



**University of Alberta**

**Risk Management in Game-Tree Pruning**

by

Y. Björnsson and T.A. Marsland

Technical Report TR 98-07  
July 1998

DEPARTMENT OF COMPUTING SCIENCE  
University of Alberta  
Edmonton, Alberta, Canada

# Risk Management in Game-Tree Pruning

Yngvi Björnsson and Tony Marsland  
Department of Computing Science  
University of Alberta  
Edmonton, Alberta  
CANADA T6G 2H1  
E-mail: {yngvi,tony}@cs.ualberta.ca

July 30, 1998

## 1 Introduction

The thinking-process used by computers for chess and other two-person games differs significantly from the one used by humans. While humans consider at most a few alternatives when deciding what to play, computers exhaustively search all the possible moves. In the half century since *minimax* was first suggested as a strategy for adversary game search, various search algorithms have been developed. The standard approach has been to use improvements to the Alpha-Beta ( $\alpha$ - $\beta$ ) algorithm to explore all continuations to some fixed depth (continuation length or search horizon). In practice, however, the algorithms are not used that way, instead heuristics vary the search horizon, exploring some continuations more deeply than others. In an indirect way, this resembles the human thinking process. Continuations that are thought to be of special interest are expanded beyond the nominal depth, while others may be terminated prematurely. The latter case is referred to as *forward pruning*.

In this paper we discuss some important aspects of forward-pruning, especially regarding risk-management, and propose ways of improving risk-assessment. Finally, we introduce two new pruning methods based on some of the principles discussed here, and present experimental results applying the methods in an established chess program.

## 2 Selective Search and Decision Quality

The number of nodes visited by Alpha-Beta increases exponentially with increasing search depth. This limits the scope of the search, especially because game-playing programs must meet external time-constraints: often having only a few minutes to decide which move to play. In practice, the quality of the move decision usually improves the further we look ahead. The question now becomes, how to find a good move while making the best use of the available time. One approach is to prune off unpromising lines, so that the time saved can be used to search selected lines more deeply.

There exist several well established forward-pruning methods for game-tree search, the most successful probably being the null-move heuristic [Beal, 1990] [Goetsch and Campbell, 1990] and ProbCut [Buro, 1995], for chess and Othello respectively. On the other hand, the theoretical issue surrounding forward pruning and how it affects decision quality has not been

studied much. However, Smith and Nau [1994] analysed a model of forward-pruning using (over)simplified game trees, concluding that forward-pruning works best when there is a high correlation among the minimax values of sibling nodes in the game tree. For specific games like chess, checkers and Othello, some empirical studies that investigate how error in leaf-node evaluation affects the move decision at the root also exist [Junghanns *et al.*, 1997].

### 3 Forward Pruning

The real task when doing forward pruning is to identify move sequences that are worth considering more closely, and others that can be pruned off with minimal risk of overlooking a good continuation. Several factors should be considered for effective forward-pruning:

- *Risk-assessment.*  
How safe is the forward pruning method? We want to minimize the risks that speculative pruning introduces into the search.
- *Applicability.*  
To maximize the possible gains from forward pruning we would like to apply the method frequently in the tree, especially where there is a potential for big savings.
- *Cost-effectiveness.*  
The investment of time and effort to decide whether to prune a node should be kept low. In any case, the savings achieved through pruning must exceed the additional effort introduced.
- *Domain-independency.*  
Ideally, we want a pruning method that can be applied in more than one specific search domain.

The above factors are by no means independent, improving one usually involves compromising another. For example, reducing the risk factor often means limiting the applicability. Also, improving cost-effectiveness can introduce other risks, and finally the more general (domain-independent) methods tend to be less efficient. A useful forward pruning heuristic needs to find the appropriate trade-off between the above factors, and this process may require careful tuning.

#### 3.1 Risk-assessment

When using forward pruning there is always some danger of overlooking good moves. We would like to minimize the risk of doing so. When deciding whether to examine a node  $v$ , the basic question is: how likely is it that the sub-tree below  $v$  includes a continuation that, if searched, would yield a new principal variation (pv)? For a new variation to emerge two things must occur; first the value returned to  $v$  must exceed the best value found so far, and second the value must propagate to the root of the tree. This in turn implies that the pruning method should be able

- to predict with reasonable accuracy the range of values for node  $v$ , and
- to measure the likelihood that the anticipated value will back up to the root of the tree.

Existing forward pruning methods address the first issue while often ignoring the second one.

