Empirical Results with Conspiracy Numbers

Norbert Klingbeil
Jonathan Schaeffer

Department of Computing Science
University of Alberta
Edmonton, Alberta
Canada T6G 2H1
403-492-3851

ABSTRACT

McAllester's *Conspiracy Numbers* algorithm is an exciting new approach to minimax search that builds trees to variable depth without application-dependent knowledge. The likelihood of the root taking on a value is expressed by its *conspiracy number*: the minimum number of leaf nodes that must change their value to cause the root to change to that value. This paper describes experiences with the algorithm, using both random and application generated trees. Experiments suggest that more emphasis on breadth, rather than depth, can lead to significant performance improvements. New enhancements to the algorithm are capable of solving 41% more problems than McAllester's original proposal.

Keywords: conspiracy numbers, alpha-beta, heuristic search, game trees, brute-force search, minimax search.

1. Introduction

There are many well-known methods for efficiently searching minimax trees. *Alpha-beta* ($\alpha\beta$) (Knuth and Moore 1975) and *SSS** (Stockman 1979), for example, are elegant algorithms that greatly reduce the search effort required. However, both have fundamental limitations: they are constrained to fixed depth searches and the tree size grows exponentially with the depth of search. Consequently, when used under real time constraints, these search algorithms can only see a limited portion of the tree, a uniform distance from the root.

To overcome this short-sightedness, $\alpha\beta$ is usually enhanced with *selective deepening*. Since the cost of searching the entire tree deeper is exponential, heuristic knowledge is used to select small portions of the tree for further expansion. By selectively deepening the tree in the right places, the search can find solutions at a greater depth of search than is normally possible. The problem, of course, is in deciding which parts of the tree to extend.

Disciples of artificial intelligence have long ignored these so called *brute-force* methods and sought a more knowledge-based approach. Berliner's *B** (Berliner 1979; Palay 1982) is an elegant best-first proof procedure that expands minimax trees to variable depths using knowledge to guide the search. Unfortunately, its application dependent knowledge requirements are its undoing: it is too difficult to reliably determine the *optimistic* and *pessimistic* values required to bound the true value of every node in the tree.

In the past few years, a handful of inventive search algorithms have appeared for building minimax trees to variable depths in an application independent manner. These include enhancements to the basic $\alpha\beta$ framework (singular extensions (Anantharaman, Campbell and Hsu 1990) and null moves (Beal 1989; Goetsch and Campbell 1988)) and entirely new methods of building these trees (Min/Max Approximation (Rivest 1987); Solution Tree and Costs Search (STC) (Schruffer 1988); Conspiracy Numbers). McAllester's *Conspiracy Numbers* algorithm is a novel approach to minimax search (McAllester 1985; McAllester 1988; Schaeffer 1990). Rather than searching to a fixed depth, the algorithm selectively expands nodes in the tree until a specified degree of confidence is achieved in the root value. Confidence is defined by the root node's conspiracy numbers. A conspiracy number is the minimum number of leaf nodes that must change their value (or *conspire*) to cause the root of the tree to change to that value. The algorithm selectively expands nodes in an *application independent* manner, without requiring, for example, extensive leaf node domain-dependent knowledge such as *B** requires. As the algorithm is new, there is little theoretical (Althofer 1988; Lister 1990; McAllester 1988; Schruffer 1988) and experimental (Klingbeil 1989; Klingbeil and Schaeffer 1988; Schaeffer 1990) data on its performance.

This paper describes our experiences implementing and experimenting with conspiracy numbers. McAllester's original paper described a strategy that resulted in continually expanding the left-most son of interior nodes (McAllester 1985). Experiments show this method often results in a program descending to unrealistic depths without any practical chances of success. By adding some breadth to the search, a significant improvement in performance is possible. These ideas have been implemented in a program that solves tactical chess problems. Experimentally, adding additional breadth to the tree and enhancing the algorithm to converge on the best son of the root, not just the best value, resulted in a 41% increase in the number of problems solved.

Section 2 provides an overview of the conspiracy numbers algorithm. In section 3, some of the implementation issues are discussed. Section 4 presents experimental results using random and application generated trees to motivate the need for improving search efficiency. Section 5 discusses alternative search strategies, and their experimental results, implemented in a program to play tactical chess, appear in section 6. Section 7 provides conclusions and future research directions.

2. The Conspiracy Numbers Algorithm

Conspiracy numbers provide a measure of the difficulty of changing the current minimax value of a node. The larger the conspiracy number, the less likely it is that this number of nodes will conspire to change their value, as a result of being searched deeper.

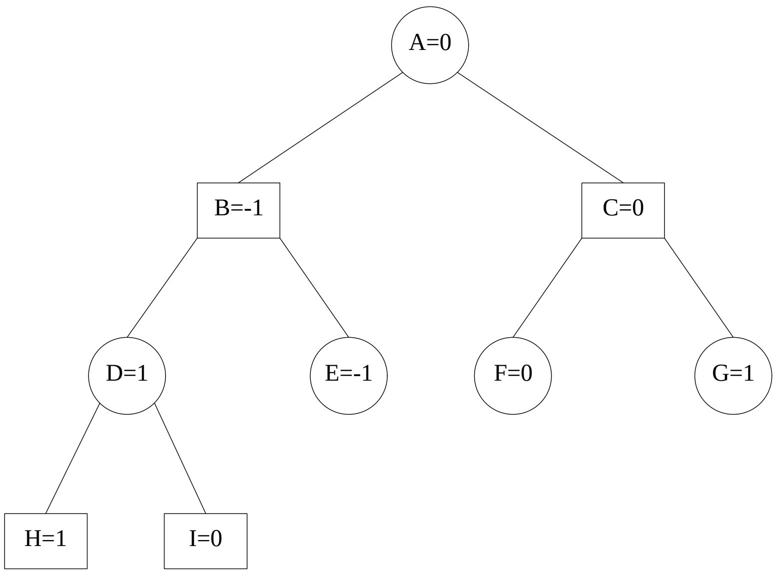

Fig. 1. A Minimax Search Tree.

Fig. 1 illustrates a sample minimax tree, with the value of circled nodes being the maximum of its successors, and boxed nodes, the minimum. How many leaf nodes in the tree have to change their value to cause the value at the root (*Vroot*) to become -2? The simplest way would be if nodes *E* and *F* changed their values to -2 (causing the backed-up values at nodes *B*, *C*, and *A* to change to -2). Another way would be for nodes *G, H*, and *I* to change their values appropriately. Nodes *G*, *H*, and *I* form a *set of conspirators* for decreasing *Vroot* to -2; all three have to *conspire* to achieve this result. Nodes *E* and *F* also form a set of conspirators for decreasing *Vroot* to -2; in this case the *minimal* set. The minimum number of leaf nodes that must conspire to change *Vroot* to a specific value is called the *conspiracy number (CN)* for that value. Fig. 2 shows the conspiracy numbers for node *A*, along with the minimal set of conspirators for several possible node values.

There are simple recursive relations for calculating the conspiracy numbers of a node from the conspiracy numbers of its descendants. In what follows, let m denote the minimax value of a node and v denote the value we would like to change m to.

