Selective Depth-First Search Methods

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

Abstract

In this paper we take a general look at forward pruning in tree search. By identifying what we think are desirable characteristics of pruning heuristics, and what attributes are important for them to consider, we hope to understand better the shortcomings of existing techniques, and to provide some additional insight into how they can be improved. We view this work as a first step towards the goal of improving existing forward-pruning methods.

1 Introduction

The thinking-process used by computers for playing 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 algorithms have been developed, most notably Alpha-Beta and its variants. The Alpha-Beta like algorithms, as formulated, explore all continuations to some fixed depth. However, in practice the algorithms are not used that way, instead various heuristics are used that allow them to vary the search horizon, exploring some moves more deeply than others. In an indirect way, this resembles the human approach. Continuations that are thought to be of interest are expanded beyond the nominal depth, while others of less interest are terminated prematurely. The latter case is referred to as *forward pruning*.

In this paper we discuss forward pruning from a general perspective, and identify important factors to consider when developing a pruning heuristic. We describe a few existing forward-pruning methods and try to assess how well they address the factors we identify as being important. This work is a first step towards the goal of improving existing forward-pruning methods and developing new ones, and will help us recognize the shortcomings of the existing techniques and thereby guide us to potential improvements. First the rationale behind forward pruning is discussed, followed by a description of desirable characteristics of forward-pruning methods and various other factors to consider. Section 4 discusses existing pruning methods, and finally we conclude by outlining our thoughts on the promise for improvement.

2 Selective Search and Decision Quality

The number of nodes visited by Alpha-Beta increases exponentially with increasing search depth. This obviously limits the scope of the search, especially because game-playing programs have to meet external time-constraints: often having only a few minutes to decide which move to play. In general, the quality of the move decision improves the further we lookahead[1]. The question now becomes, how to make best use of the available time to find a good move. The rationale behind selective search is that the time saved by pruning off non-promising lines is better used to search other lines more deeply and therefore, hopefully, to increase the overall decision quality of the search.

The theoretical issue surrounding forward pruning and how it affects decision quality has not been studied much. Smith and Nau [1994] introduced 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. Conversely, for chess, checkers and Othello some empirical studies exist that investigate how error in leaf-node evaluation affects the move decision at the root [Junghanns *et al.*, 1997].

[1]Some artificial games have been constructed where the opposite is true; when backing up a minimax value the decision quality actually decreases as we search deeper. This phenomenon has been studied thoroughly and is referred to as *pathology* in game-tree search [Nau, 1980]. However, this pathology is not seen in chess or the other games we are investigating.

3 Forward pruning

When applying forward pruning the real task is to identify which move sequences are worth considering more closely and which can be pruned off with minimal risk of overlooking a good continuation. Several factors should be considered when coming up with an effective forward-pruning heuristic:

- *Risk-assessment.*
  How safe is the forward pruning method? We want to minimize the risk that speculative pruning introduces errors into the search.
- *Applicability.*
  To maximize the possible gains of using forward pruning while searching we would like to apply the pruning frequently in the tree, especially where there is a potential for big savings.
- *Cost-effectiveness.*
  The investment of time and effort used 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 equally well in other search domains.

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 effective. A useful forward pruning heuristic needs to find the appropriate trade-off between the above factors, and this process may require a careful tuning effort.

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. Basically, the question we need to answer when deciding whether to examine a node v is: how likely is the sub-tree below v to include a continuation that, if searched, would yield a new principal variation. For a new principal variation to emerge two things must occur; first the value returned by 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
- 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 generally ignoring the second one.

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 α-β 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 β-bound, thereby causing a cutoff.