### 3.1.1 Error Introduction

For most subtrees we are not so much interested in knowing the exact value of each particular node, but rather whether the value lies outside the bounds of the  $\alpha$ - $\beta$  window. This is because we know that continuations which result in values outside the window can never become a part of the principal variation. When using a null-window search the bound is the value of the current principal variation, so when comparing node values to the bound we are determining whether a better continuation is found. In that case we are simply interested in knowing if a value returned by searching a node further is as good as the  $\beta$ -bound, since it causes a cutoff.

When predicting where the value of a node  $v$  lies relative to the  $\alpha$ - $\beta$  bounds, most pruning methods carry out a shallow search. They use the value returned to estimate the range in which the actual value of node  $v$  is likely to be when the node is searched more deeply. For example, a 5-ply-deep search is used to predict the window for a 6-ply-deep search. The outcome of the shallow search decides whether to search node  $v$  further. If we are confident enough that further search will not yield an improvement, node  $v$  is not expanded. The exact criteria used to relate the value of the shallow search to the anticipated return value of the deeper search varies with the pruning technique. Some approaches use statistical methods to define confidence intervals, while others simply use *ad hoc* heuristics. Error is introduced into the search when a wrong pruning decision is made.

Although, shallow searches are usually reasonable indicators of the values returned by deeper searches, additional information can enhance the overall prediction capabilities of the pruning heuristics, thereby reducing the risk involved. For example, consider the tree in Figure 1. The shaded area marks the parts of the tree searched to decide whether to prune

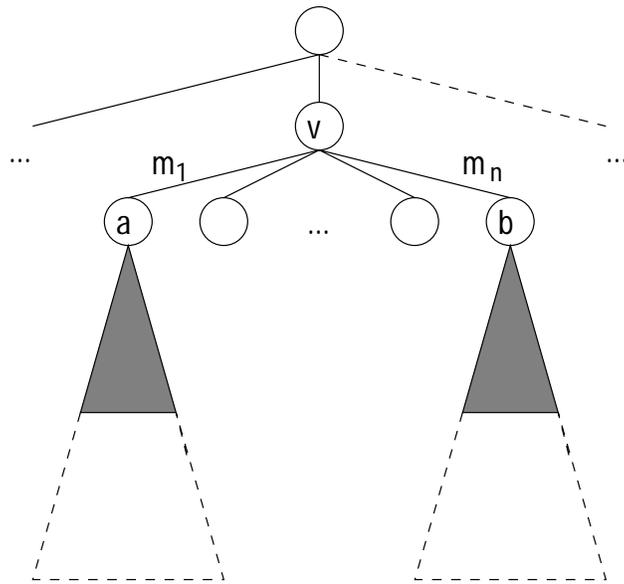


Figure 1: Different risk-assessment.

nodes  $a$  and  $b$ . Each pruning decision is made independently of the other, based only on

the outcome of the local search. However, by looking at each node in isolation information is lost. For example, when looking at move  $m_n$  current pruning methods are interested in knowing if the move will lead to a value that causes a cutoff, that is, in estimating the probability

$$P(v(m_n) \geq \beta).$$

But having already searched moves  $m_1, \dots, m_{n-1}$ , and knowing that none caused a cutoff, provides a strong indicator that move  $m_n$  will also fail to do so, especially because the preliminary move-ordering scheme believes that move  $m_n$  is no better than the moves already searched. Instead one should compute the probability that move  $m_n$  causes a cutoff, given that moves  $m_1, \dots, m_{n-1}$  have failed to do so, that is compute:

$$P(v(m_n) \geq \beta \mid v(m_1, \dots, m_{n-1}) < \beta).$$

The values of the moves are not independent of each other, and so by assuming otherwise we are ignoring potentially useful information. Existing pruning methods and probability based best-first search algorithms totally ignore the dependencies, or unrealistically assume the search values (or the error in the values) to be independent of each other. *Instead, the fact that the values tend to be dependent should be used to make more informed pruning decisions.*

### 3.1.2 Error Propagation

Figure 2 shows two different game trees. The solid lines identify the parts of the tree that have already been visited, while the dotted lines show nodes that have not been expanded. Assume that the search is currently situated at node  $v$  and that the sub-tree resulting from