At a leaf node, changing m to any other value requires a conspiracy of only that node itself, and hence has a conspiracy number of 1. If we do not want to change the node's value, then no conspiracy is required and the conspiracy number is 0. If the leaf node is also a terminal node, then there is no way to change its value and a conspiracy number of ∞ is assigned. Hence, the conspiracy numbers for a leaf node are:

$$CN(v) = \begin{cases} 0 & \text{if } v = m \\ 1 & \text{if } v \neq m \\ \infty & \text{if } \textit{terminal node} \end{cases}$$

At a maximizing interior node, to increase the value to v requires only one son to change its value. If the conspiracy number for each son has already been calculated, then the minimum number of conspirators required to increase the node to v, $\uparrow CN(v)$, is just the minimum number of conspirators to increase one of the sons to v. This yields the relation:

$$\uparrow CN(v) = \begin{cases} 0 & \text{for } \textit{all } v \leq m \\ \underset{\textit{all sons } i}{MIN} \uparrow CN_i(v) & \text{for } \textit{all } v > m \end{cases}$$

To decrease the node's value to v, $\downarrow CN(v)$, requires all sons whose value is greater than v to decrease their value to v. Given the minimal set of conspirators for decreasing each son to v, *all* members of each of these sets must conspire together to decrease the node's value to v. The sum of the son's conspiracy sets is the total number of conspirators required to change this node. Therefore:

$$\downarrow CN(v) = \begin{cases} 0 & \text{for } \textit{all } v \geq m \\ \sum_{\textit{all sons } i} \downarrow CN_i(v) & \text{for } \textit{all } v < m \end{cases}$$

For a minimizing interior node, the following dual relations apply:

A terminal node is a leaf node that has a fixed, accurate value and therefore does not have any successors.

$$\uparrow CN(v) = \begin{cases} 0 & \text{for } all\, v \le m \\ \sum_{all\ sons\ i} \uparrow CN_i(v) & \text{for } all\, v > m \end{cases}$$

$$\downarrow CN(v) = \begin{cases} 0 & \text{for } all\, v \ge m \\ \underset{all\ sons\ i}{MIN} \downarrow CN_i(v) & \text{for } all\, v < m \end{cases}$$

Fig. 3 shows the conspiracy numbers for node values in the range [-3, 3] at each node. For brevity, $\downarrow CN(v)$ and $\uparrow CN(v)$ are merged into one vector, since $\downarrow CN(v)$ ($\uparrow CN(V)$) is zero for those node values where $\uparrow CN(v)$ ($\downarrow CN(V)$) is non-zero. It is worth noting that if $v < w$ then $\uparrow CN(v) \le \uparrow CN(w)$ and $\downarrow CN(w) \le \downarrow CN(v)$. Also, given a set of conspirators for changing the value of a node to *v*, ($v \ne m$), this same set can conspire to change the node to any value between *m* and *v*.

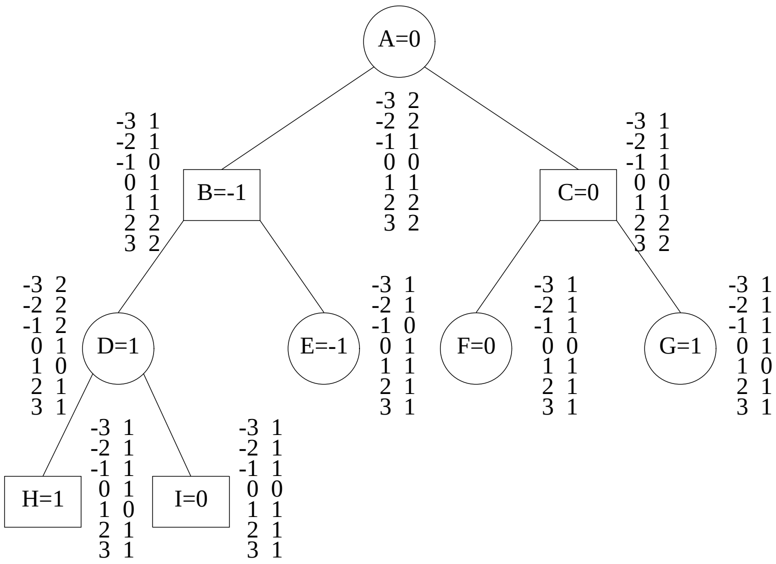

Fig. 3. Conspiracy Numbers.

Fig. 4 provides an outline of the conspiracy numbers search algorithm. The algorithm takes two parameters: a *range parameter* (Δ) which specifies the accuracy to which the root value is to be determined, and a *conspiracy threshold (CT)* that defines the degree of confidence required in the root value. The algorithm returns the minimax value of the root (*Vroot*). Search trees are grown to gather information for increasing the accuracy of the root value.

```
CN_Search( Δ, CT ) {
     compute conspiracy numbers at root node
     compute Vmin, Vmax
     while ( Vmax - Vmin > Δ ) do {
          choose to rule out Vmax or Vmin
          select appropriate leaf
          expand leaf node
          compute minimax values along search path
          compute conspiracy numbers at root node
          compute Vmin, Vmax
     }
     return( Vroot )
}
```

Fig. 4. Conspiracy Numbers Algorithm.

A range of likely values for the root node is maintained; *Vmin* being the lower bound and *Vmax*, the upper bound. The algorithm repeatedly extends the search tree in an attempt to narrow this range. To do this, two strategies are used: *rule_out_Vmin*, attempting to increase the lower bound, and *rule_out_Vmax*, attempting to decrease the upper bound. Having selected a strategy (described later), an appropriate leaf node is chosen for expansion (searched one move, or *ply*, deeper). After the node is expanded, the minimax values and conspiracy numbers at the root are recomputed. The search terminates when the range of likely root values ($Vmax - Vmin$) achieves the desired accuracy (Δ).

Conspiracy numbers at the root node, combined with a conspiracy threshold, are used to compute a range of likely root values. The conspiracy threshold specifies the minimum number of conspirators required before it is considered unlikely a node can take on that value. The range of likely root values is denoted by $[Vmin, Vmax]$ where *Vmin* is the lower bound value, *Vmax* is the upper bound value, and *Vroot* is a member of this range. Obviously, the higher the *CT*, the greater our confidence is that values outside the *Vmin*/*Vmax* range are unlikely. *Vmax* is simply defined as the maximum possible value for which there are less than *CT* conspirators:

$$\uparrow CN(v) < CT \text{ and } \uparrow CN(v+1) \geq CT.$$

Similarly, *Vmin* is the minimum value for which the root has less than *CT* conspirators:

$$\downarrow CN(v) < CT \text{ and } \downarrow CN(v-1) \geq CT.$$

In Fig. 3, if $CT = 2$, then the range of likely root values would be [-1,1].

The search continues until the range of likely root values has been narrowed to the point where $Vmax - Vmin \leq \Delta$. In practice, we usually continue searching until all but one root value has been ruled out (i.e. $\Delta = 0$). At this point, further search is unlikely (as defined by the setting of *CT*) to change the root's minimax value. When $\Delta = 0$ and the algorithm's condition for terminating the search evaluates to true, we often say that the algorithm has *converged* on a single root value.

It is important to note that the conspiracy numbers algorithm is not *guaranteed* to converge to the correct minimax value. Our confidence in the search result can be measured by the conspiracy threshold: the higher the threshold, the greater the confidence in the value. Not surprisingly, the higher the threshold the greater the amount of work required to narrow the range of likely values.

Given a range of likely root values, how do we rule out all but one of them? The obvious way is to rule them out one by one, starting with either *Vmax* or *Vmin*. To rule out *Vmin* or *Vmax*, the algorithm tries to increase the corresponding conspiracy number to at least *CT*. This is done by "proving" that a member of the minimal conspiracy set will not conspire with the other members of the set to help change the value of the root node to either *Vmin* or *Vmax*. During each step of the tree growth procedure, the algorithm must choose to either *rule_out_Vmax* (try to demonstrate that *Vmax* is unlikely) or *rule_out_Vmin* (try to eliminate *Vmin*). Faced with these two alternatives, McAllester proposed that the algorithm always attempt to rule out the value furthest from *Vroot*. If both are equidistant from the root value, then arbitrarily choose to *rule_out_Vmin.*

Consider the case where *rule_out_Vmax* has been selected. A leaf node from the minimal set of conspirators must be found to search one ply deeper (or *expanded*). To find this node, the algorithm descends from the root using the following procedure:

a) Maximizing Node:Only one successor node must increase its value to *Vmax* for the root node to do likewise. The branch most likely to do this is the one requiring the least number of conspirators to increase it to *Vmax*. After computing $CN(Vmax)$ for each successor, choose the successor node requiring the minimum conspirators. If more than one branch has the minimum, arbitrarily choose the left-most one.

b) Minimizing Node:Here there may be many descendant nodes that have to increase their value to increase this node to *Vmax*. Each such branch contains conspirators which together form the minimal set of conspirators to increase this node to *Vmax*. Again the algorithm could choose to traverse any of the appropriate branches, but the left-most one is arbitrarily chosen.

