association Journal* 20(3):189–193.

Buro, M. 1999. Experiments with Multi-ProbCut and a new high-quality evaluation function for Othello. In van den Herik, H. J., and Iida, H., eds., *Games in AI Research*.

Donninger, C. 1993. Null move and deep search: Selective search heuristics for obtuse chess programs. *International Computer Chess Association Journal* 16(3):137–143.

Goetsch, G., and Campbell, M. S. 1990. Experiments with the null-move heuristic. In Marsland, T., and Schaeffer, J., eds., *Computers, Chess, and Cognition*. Springer-Verlag. 159–168.

Heinz, E. A. 1998a. Darkthought goes deep. *International Computer Chess Association Journal* 21(4):228–244.

Heinz, E. A. 1998b. Extended futility pruning. *International Computer Chess Association Journal* 21(2):75–83.

Heinz, E. A. 1999a. Adaptive null-move pruning. *International Computer Chess Association Journal* 22(3):123–132.

Heinz, E. A. 1999b. Self-play experiments revisited. In van den Herik, J., and Monien, B., eds., *Advances in Computer Chess 9*, 73–91.

Heinz, E. A. 2000. AEL pruning. *International Computer Games Association Journal* 23(1):21–32.

Jiang, A., and Buro, M. 2003. First experimental results of probcut applied to chess. In van den Herik, H. J.; Iida, H.; and Heinz, E. A., eds., *Advances in Computer Games Conference 10*, 19–31.

Junghanns, A.; Schaeffer, J.; Brockington, M.; Bjornsson, Y.; and Marsland, T. 1997. Diminishing returns for additional search in chess. In van den Herik, J., and Uiterwijk, J., eds., *Advances in Computer Chess 8*, 53–67. Univ. of Rulimburg.

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

Kocsis, L. 2003. *Learning Search Decisions*. Ph.D. Dissertation, Universiteit Maastricht, IKAT/Computer Science Department.

Marsland, T. A. 1986. A review of game-tree pruning. *International Computer Chess Association Journal* 9(1):3–19.

Moriarty, D., and Miikkulainen, R. 1994. Evolving neural networks to focus minimax search. In *Proceedings of 12th National Conference on Artificial Intelligence (AAAI-94)*, 1371–1377.

Nau, D. S. 1979. *Quality of decision versus depth of search on game trees*. Ph.D. Dissertation, Duke University.

Plaat, A. 1996. *Research Re: search & Re-search*. Ph.D. Dissertation, Erasmus University Rotterdam, Rotterdam, Netherlands.

Reinefeld, A. 1989. *Spielbaum Suchverfahren*, volume Informatik-Fachberichte 200. New York, NY: Springer-Verlag. In german.

Schaeffer, J. 1986. *Experiments in Search and Knowledge*. Ph.D. Dissertation, University of Waterloo.

Schaeffer, J. 1989. The history heuristic and alpha-beta search enhancements in practice. *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-11(11):1203–1212.

Tabibi, O. D., and Netanyahu, N. S. 2002. Verified null-move pruning. *International Computer Games Association Journal* 25(3):153–161.