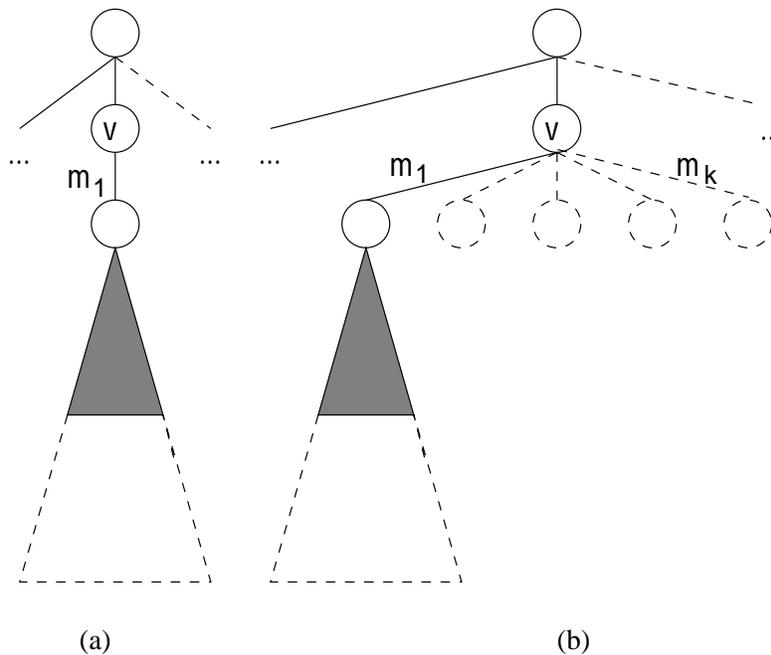


Figure 2: Controlling error propagation.

playing move  $m_1$  has already been searched. Furthermore, assume that a part of that subtree has been pruned off using some forward pruning technique, and that the value returned

is greater or equal to the  $\beta$ -bound used at node  $v$  (if node  $v$  is a cut-node this is what we would expect). Therefore, a  $\beta$ -cutoff occurs and the value returned by move  $m_1$  will back up to the root. From the root's perspective this branch is inferior to the current principal variation and the search therefore continues to expand the other children of the root without switching the principal variation.

If the pruned subtree in Figure 2(a) does not contain a better line, search effort has been saved. The case of interest here is: what if a better line *is* present? In Figure 2(a), if a better line is present but is overlooked, the value of  $m_1$  is wrong and the error may propagate through node  $v$  to the root. However, if alternatives to  $m_1$  are present, as in Figure 2(b), it is possible that one of the alternative moves in  $[m_2, \dots, m_k]$  may contain a line that enables it to deliver a beta-cutoff at  $v$ , acting as a substitute for  $m_1$ , thus preserving the value assigned at node  $v$ . Thus in Figure 2(b), an error in the subtree below  $m_1$  does not necessarily propagate to the root. This situation is common in practice: if the first move fails to cause a cutoff, one of the alternative moves may do so. This means that even though the pruning below  $m_1$  was risky, the risk of affecting the move decision at the root is less in Figure 2(b) than in Figure 2(a), because one of the other moves  $m_2 \dots, m_k$  might preserve the cutoff if  $m_1$  changes its value. Thus, *even though an erroneous pruning is made it will not necessarily affect the move decision at the root*. Hopefully, this illustrates that, when assessing risk, pruning methods should not only take into account the expected return value of a pruned node, but also assess the likelihood that an erroneous pruning decision will propagate up the tree.

### 3.2 Applicability

The most popular pruning heuristics used in two-person game-playing programs have one thing in common: they apply frequently, though not without restriction. The more frequently a pruning heuristic is applied in the search, especially at places where there is a high probability of big savings, the more potential it has for being effective. However, the applicability is restricted, since pruning can only be done where it is expected to be safe. Depending on the heuristics used, this can differ substantially. Some heuristics need additional pre-requirements for them to be applied. An example of one such pruning method is the special cutoff introduced by the NegaScout [Reinefeld, 1983] algorithm. Although the cutoff is risk-free when search extensions are not used, the savings are very small (less than 2%). This is because the necessary pre-requirement (that is, a change in the principal variation is occurring two or less plies from the search horizon) for the cutoff is met infrequently.