Having reached a leaf node, that node is then extended one ply deeper. Since each descendant may yield a favorable or unfavorable assessment, as determined by an application dependent routine that assesses the quality of the resulting node, the descendants are ordered according to the results of their evaluation. By putting the more favorable descendants to the left, this increases the chances that the left-most descendant is the best, justifying the above choices.

What is being accomplished by expanding this node? If we are successful at increasing the value of this node to *Vmax*, then the number of conspirators in this set has been decreased by one and therefore other members of the set can be expanded to see if they will conspire successfully. If the value is less than *Vmax* and the expanded node is minimizing, then we may have been successful at increasing the number of conspirators at the root (i.e. increased the minimal set of conspirators). The number of conspirators may have reached *CT*, resulting in a narrowing of the range of likely values at the root. At a maximizing expanded node with a value less than *Vmax*, nothing has been accomplished towards ruling out *Vmax*.

A dual strategy exists for ruling out *Vmin*. This tree growth procedure was McAllester's original proposal. Considering Fig. 3 (with $CT = 2$), *rule_out_Vmax* would choose to expand node *E*, while *rule_out_Vmin* would expand *F*.

3. Implementation Issues

There are a number of issues that arise when implementing conspiracy numbers. Following is a brief discussion of the most important ones. Further details can be found in Klingbeil (1989).

3.1. Space-Time Trade-offs

Unlike $\alpha\beta$, with its depth-first, left-to-right traversal of a tree, the order in which leaf nodes are chosen for expansion by the conspiracy numbers algorithm does not follow a regular pattern. At times, search effort is concentrated in small localized areas but at other times the algorithm moves about the tree in a more-or-less random manner. Consequently, it is necessary to save the search tree in memory.

Disregarding the cost of leaf node evaluation (an application-dependent function), most of the execution time of the algorithm is spent calculating the conspiracy numbers. To compute a node's conspiracy numbers requires information from the entire sub-tree below it. If conspiracy numbers for every value are stored at every node in the tree, then they can be quickly retrieved without additional calculation. Otherwise they must be re-calculated each time and the overhead of traversing the tree quickly overwhelms everything else. The problem then becomes one of a space-time trade-off. To investigate this trade-off, three implementation versions were tested: *Basic* (save no conspiracy numbers in the search tree), *Storage Intensive* (save all), and *Compromise* (save some).

The Basic implementation represents an attempt to use minimal storage. Only the search tree is saved and therefore the conspiracy numbers must be computed whenever needed. As the search tree grows, a significant effort is devoted to computing the conspiracy numbers.

The Storage Intensive implementation attempts to minimize execution time by saving maximal information in the tree. Conspiracy numbers are saved for each possible node value at all nodes. This information must be updated after each expansion. However, only those conspiracy numbers at nodes along the search path need to be updated. Unlike the Basic version, this implementation requires a limit on the size of the range of node values. Obviously, when the range size becomes large, its performance deteriorates because there are more conspiracy numbers to update after each expansion, and memory requirements increase significantly.

The Compromise implementation tries to find a compromise on storage usage. It saves conspiracy numbers for only the most important values at each node: $Vmin$, $Vmin - 1$, $Vmax$,

$Vmax + 1$. This has the advantages of reducing the cost of updating conspiracy numbers along the search path, the amount of re-computation done, and the memory requirements for storing large trees. However, its success hinges on the assumption that *Vmin* and *Vmax* remain fairly stable. When they change, the entire tree may have to be traversed to recompute the conspiracy numbers for the new values.

The above three versions have been implemented (Klingbeil 1989). Not surprisingly, the Basic version is significantly inferior and continues to deteriorate as the tree size increases. Both the Storage Intensive and Compromise implementations are competitive with respect to execution time. In deciding which implementation to choose, the most important factor is the range size. For the Storage Intensive version, the larger the range of node values, the more storage required and the greater the effort needed to maintain all the conspiracy numbers. In contrast, the Compromise version remains stable since it always maintains a constant amount of information. This is illustrated in Fig. 5, which shows some experimental data run on a Sun 3/50.

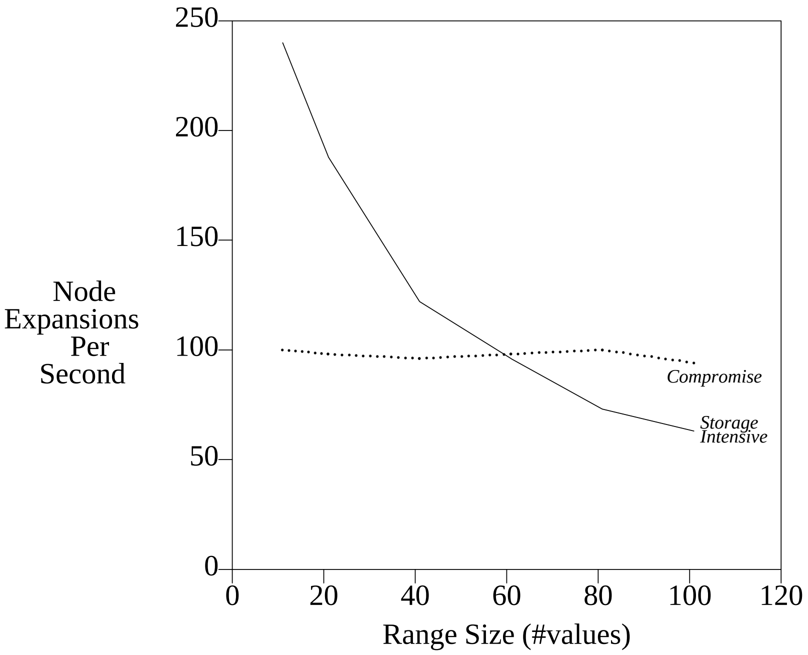

Fig. 5. Compromise vs Storage Intensive.

3.2. Controlling the Conspiracy Threshold

When calling conspiracy numbers, the driver routine must supply it with a conspiracy threshold (*CT*). Two factors influence the choice of CT: the higher the threshold the greater our confidence in the result but the longer it takes the algorithm to converge. Unfortunately, it is difficult to balance these factors since, for a given threshold, one cannot predict how long it takes for convergence to occur and it is difficult to quantify the degree of confidence. If the algorithm is run under time constraints, one would like the search to be completed to the highest possible threshold subject to the constraints.

There is an approach that allows the degree of confidence in the root value to increase over time. It starts out by assigning CT a low value such as 2. Once the algorithm converges, the threshold is bumped up to 3, and continues so forth until a specified stopping criteria, such as a time limit, is reached. Whenever the search is stopped, the previously completed threshold provides an answer as well as the degree of confidence in that answer.

This method is analogous to iterative deepening that is used for a variety of depth-first search algorithms (e.g. $\alpha\beta$ (Slate and Atkin 1977) and A* (Korf 1985)).

3.3. Depth of Search

Since the conspiracy numbers algorithm always explores a single search path at a time, without consideration of the alternatives, problems with search depth can result. First, the program can explore a line to an arbitrary depth, fruitlessly trying to discover something that isn't there. Second, most game trees are really directed graphs. Consequently, a search path containing repeated states will cause an infinite loop.

The danger of an unrestricted search depth is that the search may be unable to adequately explore certain parts of the tree in a given amount of time. The consequences can be two-fold. First, there may not be enough time to explore other branches at the root that may actually turn out to be better. Second, effort may be concentrated on a line of play that appears promising, but the program runs out of time exploring all the opponent's alternatives, leaving open the possibility of a large error being propagated to the root. A solution that is not entirely satisfactory is to put a depth restriction on the search, not allowing the expansion of nodes beyond a given fixed depth.

Repeated game states along a single path in the tree can cause an infinite loop. To show how this can happen, suppose the algorithm is exploring a line of play and a new leaf node is created which represents the same game state as an ancestor node. For convenience, call the ancestor node *A* and the leaf node *B*. Suppose that node *B* is explored further later in the search. The subtree that will be grown below *B* will be identical to the subtree that was grown below *A*. The algorithm will repeatedly follow this path, with no convergence possible. The solution is to detect cycles along a path by checking each expanded node to see if it is identical to any prior state along the current search path.