When predicting where the value of a node v lies relative to the α-β bounds, most pruning methods perform a shallow search, and use the value returned by it to estimate the range in which the actual value of node v is likely to be, should the node be searched more deeply. For example, a 5-ply-deep search is used to predict the outcome of 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 generally 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, look at the tree in Figure 1. The shaded area marks the parts of the tree searched

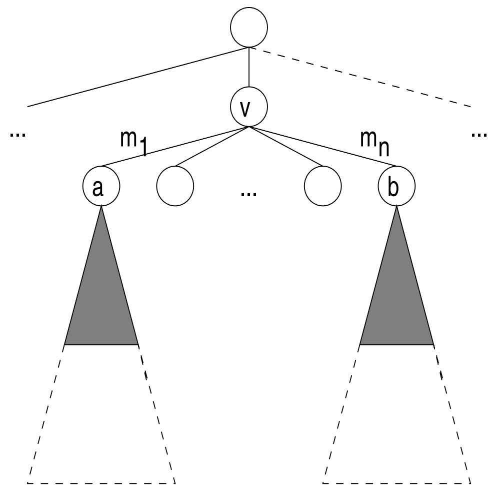

Figure 1: Different risk-assessment.

to decide whether to prune 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 existing methods might be interested in knowing if the move will lead to value that causes a cutoff, that is, the probability

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

But having already searched moves m_1,...,m_{n-1}, and knowing they all failed to cause a cutoff, is a strong indicator that move m_n will also fail to do so, especially because according to the move ordering scheme move m_n is no better than the moves already searched. Instead we should be asking for the probability of move m_n causing a cutoff given that moves m_1,...,m_{n-1} have failed to do so, that is:

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

The values of the moves are not independent of each other, and by assuming they are 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.*

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 playing move m_1 has already been searched. Furthermore, assume that a part of that sub-tree has been pruned off using some forward pruning technique, and that the value returned is greater or equal to the β-bound used at node v (because node v is a cut-node this is what we would expect). Therefore, a β-cutoff occurs and the value returned by move m_1 will backup 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.

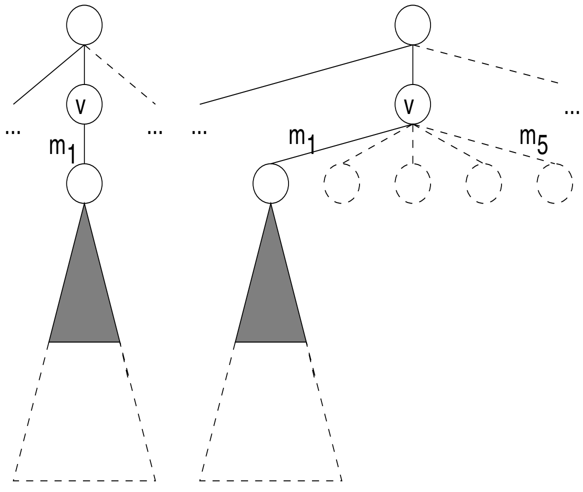

Figure 2: Error propagation.

If the part of the sub-tree that was pruned does not contain a better continuation than the current principal variation, no harm is done and some search effort is saved. The case of interest here is: what happens if the subtree does contain a better line of play? In the first tree given in Figure 2, the error introduced by the pruning propagates all the way to the root, thereby affecting the move decision. Instead of a new principal variation emerging (as would happen if no pruning were done), the line is discarded as being inferior. However, in the second tree the scenario is not so clear. As with the first tree, it is again true that a new principal variation will not emerge. The difference here is that a search without pruning will not necessarily result in a change of principal variation. What might happen, and what indeed often happens in practice, is that even if the first move at a cut node fails to cause a cutoff, one of the alternative moves will do so. This implies that even though we make an erroneous pruning decision in the sub-tree below move m_1, the risk of affecting the move decision at the root is less than in the first tree, because if one of the other moves m_2, ..., m_n causes a cutoff then the move decision at the root will not be affected. 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 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. This is because the necessary prerequirement (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 risk pruning 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 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 achieves most of the savings. Therefore the method of analogies is again a topic worthy of investigation.

3.4 Domain-dependency

It is desirable for a pruning method to be domain-independent. This implies that such methods not use such domain specific knowledge as: is a king in check, or whether a corner square is occupied. Such domain-specific knowledge, if used, should be incorporated in a domain independent way, for example externally state the knowledge in a language that can be compiled and used by the search in a general way.

The only information revealed to the search by the evaluation function is a numerical estimate of how good a problem state is. This clear separation of the search and the problem domain 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 domain 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 in the risk-assessment discussion.