### 3.3 Cost-effectiveness

Although some pruning methods offer low risks and substantial savings in terms of nodes searched, the overhead needed to implement them is often prohibitive. The effort expended gathering and tracking in real-time the information required by the heuristics may outweigh the potential time-savings introduced by the pruning. An example of such a heuristic is the method of analogies. Although, the method offers almost risk-free pruning, the overhead of tracking how pieces influence each other originally proved too high for practical use in a competitive chess playing program [Adelson-Velskiy *et al.*, 1975]. However, changes in software and hardware technology may improve the efficiency of such methods. It might also be possible to approximate the original heuristic by another that is less costly to maintain,

and yet achieve most of the savings. Therefore the method of analogies is again a topic worthy of investigation.

### 3.4 Domain-dependency

Ideally we want domain-independent pruning. Such methods would not rely on such explicit knowledge as: is a king in check, or whether a corner square is occupied. If domain-specific knowledge, is used, it is incorporated in a domain independent way, for example externally state the knowledge in a language that can be compiled and used in a general way by the search.

In the domain independent methods, the only information revealed to the search by the evaluation function is a numerical estimate of a problem state's quality. This clear separation of the search and the problem encourages more domain-independent pruning methods. On the other hand the methods are then denied access to potentially useful information about the problem domain, thereby restricting their pruning capabilities. However, there is a wealth of information to be gathered about the problem by simply looking at the shape of the expanded search tree. This knowledge is accessible without having to uncover any additional domain-specific knowledge. We have already mentioned a few cases of interest during our risk-assessment discussion.

In practice, it is extremely difficult for pruning methods to be domain independent. As said earlier, there is a trade-off between generality and effectiveness, and to achieve the full pruning capability we have to exploit some special characteristics of the search space. Most existing forward pruning methods are therefore domain specific. Even though methods like null-move [Beal, 1990] and ProbCut [Buro, 1995] do not use explicit knowledge about their domain, they make implicit assumptions that tie them down for use in one, or at best very few, two-person games. For example, the null-move heuristic is very effective in chess, but inappropriate in Othello. Conversely, ProbCut is the pruning heuristic of choice in Othello but has not yet been shown useful in chess or checkers.

## 4 New Forward Pruning Methods

As mentioned before, pruning heuristics should be concerned with the question: *What is the likelihood of making an erroneous pruning decision, and if an erroneous decision is made how likely is it to affect the principal variation?* Existing forward-pruning methods generally do not consider the second part of this question. When assessing risk, pruning methods should not only speculate whether a subtree contains a good continuation, but also determine if there are alternatives to any potentially overlooked continuation that could preserve the principal variation. To answer these questions the methods must consider each node in the context of its location in the game-tree, instead of looking at each node (and the subtree below it) in isolation.

In the following sections we introduce two new forward pruning methods: *Multi-Cut*, and Variable Null-move Bound. Both our new methods indirectly consider the likelihood that the consequences of an erroneous pruning decision propagate to the root of the game tree.

## 5 Multi-Cut

In this section we present the idea behind the multicut method, then give implementation details, and experimental results.

### 5.1 Multiple cut-offs idea

In a traditional  $\alpha\beta$ -search, if a move returns a value greater or equal to  $\beta$  there is no reason to examine that position further, and the search can return. This is often referred to as a  $\beta$ -cutoff. Intuitively, this means that the player to move has found a way to refute the current line of play, so there is no need to find something better. By way of explanation, and to introduce our terminology, we are seeking the principal variation: the best sequence of moves from the root node (current position in the game) to the best of the accessible nodes on the search horizon. We expect  $\beta$ -cutoffs to occur at so called cut-nodes (that is, nodes that are refuted). The root node of a game-tree is a pv-node (principal variation node), the first child of a pv-node is also a pv-node, while the other children are cut-nodes. All children of a cut-node are all-nodes (where every successor must be explored) and vice versa. In a perfectly ordered tree only one child of a cut-node is expanded. If a new best move is found at a pv-node, the node it leads to is also a pv-node. At pv- and all-nodes every successor is examined. Most often it is the first child that causes the cutoff, but if it fails to do so the sibling nodes are expanded in turn, until one of them returns a value greater or equal to  $\beta$  (thus causing a cutoff) or all the children have been searched. If none of the moves causes a cutoff, a cut-node becomes an all-node.

For a new principal variation to emerge, every expected cut-node on the path from a leaf-node to the root must become an all-node. In practice, however, it is common that if the first move does not cause a cutoff, one of the alternative moves will. Therefore, *expected cut-nodes, where many moves have a good potential of causing a  $\beta$ -cutoff, are less likely to become all-nodes, and consequently such lines are unlikely to become part of a new principal-variation.* This observation forms the basis for the new forward pruning scheme we introduce here, *multi-cut  $\alpha\beta$ -pruning*. Before explaining how it works, let us first define a *mc-prune* (multi-cut prune).