Nodes where the search is terminated, either by a depth restriction or a repeated state, are treated as terminal nodes. They are assigned the maximum possible conspiracy number, preventing *rule_out_Vmin* and *rule_out_Vmax* from every considering them again.

3.4. Transposition Tables

In the case where the trees are really directed graphs, transposition tables have been useful for a variety of search algorithms (e.g. $\alpha\beta$ (Slate and Atkin 1977)). These tables are used to save previously seen nodes in the tree, in the event that they occur in different parts of the tree. In such cases, rather than re-generating that portion of the tree, a pointer can direct the node to be expanded to the previously seen state. This helps reduce the size of the search tree and improves the performance of the program. However, the data structure now becomes a graph, instead of a simple tree, where one node may have several parents.

Extending the conspiracy numbers algorithm to incorporate this data structure is conceptually simple, but does pose some implementation difficulties. Instead of having only to back up minimax values and conspiracy numbers along the current search path, they must be backed up along all paths that exist between the expanded node and the root node. This complicates the design of the tree data structure, and requires additional storage for maintaining back pointers.

4. Initial Experience

Experiments were conducted with the implementation described in section 3 to discover interesting properties of the trees built by conspiracy numbers. McAllester's original version of the algorithm was used for experiments with both random and application (chess) generated trees. Iterative deepening without transposition tables was used. The search depth was restricted to a maximum of 20 ply.

4.1. Searching Random Trees

In considering what factors can affect tree size and the rate of convergence, several come to mind:

a) Branching Factor:Obviously, for larger branching factors, one would expect larger trees. This is generally true, but anomalies can occur; larger branching factors increase the probability that enough conspirators can be found. Consider, for example, a case where $CT = 2$ and a node can take on one of four randomly generated values, When the branching factor is four, odds are only one conspirator will be found and at least one more expansion will be necessary. When the branching factor is six, there is a better chance that both conspirators will be found, and additional expansions will be unnecessary.

b) Range of Values:The greater the range of values that a node can be assigned, the more work the algorithm may have to do converge. Larger value ranges increase the initial likelihood that *Vmin* and *Vmax* are far apart. This may result in additional work being needed for them to converge.

c) Successor Ordering:Since McAllester's original version prefers to consider the left-most son of a node, the greater the success in getting the best son in the left-most position, the faster the algorithm will converge.

Fig. 6 and Fig. 7 illustrate these points. The results were computed using randomly generated trees using the method described in Marsland, Reinefeld and Schaeffer (1987). For each parameter combination, the root value was fixed and 20 trees were randomly generated and searched. The larger the trees searched, the higher was the standard deviation of the results. The

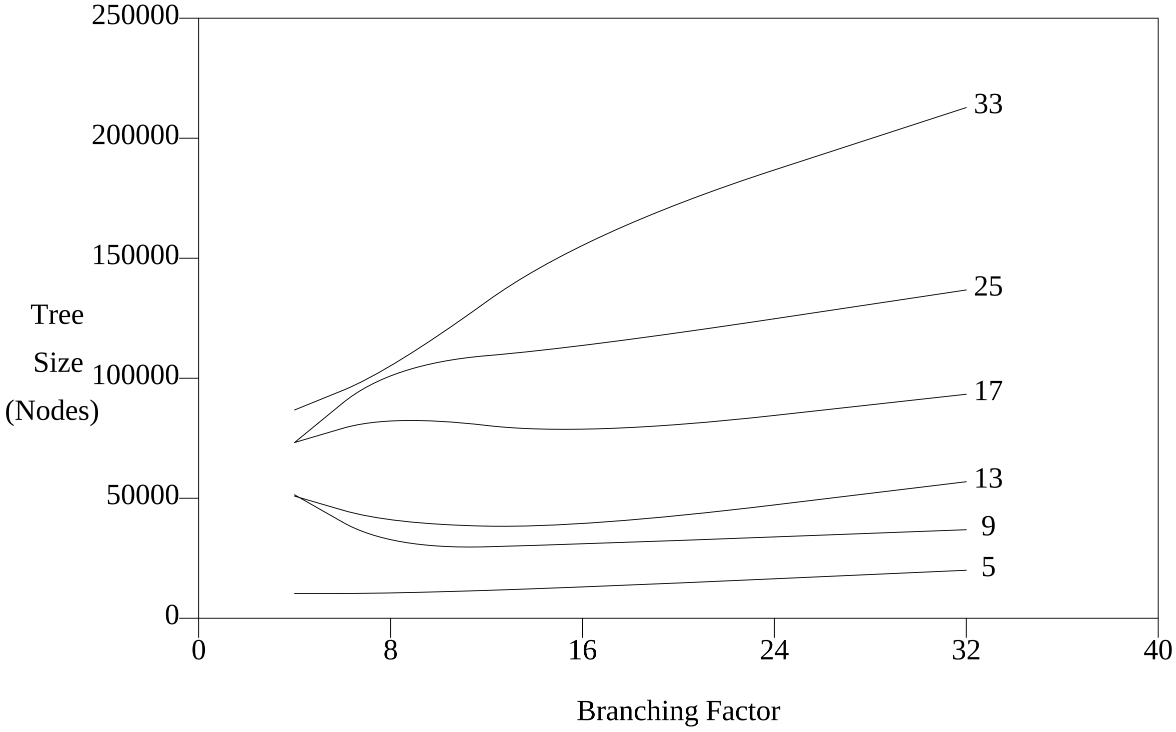

Fig. 6. Effect of Range Size on Tree Size (Ordering 100%).

graphs presented in this section are indicative of the algorithm's performance and should not be taken as absolute measures. Experiments were conducted for branching factors of 4, 8, 16, and 32, value ranges of 5, 9, 13, 17, 25, and 33 and probability that the best son was in the left-most position of 50, 75, and 100%. In Fig. 6, the best son is in the left-most position 100% of the time, showing how an increased range of values increases tree size. Fig. 7 holds the range size constant at 33 and illustrates how improved ordering reduces tree size.

Although not formally proven, it appears that the rate of convergence is strongly tied to the accuracy of the evaluation function. If an expanded node results in the value of the parent changing infrequently, then convergence is enhanced since the sizes of the conspiracy sets being examined may increase. If deeper search changes a node's value, then the algorithm may jump around, trying to prove a continually changing value. A formal proof of this conjecture remains an open problem.

4.2. Searching Chess Trees

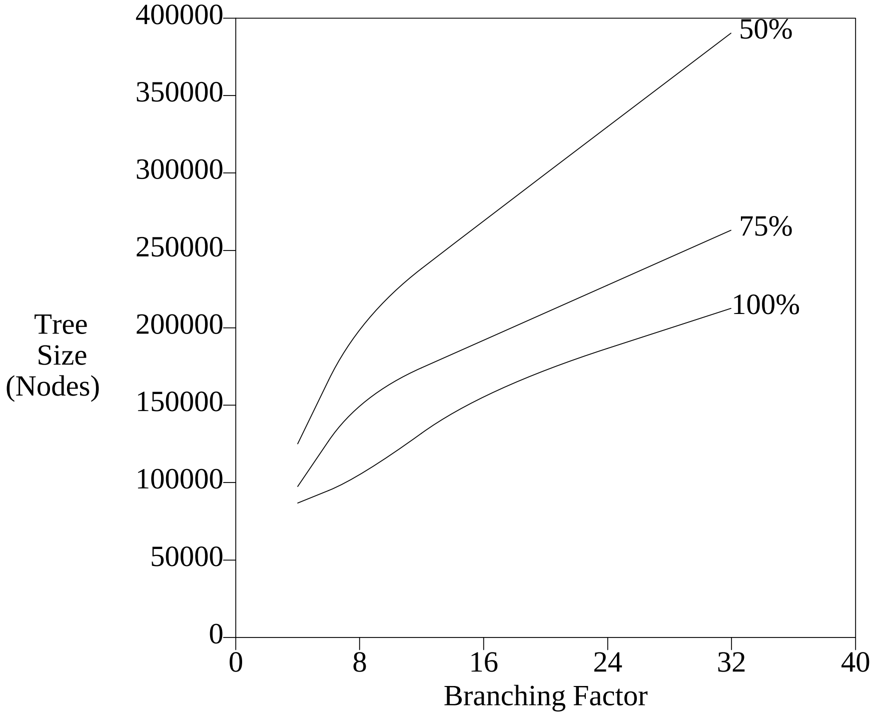

Fig. 7. Effect of Ordering on Tree Size (Value Range = 33).