In practice, it is extremely difficult for pruning methods to become domain independent. As we mentioned before 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 and ProbCut 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 completely useless in Othello. Conversely, ProbCut is the pruning heuristic of choice in Othello but has not yet been shown useful in chess or checkers.

4 Existing Forward Pruning Methods

Here we describe some existing forward pruning methods and see how well they meet the factors we have discussed so far.

4.1 The Null-move heuristic

In some games, such as Go, one legal move is to pass; that is, to make no move. This is called a *null-move*. The only change to the game state is which side has to move. In games like chess the null-move is not legal, but even so, it can be useful to assume that a null-move can be played. The idea of a null-move has been known for a long time [Adelson-Velskiy *et al.*, 1975], and is now used in most chess programs. However, the method did not get much attention in the literature until later [Palay, 1983; Goetsch and Cambpell, 1988; Beal, 1990].

When searching a position to depth d, instead of making a legal move in that position a null-move is made. The position is then searched to a depth less than d, most often $d-1$ or $d-2$. If the resulting score is greater than β, a cutoff is made based on the shallow search. The null-move can be applied recursively in the tree. The underlying idea in chess is that it is almost always better to make a move rather than to pass. Therefore, if the score received by giving up a move is still good enough to cause a cutoff, it is very likely that some of the legal moves will also cause a cutoff. Because the position was searched using a shallower search than we would otherwise, considerable effort is saved. In chess it is usually safe to assume that making a move will improve the position, but there are special cases where this is not true. These cases are most likely to arise in the endgame and are called *zugzwang* positions. Thus many chess programs turn off the null-move heuristic upon entering the endgame phase. Some other less used forward pruning schemes like *razoring* and *futility-cutoff* are based on the same idea as the null-move, but these heuristics are more restrictive and can only be applied close to leaf nodes. They can be considered as a special case of the null-move heuristic.

While the null-move heuristic works well in chess, it is inappropriate in many other game-tree domains, either because the null-move is legal in some games, or if it was a legal move then it would be a good move.

4.2 ProbCut

The *ProbCut* [Buro, 1995] heuristic uses shallow searches to predict the result of deep searches. In Othello, where the score of a position generally does not change significantly by searching deeper, this heuristic works well. Therefore, if a shallow search predicts with a high confidence that a deeper search will produce a value outside the α-β window, a node is not expanded further. The heuristic is applied, as formulated, at one pre-determined depth level in the tree.

A confidence interval of how well a shallow search predicts the value of a deeper one is calculated off-line by searching a database of positions that were pre-classified depending on the phase of the game. A separate confidence interval is calculated for each class.

Initial attempts to get this heuristic to work for other games, such as chess, have been unsuccessful. There are several reasons for that. One is that in chess it is difficult to classify positions into equivalence classes based on desired attributes. By creating these classes off-line based on a database of positions, there is a high risk that we are generalizing too much. Another reason is that chess is a more tactical game, thus shallow searches are not as good predictors of deep searches as they are in Othello. We think that this method still has potential in games like chess, but for it to work the heuristic must consider the dynamic features of the search space. For these reasons it is worth studying this topic further.

4.3 Fail-high reductions

Fail-high reductions [Feldmann, 1997] is a new forward pruning heuristic. It is still too early to judge how effective it is, but preliminary experiments in chess show some promise. The idea is to search to shallower depth positions that are seemingly quiet and where the side to move has established a substantial advantage, according to a static evaluation. Apart from the evaluation function, the heuristic requires an additional function that returns a value indicating the quietness of a position. This function returns a value in the range $(0, \infty)$, where a high value indicates a position with many threats. Various positional attributes are used in the scoring, such as is the side to move in check, or a piece is hanging.

More formally: let $e()$ be the evaluation function, and $t()$ be the function that scores threats against the side to move. A node v is called a *fail high node* if $e(v) - t(v) \geq \beta$, where β is the upper-bound used when v is searched. In a null-window search to depth d, fail-high nodes are searched only to depth $d - 1$. The fail-high heuristic is applied recursively in the null-window search.