**Definition 1 (mc-prune)** *When searching node  $v$  to depth  $d + 1$  using  $\alpha\beta$ -search, and if at least  $c$  of the first  $m$  children of  $v$  return a value greater or equal to  $\beta$  when searched to depth  $d - r$ , a mc-prune is said to occur and the search can return.*

In multi-cut  $\alpha\beta$ -search, we attempt a *mc-prune* only at expected cut-nodes (we would not expect it to be successful elsewhere). Figure 3 shows the basic idea. At node  $v$ , but before searching  $v_1$  to a full depth  $d$ , as normal  $\alpha\beta$ -search does, the first  $m$  successors of  $v$  are expanded to a reduced depth of  $d - r$ . If  $c$  of them return a value greater or equal to  $\beta$  a *mc-prune* occurs and the search returns the value of  $\beta$ , otherwise the search continues as usual exploring  $v_1$  to a full depth  $d$ . The subtrees below  $v_2, \dots, v_m$  represent extra search overhead introduced by *mc-prune*, since they would not be expanded by normal  $\alpha\beta$ -search. The dotted area of the subtree below node  $v_1$  represents the savings that are possible if the *mc-prune* is successful. However, if the pruning condition is not satisfied, we are left with the overhead but no savings. By searching the subtree of  $v_1$  to a shallower depth, there is of course some risk of overlooking a tactic that would make  $v_1$  become a new principal variation. We are willing to take that risk, because we expect at least one of the  $c$  moves that returns a value greater or equal to  $\beta$ , when searched to a reduced depth, will still cause a genuine  $\beta$ -cutoff if searched to a full depth.

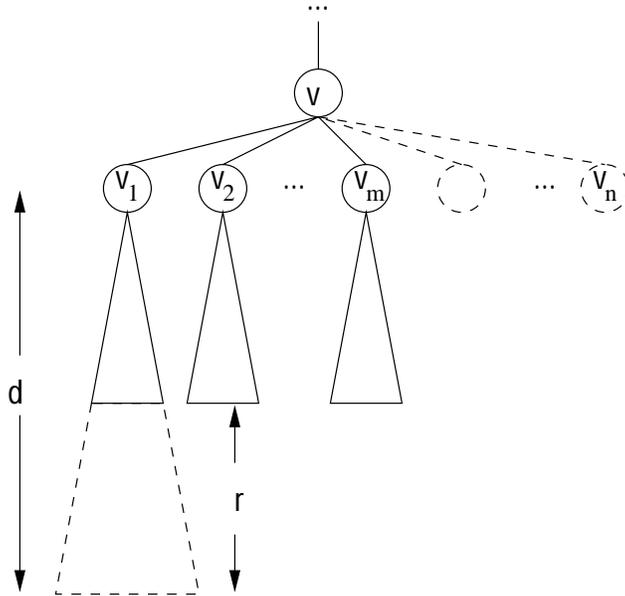


Figure 3: Principles of multi-cut pruning

## 5.2 Implementation

Figure 4 is a C-code version of a null-window search (NWS) routine using multi-cut. For clarity we have omitted details about search extensions, transposition table lookups, null-move searches, and history heuristic updates that are not immediately relevant to our discussion. The *NWS* routine [Marsland, 1993] could for example be called by an enhanced  $\alpha\beta$ -variant like *Principal Variation Search* or *NegaScout*. The parameter *depth* is the remaining length of search for the position, and  $\beta$  is an upper-bound on the value we can achieve. There is no need to pass  $\alpha$  as a parameter, because it is always equal to  $\beta - 1$ . In our case, however, the new parameter *ntype* is needed to identify the nodes where *mc*-pruning applies.

As is normal, the routine starts by checking whether the horizon has been reached, and if so to use a quiescence search (QS) to return the value of the position. Otherwise, we look in the transposition table for useful information to guide the search. This is followed by a null-move search (most chess programs use this powerful technique) [Beal, 1990]. Normally one of the  $\alpha\beta$  search variants would follow, if the null-move does not cause a cut-off. Instead we insert here a multicut search to see if the *mc*-prune condition applies. The parameters *mc\_M*, *mc\_R*, and *mc\_C* stand for *m* (number of moves to look at), *r* (search reduction), and *c* (number of cutoffs needed), respectively. Although they are shown here as constants, they could be determined more dynamically and allowed to vary during the search.