The conspiracy numbers algorithm has been implemented in the chess program *Phoenix* (Schaeffer 1986). The program has been modified to do only tactical searching, looking for the moves that win the most amount of material (value of the pieces on the board) without, in the event of a tie, using any positional information to differentiate them. The advantage of tactical chess is that only a small range of possible values are required. The program used iterative deepening without transposition tables.

The performance of this program was tested on a problem set of 95 chess positions taken from the *Encyclopedia of Middlegame Combinations* (Chess Informator 1985). The 300 positions in the *Win At Chess* (Reinfeld 1945) problem set is often used for the benchmarking of tactical chess programs. Unfortunately, most of problems are easily solved using small (5-ply) $\alpha\beta$ searches. Some results using this test set can be found in Klingbeil (1989) and Schaeffer (1990). The problems from the *Encyclopedia of Middlegame Combinations* are challenging both for computers and humans.

A frequent competitor in the World Computer Chess Championship.

Initial results for McAllester's original algorithm in tactical chess were disappointing. The $\alpha\beta$ algorithm consistently out-performed it in both the number of problems correctly solved and the speed at which they were solved (Klingbeil 1989; Schaeffer 1990). Analyzing the trees built by conspiracy numbers led to the observation that the algorithm consistently explored one branch until there were no appropriate conspirators below that son. When there were no conspirators below that son, and there were alternative branches with appropriate conspirators beneath them, then the algorithm explored the left-most one of these alternative sons. Often, however, these alternative sons do not get much (if any) chance for consideration of being explored. This results in circumstances where about 80% of the nodes below a certain node can be found below just one son as is illustrated in Fig. 8.

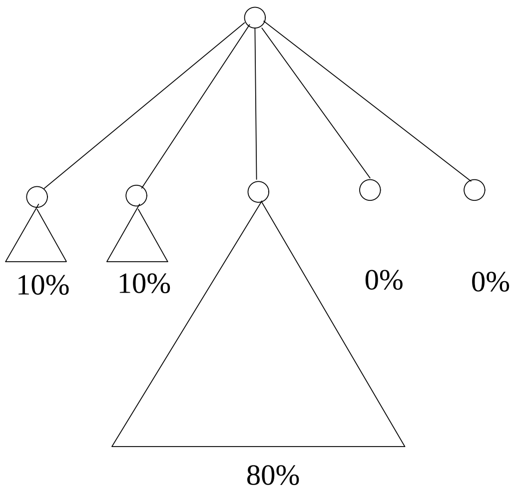

Fig. 8. Distribution of Nodes.

A more detailed examination of how the search effort was being distributed was done on 95 problems. Of the 42 problems that were solved, 39 had the son corresponding to the correct move explored much more often than the other sons. For the remaining 3 problems, the exploration effort was spread more evenly across several sons of the root node and the son corresponding to the correct move was not explored the most.

Of the 53 problems that were incorrectly solved, 31 had the son corresponding to the correct

This is approximately how it looks like beneath nodes on level 0 and 1 of the search tree.

move appearing too far down the list, because of a low initial evaluation, which meant that this son had to wait a long time before being considered for exploration. In fact, in the majority of these 31 problems, the correct move was hardly explored at all! In 19 other problems, the correct move was explored much more than the other moves. However, some of the other moves had minimax values that were equal to or higher than the minimax value of the correct move. Here the problem appears to be that the algorithm may not have put any effort into trying to lower the other moves' minimax values. For the remaining 3 problems, the search effort was on moves that appeared further down in the list than the correct move.

There are at least two methods for reducing the effects of the above problems. The first is to enhance the static evaluation function to provide better ordering of the sons. A second, and more significant improvement, would be to modify the algorithm so that it spreads the exploration effort more evenly across the alternative sons (i.e. introduce more breadth at the nodes).

5. Improving the Search Strategy

McAllester's original procedure has two inherent problems. First, the search has a depth-first behavior that has difficulty finding the correct answers quickly. Both *rule_out_Vmin* and *rule_out_Vmax* keep choosing the same search path, forcing the algorithm to go deeper into the search tree, until the desired information is obtained. Second, McAllester's proposal does not necessarily result in the "ideal" node being picked for expansion. For example, given several minimal conspiracy sets, it might make sense to first test each set to determine which one is more likely to achieve the desired result. If we had a bit more knowledge about the probability of a leaf node changing its value, we could better assess which minimal conspiracy set offered the best chance for success.

There are primarily four areas in the algorithm which influence the choice of leaf node to be expanded: 1) the way we choose to order descendants of a node, 2) the value that has been chosen to be ruled out, 3) the procedure for traversing the search tree, and 4) the value of the current conspiracy threshold. The ordering of descendants is application dependent and is not addressed here. New ideas for enhancing 2), 3), and 4) are presented in this paper with experimental results in the next section.

5.1. Adding Breadth

A solution to the above problems is to avoid exploring one minimal conspiracy set to the exclusion of others. A scheme which cycles among the alternative sets allows the search to consider them in an unbiased manner, providing more of a breadth-first than depth-first exploration. An obvious choice is to use a round-robin scheme for selecting among alternate branches; the search would always start at the branch immediately adjacent to the one that was previously chosen and cycle around until an appropriate branch is found. Adding additional breadth to the search spreads the exploration effort more evenly throughout the tree and can increase the likelihood of finding the solution quickly.

To help describe the various types of round robin we intend to use, new terminology will be introduced. When *rule_out_Vmax* visits a MAX node, only one son has to increase its value to *Vmax* to cause the parent to do likewise. Using a loose analogy to AND-OR trees, we call these *OR* nodes since only one branch needs to successfully conspire to affect the parent. At MIN nodes, one or more sons have to increase their value to *Vmax* to cause that node to do likewise. Again using the analogy, since all the sons must conspire to affect the parent, we call these *AND* nodes. *Rule_out_Vmin* views MIN and MAX nodes oppositely; MAX nodes are *AND* nodes and MIN nodes, *OR* nodes.

Breadth can be added to the search by simply altering the search strategy at OR and AND nodes. One simple modification is to have a round robin approach for selecting among alternative branches to descend at either OR nodes, AND nodes, or both. At OR nodes, several sons may have the minimum number of conspiracy nodes beneath them. Round robin at OR nodes changes the search behavior so that successive expansions cycle between all nodes having the minimum number of conspirators. At AND nodes, all sons having conspirators beneath them must eventually be explored. Hence, one would expect there to be no advantage to be gained by spreading the exploration effort across these sons.

Alternate node selection strategies could apply limited forms of round robin. For example, if the ordering of descendants of expanded nodes is good, round robin between the first 25% of the sons, for example, might be a better way of obtaining breadth but concentrating it where it appears more likely to succeed. Another strategy could be to apply round robin for only a fixed number of rounds and then revert to another rule (e.g. McAllester's original rule). This would give each eligible son an equal chance of reducing its conspirators and, after this, concentrating the exploration effort on only the best of the sons.

There are many breadth and depth variants that can be tested. The strategies implemented include:

1) BASIC:McAllester's original proposal.

2) OR RR:applying round robin at the OR nodes and McAllester's rule at the AND nodes.

3) AND/OR RR:applying round robin at both OR and AND nodes.

4) OR PRR:applying a partial round robin at OR nodes where only the first 25% of the sons will be considered in the round robin. If the only alternative branches are not in the first 25% then McAllister's rule applies. McAllester's rule also applies at AND nodes.

5) MB (minimal breadth):applying round robin at OR nodes, but limiting it to one round and then reverting back to McAllester's rule. McAllester's rule applies at AND nodes. This variant provides minimum round robin breadth.