4.4 Method of Analogies

The method of analogies [Adelson-Velskiy *et al.*, 1975] is a unique search reduction technique. It was implemented in the KAISSA chess program. Given that a move m loses material in position p, then it is possible to find necessary conditions, although they might be insufficient, for move m to also lose material in other positions. Therefore, those positions need not be searched further. The necessary conditions are derived by observing some geometrical relationship between positions and knowing how pieces influence each other.

Although the method offers savings in terms of nodes visited, the gain is largely offset by the time-costs of computing the additional information needed to apply the method. However, since the method was first proposed tree-search techniques and software tools have advanced, possibly making it viable in a simplified form. This method is well worth experimenting with.

5 A Case Study - Variable Null-move Bound

Having looked at some existing forward pruning methods a natural next question to ask is how can they be improved based on the observations we discussed earlier? 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.

5.1 Idea

Goetsch and Campbell [1988] 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 β, but also if the returned value is slightly less. For formally, a cutoff is done 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 the value of t dependent on the evaluation function, we let t vary according to how likely we think an erroneous pruning decision affects the principal variation. Also, 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.

5.2 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 remaining ones. 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 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*[2]. The program uses principal variation search, and null-moves are applied recursively with a search reduction of 2. Several restrictions apply where and when the null-move heuristic is done. 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 3 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[3].

```
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 3: Variable bound null-move cutoff decision.

5.3 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. 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 3, and was chosen based on some trial and error tests. In a future implementation a more appropriate relationship must 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 opening once from the white side and once as black. The match results are shown in Table 1. 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 strength between any two programs, and second, games with actual tournament time controls also

[2] *The Turk* is a chess program developed at University of Alberta by Yngvi Björnsson and Andreas Junghanns.

[3] As the first move expanded is often the most critical one, this implementation compromise might actually be beneficial.

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 1: Match results

must be played. The preliminary results indicate that this method has some potential and is definitely worth refining and experimenting with, but more study is definitely needed to reach a final verdict.

6 Conclusions and Future Work

Important characteristics of forward pruning methods such as *risk-assessment*, *applicability*, *cost-effectiveness*, and *domain-independency* were discussed. We tried to assess how well existing methods address these issues, what the shortcomings are, and where to look for improvements.

Risk-assessment is one factor we think can be improved. We proposed two ways of improving it, by considering:

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

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 if there are alternatives to any potentially overlooked continuation that could preserve the principal variation. To answer these questions the methods have to 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.

Because this work is still in a preliminary stage, there are still many implementation details to be worked out. However, some problems can be anticipated. One is that such methods may require additional computing overhead, which might result in having to restrict their application to the upper part of the tree. Another is the use of a transposition table. Because pruning will be done partially depending on where the move is located in the game-tree, transpositions complicate the issue; a move pruned in one context might not be pruned in another, and vice versa. In might be necessary to store additional information in the transposition table to overcome this lack of consistency.

We feel that there is still considerable scope for improvement in selective depth-first search, and we intend to develop new forward-pruning methods based on the observations mentioned in this paper. Hopefully, some of the existing pruning methods can also be improved. Some preliminary experiments with the null-move heuristic indicate that it is possible.

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.

[Beal, 1990] D. F. Beal. A generalized quiescence search algorithm. *Artificial Intelligence*, 43:85–98, 1990.

[Buro, 1995] M. Buro. ProbCut: An effective selective extension of the alpha-beta algorithm. *ICCA Journal*, 18(2):71–76, 1995.

[Feldmann, 1997] R. Feldmann. Fail high reductions. In *Advances in Computer Chess 8*, June 1997. To appear.

[Goetsch and Cambpell, 1988] G. Goetsch and M.S. Cambpell. Experimenting with the null move heuristic in chess. In *AAAI Spring Symposium Proceedings*, pages 14–18, 1988.

[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.

[Nau, 1980] D. S. Nau. Pathology on game trees: A summary of results. In *Proceedings of the ACM National Conference on Artificial Intelligence*, pages 102–104, 1980.

[Palay, 1983] A. J. Palay. *Searching with probabilities.* PhD thesis, Carnegie-Mellon Univ., Boston, Mass., 1983. See also (1985), book same title, Pitman.

[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.