We do not check for the *mc*-prune condition at every node in the tree. First, we test for it only at expected cut-nodes. Second, it is not applied at the levels of the search tree close to the horizon, thus reducing the time overhead involved in this method. Finally, there are some game-dependent restrictions that apply. In Figure 4 these restrictions are encapsulated in the function *TryMultiCut()*. In our experiments in the domain of chess (see later) the pruning is disabled when the endgame is reached. Usually only a few viable move options exist there, and so the *mc*-search is not likely to be successful. Also, the

```

// Multi-Cut parameters
#define mc_M 10 //# of moves to look at
#define mc_C 3 //# of cuts to cause a mc-prune
#define mc_R 2 //depth reduction

#define CUT 2
#define ALL 3
#define TYPE(t) (((t)==CUT) ? ALL : CUT)

VALUE NWS(int depth, VALUE beta, NODETYPE ntype)
{
    VALUE score;
    MOVE move;

    // Search extension code, omitted ...

    if ( depth <= 0 ) return QS(beta-1);

    // Transposition table lookup, omitted ...
    // ... retrieve best move and bound information...

    // Nullmove search, omitted ...
    // ... terminate search on null-move cutoff ...

    // Multi-Cut pruning
    if ( (ntype == CUT) && (depth > mc_R)
        && TryMultiCut() ) {
        int m = 0;
        int c = 0;
        move = MoveFirst();
        while ( m < mc_M && move ) {
            MakeMove( move );
            score = -NWS(depth-mc_R-1, -beta+1, TYPE(ntype));
            RetractMove( move );
            if ( score >= beta ) {
                c++;
                if ( c == mc_C ) return beta;
            }
            m++;
            move = MoveNext();
        }
    }

    // Standard null-window (minimal window) search
    move = MoveFirst();
    while ( move ) {
        MakeMove( move );
        score = -NWS(depth-1, -beta+1, TYPE(ntype));
        RetractMove( move );
        if ( score >= beta ) break;
        move = MoveNext();
    }

    // Store node information, omitted ...
    // ... update trans table and history heuristic...

    return score;
}

```

Figure 4: Multi-Cut null-window search.

positional understanding of chess programs in the endgame is generally poorer than in the earlier phases of the game. Therefore the programs rely heavily on the search to guide them, and so any forward pruning scheme is more likely to be harmful. Finally, the pruning is not done if the side to move is in check, or if a search extension has been applied at any of the three previous moves leading to the current position.

### 5.3 Experimental Results

Ultimately, we want to show that game-playing programs using the new pruning method can achieve increased playing strength. To test the idea in practice multi-cut  $\alpha\beta$ -pruning was implemented in *The Turk*<sup>1</sup>. Two versions of the program were matched against each other, one using multi-cut pruning and the other without. Several matches, with 80 games each, were played using different time controls. To prevent the programs from playing the same game over and over, forty well known opening positions were used as a starting point. The programs played each opening once from the white side and once as black. Table 1 shows the match results.  $T$  stands for the unmodified version of the program and  $T_{mc(c,m,r)}$  for the version with multi-cut implemented. We experimented with the case  $m = 10$ ,  $r = 2$ , and  $c = 3$  (i.e. 10 moves searched using a depth reduction of 2 and requiring 3  $\beta$ -cutoffs to achieve the  $mc$ -prune condition). These parameter values are somewhat arbitrary, although based on experience and a few test trials.

$T_{mc(3,10,2)}$ versus $T$		
Time control	Score	Winning %
40 moves in 5 minutes	41 - 39	51.3
40 moves in 15 minutes	40 - 40	50.0
40 moves in 25 minutes	44 - 36	55.0

Table 1: Multi-cut pruning match result.

The multi-cut version shows a slight improvement over the unmodified version. In tournament play this winning percentage would result in about 15 points difference in the players' rating. Since more than 1000 games are typically needed to obtain a standard error of less than 10 rating points [Anantharaman, 1990], we cannot be sure that the multi-cut version is the stronger through this single set of experiments.

One final insight, the programs gathered statistics about the behavior of the multi-cut pruning. The search spends about 25%-30% of its time (in terms of nodes visited) in shallow multi-cut searches, and  $mc$ -prune occurs in about 45%-50% of its attempts. Obviously, the tree expanded using multi-cut pruning differs significantly from the tree expanded when it is not used.