5.2. Likelihood Strategies

What is the "ideal" node to expand? Suppose we are interested in finding a member of a minimal conspiracy set to expand, but there are several such sets. If we could compute the probability that each member of a set has of changing its value appropriately, then we can compute which minimal conspiracy set has the highest probability of success. Instead of assigning a single score to a leaf node, we could use an estimation function that provided a probability distribution indicating the likelihood that a node could change to a specified value. From this, we could compute the probability of a minimal conspiracy set successfully achieving a certain value at the root.

Our definition of the ideal node to expand is the one that has the highest probability of changing its value appropriately and is a member of the minimal conspiracy set that has the highest probability of changing the root value to its desired value.

There are several reasons why we may not want to work with probabilities. Probabilities usually imply the use of floating point arithmetic and the necessity of saving floating point values at nodes. Also, depending on how knowledgeable the estimation function is, the probabilities are still just educated guesses at best. Therefore, to reduce computing and storage costs, integers were used to approximate these probabilities.

The likelihood of a leaf node achieving a certain value will be given by an application-dependent function; the larger the returned integer, the greater the perceived chances of attaining that value. To compute the *likelihood* of a set of leaf nodes conspiring successfully to change the root value, we will approximate the true probability by summing the likelihoods of each son changing their value appropriately. Comparing likelihood values between different sized conspiracy sets is based on the average likelihood value per member. Normally, probabilities would be multiplied together. However, even with moderate branching factors, the resulting answer will quickly use up all the machine's precision.

Simple recursive relations can be used for computing likelihood values of a node from the likelihood values of its descendants. In what follows, let m denote the minimax value of a node, v the value we would like to change m to, and j an interior node. Let $\uparrow LIKE(v)$ denote the likelihood of increasing a node's value to v and $\downarrow LIKE(v)$, of decreasing it to v.

At a maximizing interior node, to increase the value to v requires only one son to change its value to v. The number of conspirators to change the node to v is just the minimum number of conspirators to increase one of the sons to v. Since the algorithm is only interested in those sons that have this minimum number, the likelihood of increasing the node to v is simply the maximum likelihood of increasing one of these sons to v. This yields the relation:

$$\uparrow LIKE_j(v) = \begin{cases} 0 & \text{for } all\ v \le m \\ \underset{all\ sons\ k\ with\ minimum\ \uparrow CN(v)}{MAX} \uparrow LIKE_k(v) & \text{for } all\ v > m \end{cases}$$

To decrease the node's value to v requires all sons whose value is greater than v to decrease their value to v. Given the minimal set of conspirators for decreasing each son to v, all members of each of these sets must conspire together to decrease the node's value to v. The likelihood of this occurring is simply the sum of the likelihoods of each of these sons decreasing their value to v. Therefore:

$$\downarrow LIKE_j(v) = \begin{cases} 0 & \text{for } all\ v \geq m \\ \sum_{all\ sons\ k\ whose\ m\ >\ v} \downarrow LIKE_k(v) & \text{for } all\ v < m \end{cases}$$

Dual relations exist for minimizing interior nodes.

How will the search routines use these likelihood values? At OR nodes, the algorithm will traverse down the branch that has the lowest conspiracy number for changing that node to the desired value ($V_{\min}$ or $V_{\max}$). If there are two or more branches that have the lowest conspiracy number, then the branch with and highest corresponding likelihood value is chosen. At AND nodes, the search will descend down the branch that has a non-zero conspiracy number and the highest corresponding average likelihood value per conspirator. Note that this procedure will not always find the leaf node with the highest likelihood value if it is a member of a larger conspiracy set. However, it should always pick a leaf node with one of the higher likelihood values from such sets. Overall though, this procedure does approximate our objective of finding the *ideal* leaf node to expand. We will call this enhancement simply *Likelihoods.*

5.3. Best Move Cut-Offs

A problem with conspiracy numbers is that its focus is on finding the most accurate root value and not on finding the best move at the root. This leads to situations where a best move is already obvious, but the algorithm wastes valuable time because it has not yet converged. To understand this problem more, we will first look at how such a situation can be detected and why it is felt exploration effort is being wasted when this occurs.

Suppose that a local *Vmin* and *Vmax* can be computed for each son of the root node using the current conspiracy threshold. Assuming that one of these son's local *Vmin* is greater than all the other sons' local *Vmax* (i.e. this son's *pessimistic* value is greater than the *optimistic* value of all other sons), clearly this son would be the best. It can easily be shown that the values for the global *Vmin* and *Vmax* are the same as the values of the best son's local *Vmin* and *Vmax*. Consequently, the search routines will only pick leaf nodes from under the best son to expand. The best son will continue to be explored until the algorithm converges. In other words, the focus of the algorithm has been shifted to finding an accurate value for the best son of the root, not which son is best.

It is probably wasteful exploring the best son any further. Instead it may be better to try to prove that this is the best son for a higher threshold. At higher thresholds, the range of likely node values of some sons may overlap the range of the best son, meaning other sons might eventually be considered for exploration.

Currently, convergence is defined as occurring when $Vmin = Vmax$. This condition will be loosened to allow convergence also to occur when one son of the root has a pessimistic value that is at least as good as all the other son's optimistic values. When this best son scenario occurs and $Vmin \neq Vmax$, search for the current threshold will be regarded as complete. This enhancement, allowing for more breadth at the root node, is called *Best Move Cutoffs.*

6. Experimental Results

The enhancements of the previous section have been implemented in *Phoenix*. The following results were obtained using the *Encyclopedia of Middlegame Combinations* test suite. Each search variant was run on each problem until a search tree of 300,00 nodes had been built (approximately 30 minutes on a Sun 3/50).

Fig. 9 compares the performance of McAllester's original algorithm with the four round robin variants. The data set is large enough to clearly demonstrate the advantage of adding some breadth to the search. However it is not large enough to adequately distinguish between the different round robin variants. In comparing the problems solved by each, all variants were able to

---

Note that it is possible for *Vmin* and *Vmax* to diverge. This could lead to a state where other sons may be considered for exploration. However, the probability of *Vmin* and *Vmax* diverging is usually small.

solve 32 common problems, while 43 were not solved by any. Of the remaining 20, 1 required the depth possible only through McAllester's version, and 4 required the breadth of the round robin schemes. Among only the round robin versions, 39 problems were solvable by all while 44 were not solvable by any.

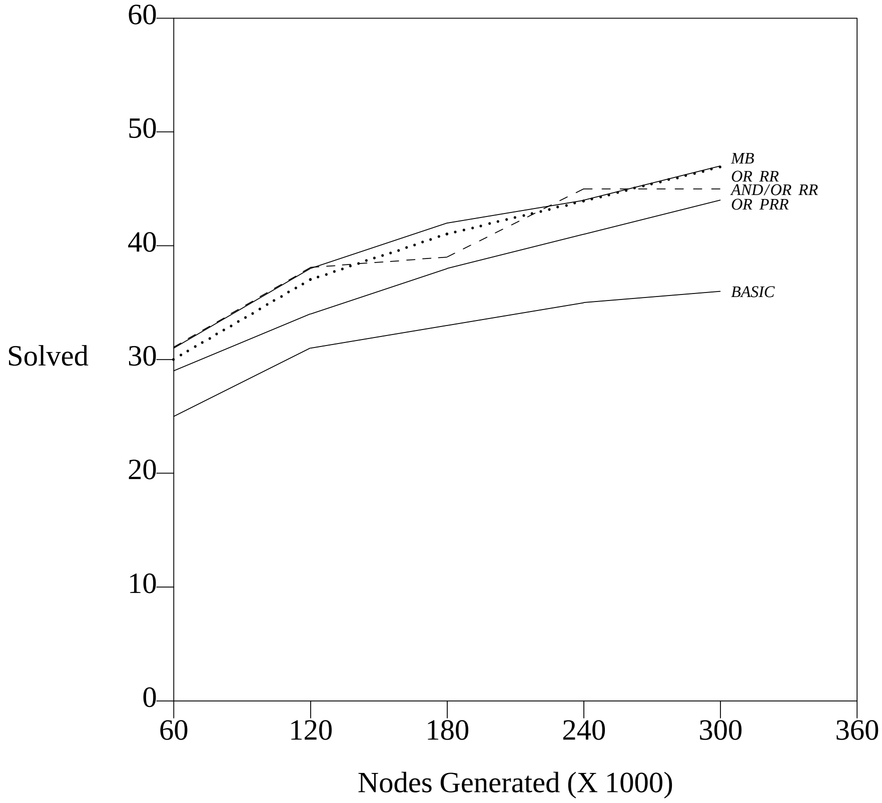

Fig. 9. Round Robin Version Results.

To experiment with the likelihood method, a simple likelihood function for leaf nodes was invented. It was based on the heuristic that it is easier to change a node's value a little than a lot. The values that were used at the leaf nodes were in the range [0,8]. The equation for $\uparrow LIKE(v)$ at leaf nodes was:

$$\uparrow LIKE(v) = \begin{cases} 8 \text{ if } v = m & \\ 7 \text{ if } (v - m) > 0 & \text{and } (v - m) \le 0.10 * Rsize \\ 5 \text{ if } (v - m) > 0.10 * Rsize & \text{and } (v - m) \le 0.20 * Rsize \\ 3 \text{ if } (v - m) > 0.20 * Rsize & \text{and } (v - m) \le 0.35 * Rsize \\ 1 \text{ if } (v - m) > 0.35 * Rsize & \text{and } (v - m) \le 0.50 * Rsize \\ 0 \text{ if } otherwise & \end{cases}$$

where *Rsize* is equal to the size of the range of legal node values and *m* is equal to the node's current minimax value. The equation for $\downarrow LIKE(v)$ is similar. The numbers in these equations should be taken as initial numbers only; they are based strictly on intuition and not on experimentation.

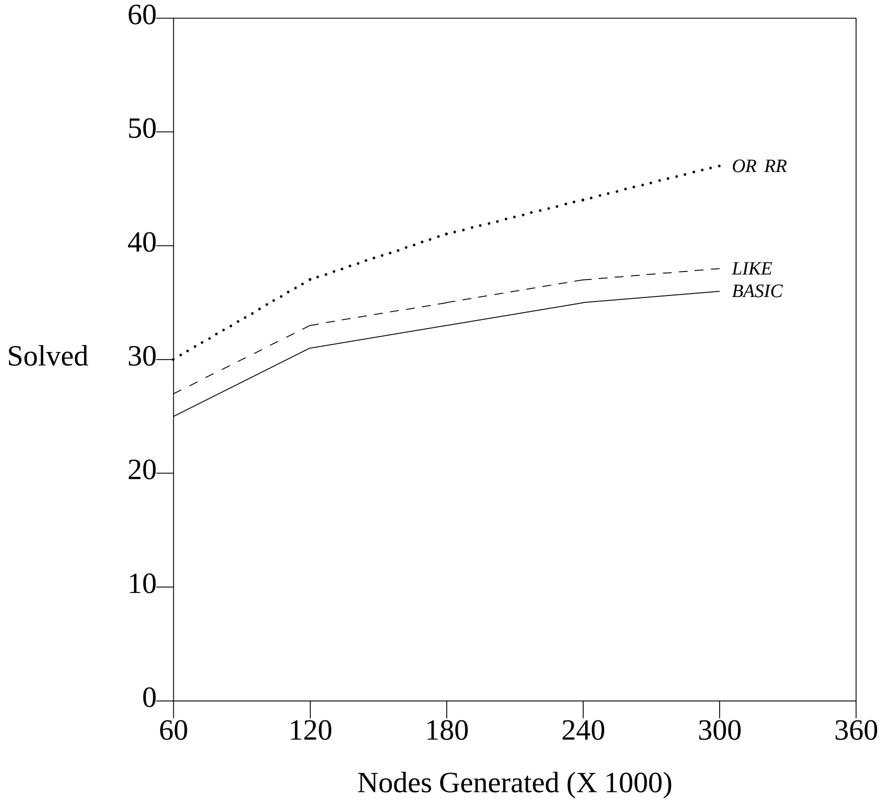

Fig. 10. Likelihood Version Results.

In Fig. 10, the results show that there is a small, but consistent improvement in the number of correctly solved problems over a program using the original version of the algorithm. The problem with the likelihood version appears to be that it has too much of a depth-first behavior: it always chooses to descend down to the left-most son that has the appropriate conspirators beneath it as well as the maximum likelihood for causing the desired result.

Two simple enhancements were made to this likelihood version adding more breadth. The first enhancement involved applying a round robin technique for choosing which branch to descend down at all nodes. We will call this new version simply *LIKE+RR*. The second enhancement involved altering the likelihood estimation function at leaf nodes so that more nodes have a better chance of being explored. The equation for $\uparrow LIKE(v)$ at leaf nodes will be:

$$\uparrow LIKE(v) = \begin{cases} 2 \text{ if } v = m \\ 1 \text{ if } (v - m) > 0 \text{ and } (v - m) \le 0.35 * Rsize \\ 0 \text{ if } otherwise \end{cases}$$

The equation for $\downarrow LIKE(v)$ is similar. We will call this new version *LIKE´+RR*.

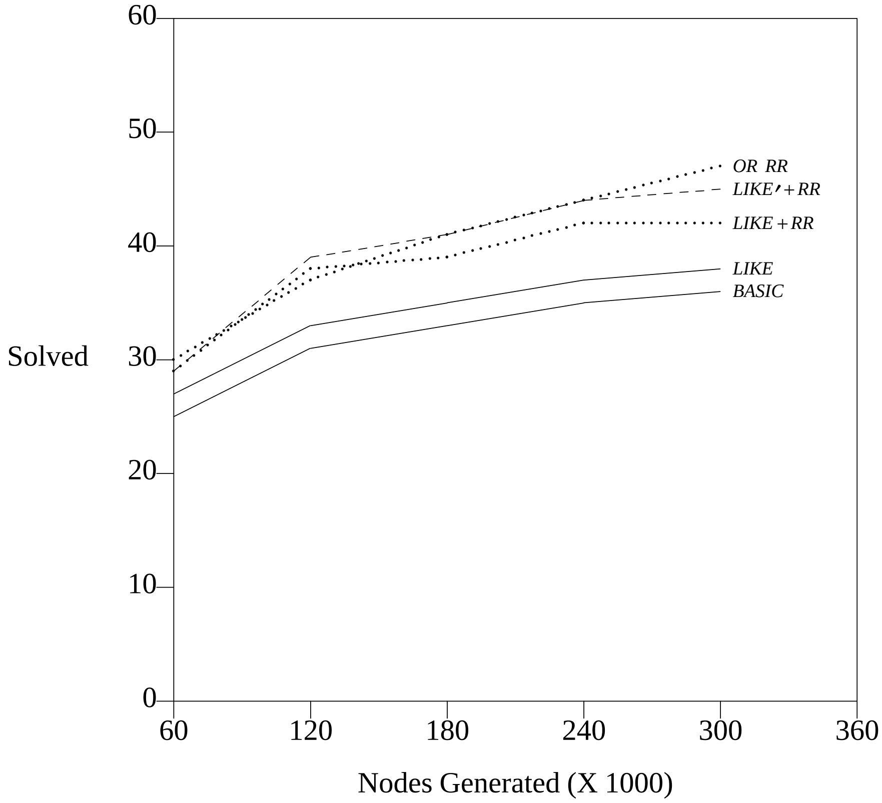

Fig. 11. Likelihood Variants Results.

Fig. 11 illustrates the performance of these likelihood variants. These results clearly show that adding breadth allows the algorithm to solve more problems. However, we were only able to raise the algorithm’s performance to that of the round robin versions. But at what cost do we achieve this? The additional overhead in using and computing likelihood values resulted in an increase in the average amount of time required to search such a tree by 36%. For comparison purposes, the round robin techniques increased the average time by less than 2%. Certainly the additional costs of using likelihood values is not worth the increased performance we were able to get through it.