## 6 Variable Null-move Bound

In this section we describe a new enhancement to the null-move heuristic, *variable null-move bound* search, that utilizes some of the aforementioned observations to search more efficiently.

---

<sup>1</sup>*The Turk* is a chess program developed at University of Alberta by Yngvi Björnsson and Andreas Junghanns

## 6.1 Basis

Goetsch and Campbell [1990] mention as a future research idea the possibility of permitting a null-move cutoff not only when a null-move search returns a value greater or equal to  $\beta$ , but also if the returned value is slightly less. They propose that a cutoff be forced if  $v \geq \beta - t$ ,  $t \geq 0$ , where  $v$  is the value returned by the null-move search and  $t$  is a small positive number that can be interpreted as the value of a tempo. This allows null-move cutoffs to be applied more frequently thereby reducing the tree-size even further, although at the cost of introducing additional errors. Furthermore, they state that the value of  $t$  must be lower than the actual value of a tempo to avoid inadvertent cutoffs, and that the value of  $t$  would be dependent on the evaluation function and could vary during the course of the game. Both these factors help reduce the risk of erroneous pruning.

The method we introduce here is based on the same idea. However, we use a different approach for approximating  $t$ . Instead of having  $t$  depend on the evaluation function, we let it vary according to how likely we think an erroneous pruning decision affects the principal variation. Indeed, in some parts of the tree we allow the value of  $t$  to exceed what would normally be considered an appropriate value for a tempo.

## 6.2 Implementation

First we need a metric to show how likely it is that a pruning error affects the principal variation. The more alternative moves a player has to refute the opponent's play, the less likely it is that an oversight in assessing an individual node will affect the move decision at the root (see the discussion with Figure 2). This suggests that one metric is the number of potentially good alternative moves a player has on the path leading from the root to the current node in the search tree. However, there are a couple of difficulties. First, at cut-nodes only one, or at most few, moves are considered, leaving us with no information about the remainder. Second, since programs commonly employ a null-window search, for most nodes in the tree we have only bounds on the actual value of a node, making it difficult to compare the merits of any two moves. One approach would be to perform additional shallow searches to estimate the value of each move, but this would imply a considerable extra search overhead, possibly offsetting any gains. Fortunately, there are more cost-effective means of approximating the number of potentially good alternative moves. Most programs use some form of the history-heuristic [Schaeffer, 1989] for move ordering. This heuristic gives a credit to moves that cause a cutoff. We simply define a move to be a potentially good alternative if it has a positive history heuristic value. Although this is not the most accurate approximation it is a cost-effective one.

We implemented the variable null-move bound heuristic in *The Turk*. The program uses principal variation search, and null-moves are applied recursively with a search reduction of 2. Several restrictions control where and when the null-move heuristic applies. For example, a null-move is not allowed on the principal variation or if the side to move is in check. Neither are two consecutive null-moves allowed. Figure 5 shows how the null-move heuristic is applied in the variable bound scheme. The variable *no\_pgam* is the number of potentially good alternative moves (as defined above) that are found on the path from the root, but are still unexplored. A separate count is kept for each player and is updated incrementally as the tree is traversed.

In the current implementation the number of potentially good alternative moves is recorded during a zero-window search, with the exception of the first move expanded at

each node. The main reasons for this is that the first move is most often taken from the transposition table and expanded before any legal moves are generated. Since we count the number of good alternative moves for each level in the tree at the time of move generation, that information is not available to pass down for the first move. Since the *no\_pgam* count is not updated for that level, this makes the program less aggressive in pruning along these paths<sup>2</sup>.

```

if ( NULLMOVE_OK() ) {
    int bound, t = 0;
    if ( !InNullMoveSearch() ) {
        if ( no_pgam > 15 ) {
            t = 20;
        }
        else if ( no_pgam > 0 ) {
            t = 10;
        }
    }
    bound = beta - t;
    Make ( Nullmove );
    score = -NWS ( depth+1,-bound+1,max_depth-2 );
    Retract ( Nullmove );
    if ( score >= bound ) {
        return ( beta );
    }
}

```

Figure 5: Variable bound null-move cutoff decision.

### 6.3 Experimental Results

We did preliminary experiments to assess the viability of the new heuristic. Three different variants of the chess program ( $TT_t$ ) were matched against an unmodified version of the program ( $TT$ ), each using a different value for the tempo,  $t$ . For two of the variants,  $TT_{10}$  and  $TT_{20}$ ,  $t$  was set to a fixed constant, 10 and 20 respectively<sup>3</sup>. The third,  $TT_{variable}$ , varied the value of  $t$  using history data (see above). The relationship between *no\_pgam* and  $t$  is as shown in Figure 5, and was chosen based on some trial and error tests. In a future implementation a more appropriate relationship will be empirically determined. Each match consisted of 40 games, with the time controls set to 40 moves in 5 minutes. To prevent the programs from playing the same game over and over, twenty well known opening positions were used as a starting point. The programs played each position twice, once from the white side and once as black. The match results are shown in Table 2.

Of the three variants,  $TT_{variable}$  performed the best, outplaying the original version with a 60% winning percentage.

Although the preliminary results are encouraging, care must be taken in interpreting them. First, more than 40 games are needed to reliably determine a difference in playing

<sup>2</sup>Since the first move expanded is often the most critical one, this compromise might actually be beneficial.

<sup>3</sup>The value of a pawn is 100.

Match	Score	Winning %
$TT$ vs. $TT_{10}$	20 - 20	50
$TT$ vs. $TT_{20}$	20.5 - 19.5	49
$TT$ vs. $TT_{variable}$	16 - 24	60

Table 2: Variable null-move bound match results

strength between any two programs, and second, games with actual tournament time controls must be played. The preliminary results indicate that this method has some potential and is worth refining and experimenting with, but more study is definitely needed to reach a final verdict.

## 7 Conclusions

We have discussed some important characteristics of forward pruning in game-tree search, emphasizing risk-management, and propose two ways to improve risk-assessment:

- consider move dependency information, and
- consider the likelihood that erroneous pruning decisions influence the principal variation.

We develop two new forward-pruning methods based on the above principles.

Preliminary experiments with the new methods are encouraging. Although the methods are still in their infancy, they demonstrate a small gain over current state of the art search techniques. We feel there is still much scope for improvement through further tuning and enhancements. For example, in *multi-cut* the optimal setting of the parameters  $c$ ,  $m$ , and  $r$  is not yet known. The two new forward-pruning methods introduced here are by no means the only way of exploiting the above principles for improved risk-assessment, there is still room for other innovative methods.

## References

- [Adelson-Velskiy *et al.*, 1975] G. M. Adelson-Velskiy, V. L. Arlazarov, and M. V. Donskoy. Some methods of controlling the tree search in chess programs. *Artificial Intelligence*, 6(4):361–371, 1975.
- [Anantharaman, 1990] T. Anantharaman. *A Statistical Study of Selective Min-Max Search in Computer Chess*. PhD thesis, Carnegie-Mellon University, Pittsburgh, PA, May 1990.
- [Beal, 1990] D. F. Beal. A generalized quiescence search algorithm. *Artificial Intelligence*, 43:85–98, 1990. See also: Experiments with the Null Move. In *Advances in Computer Chess 5*, 1989, 65-79.
- [Buro, 1995] M. Buro. ProbCut: An effective selective extension of the alpha-beta algorithm. *ICCA Journal*, 18(2):71–76, 1995.
- [Goetsch and Campbell, 1990] G. Goetsch and M.S. Campbell. Experimenting with the null move heuristic. In T. Marsland and J. Schaeffer, editors, *Computers, Chess and*

- Cognition*, pages 158–168, 1990. See also: 1988 AAAI Spring Symposium Proceedings, 14-18.
- [Junghanns *et al.*, 1997] A. Junghanns, J. Schaeffer, M. Brockington, Y. Björnsson, and T. Marsland. Diminishing returns for additional search in chess. In *Advances in Computer Chess 8*, June 1997. To appear.
- [Marsland, 1993] T. Marsland. Single-Agent and Game-Tree Search. In A. Kent and J. G. Williams, editors, *Encyclopedia of Computer Science and Technology*, volume 27, pages 317–336, New York, 1993. Marcel Dekker, Inc.
- [Reinefeld, 1983] A. Reinefeld. An improvement to the Scout tree search algorithm. *ICCA Journal*, 6(4):4–14, 1983.
- [Schaeffer, 1989] Jonathan Schaeffer. The history heuristic and alpha-beta search enhancements in practice. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(1):1203–1212, 1989.
- [Smith and Nau, 1994] S. J. J. Smith and D. S. Nau. An analysis of forward pruning. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, volume 2, pages 1386–1391, 1994.