The *best move cutoff* enhancement was added to a program that was already using the *OR RR* version of the Conspiracy Numbers algorithm. Fig. 12 compares the performance of the program before and after this enhancement was added. With the enhancement, the program was able to solve 4 new problems in addition to the 47 problems it could solve before. On the 47 original problems, the cutoff was applied an average of 3.7 times per problem causing an increase in the average attained conspiracy threshold from 9.1 to 9.6. On the 4 new problems, the cutoff was applied an average of 2.25 times, allowing the search to concentrate its effort in a portion of the tree which eventually paid off. Overall, this enhancement can be viewed as an important improvement to the algorithm.

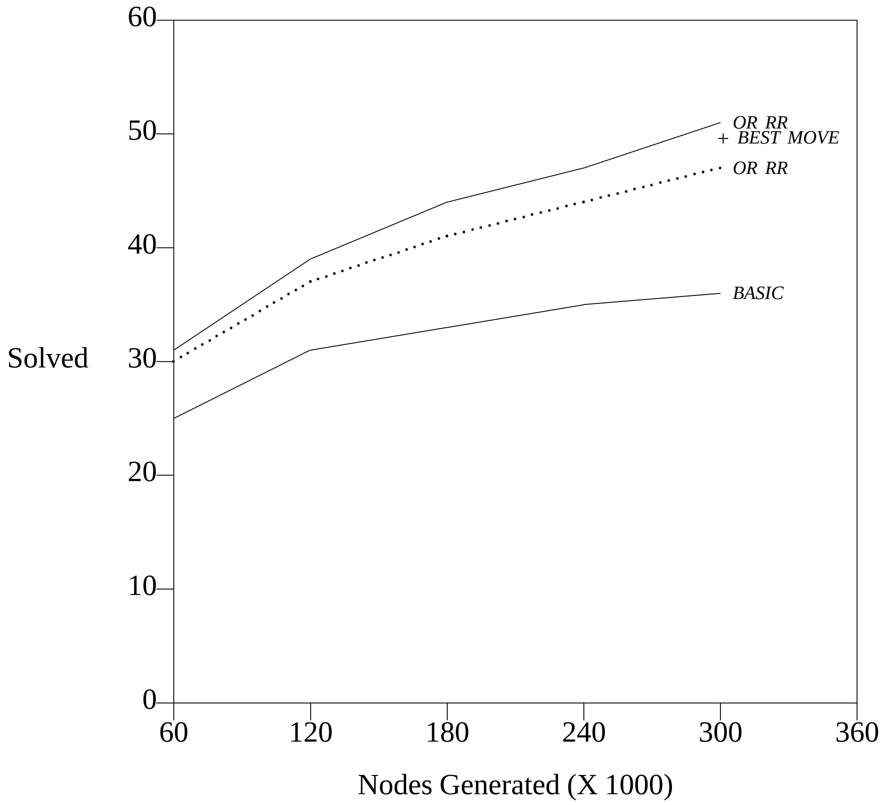

Fig. 12. Best Move Cutoff Results.

Alterations to two other sections of the algorithm were also attempted. One involved applying additional application-dependent knowledge in the sorting phase of the algorithm to improve the order in which newly generated nodes are placed in. In the other, the algorithm did not try to rule out *Vmin* or *Vmax* (the likely root values furthest from *Vroot*), but tried to change *Vroot* to either $Vroot + 1$ or $Vroot - 1$. We called this second alteration *Minimum Windows* because it is analogous to minimum window searches of minimax trees (Marsland, Reinefeld and Schaeffer 1987). Unfortunately, neither alteration improved the performance of the algorithm significantly (Klingbeil 1989).

Clearly breadth and best move cutoffs enhance McAllester's original algorithm. Although our experiments with likelihoods were not demonstrably superior, we believe this approach is important. With better application-dependent knowledge to reliably assign the probabilities, we believe program performance will improve. The resulting algorithm has similarities to B* (Berliner 1979; Palay 1982).

The performance of McAllester's original version has not fared well in comparisons with the $\alpha\beta$ algorithm. The enhancements to conspiracy numbers presented in this paper allow the algorithm to solve as many problems as $\alpha\beta$, given enough search time (Klingbeil 1989; Schaeffer 1990). Nevertheless, there is still considerable research work required on the algorithm before it can become a viable search method for real-time applications.

7. Conclusions

It is interesting to note that there is a strong relationship between conspiracy numbers and the new $\alpha\beta$ enhancements of singular extensions and null moves. Singular extensions (Anantharaman, Campbell and Hsu 1990) tries to identify *forced* moves in the tree, where a forced move is defined as one move being significantly better than all the alternatives. In conspiracy numbers, forced moves can easily be recognized by their requiring considerably less conspirators to achieve a value than the alternatives. Null moves (Beal 1989; Goetsch and Campbell 1988) try to identify moves with small chances of success and eliminate them from the search. Again, this information is available in conspiracy numbers; null move nodes have high conspiracy numbers. Although $\alpha\beta$, with its variety of search enhancements, is not the same as conspiracy numbers, it is obvious that there is a strong relationship between the two algorithms.

Perhaps the biggest contribution of conspiracy numbers is how it has influenced our reassessment of αβ. αβ has long been favoured because of the large portions of the search tree that it can eliminate from search. However, it is precisely at those nodes in the tree where cutoffs occur that conspiracy numbers may continue searching, not necessarily to show that there is something interesting to uncover, but rather to gather information that can be used to make better quality search decisions. It is the gathering and use of this likelihood information that forces us to re-examine αβ and the cutoffs which make it attractive. Whereas it is clear that both brute-force αβ and selective deepening conspiracy numbers have both pros and cons, the search is on for a new, possibly hybrid algorithm, that allows us to build mini-max trees to a variable depth in an application-independent manner.

There are many unanswered questions about conspiracy numbers. Little theoretical analysis of the algorithm has been done. It would be interesting to see if such work could predict the observed breadth/depth trade-offs. Lisa Lister's analysis of the minimal conspiracy numbers tree suggests some possible improvements to the algorithm (Lister 1990), however analysis of the algorithm's expected performance remains an open problem. Experimentally, conspiracy numbers has proven successful at tactical chess, but not positional chess. Positional chess has two characteristics that currently appear to hinder performance. First, the granularity of the results are much finer, necessitating a larger range of values. Second, node values are more unstable. In tactical chess, the evaluation of the son often yields the same as the parent. In positional chess, the son's score is usually close to, but different from the parent's. Consequently, expanding a node tends to change the minimax score, introducing some instability into the search. It remains an open problem how best to implement conspiracy numbers in this environment.

Acknowledgments

Robert Enns helped program our first implementation of conspiracy numbers. The financial support of the Natural Sciences and Engineering Research Council of Canada is greatly appreciated.

Althofer, I. 1988. Root Evaluation Errors: How They Arise and Propagate. Journal of the International Computer Chess Association, **11**, pp. 55-63.

Anantharaman, T., Campbell, M., and Hsu, F.H. 1990. Singular Extensions: Adding Selectivity to Brute-Force Searching. Artificial Intelligence, in press. *Also in* Journal of the International Computer Chess Association, **4**, 135-143, 1988.

Beal, D. 1989. Null Moves. *In* Advances in Computer Chess V. *Edited by* D. Beal. Elsevier Science Publishers, Amsterdam, pp. 65-79.

Berliner, H.J. 1979. The B* Tree Search Algorithm: A Best First Proof Procedure. Artificial Intelligence, **12**, pp. 23-40.

Chess Informator. 1985. Encyclopedia of Middlegame Combinations. Sahovski Informator, Beograd.

Goetsch, G., and Campbell, M. 1988. Experiments with the Null Move in Chess. AAAI Spring Symposium, pp. 14-18.

Klingbeil, N., and Schaeffer, J. 1988. Search Strategies for Conspiracy Numbers. Canadian Society for the Studies of Computational Intelligence Conference, pp. 133-139.

Klingbeil, N. 1989. Search Strategies for Conspiracy Numbers. M.Sc. thesis, Department of Computing Science, University of Alberta.

Knuth, D.E., and Moore, R.W. 1975 An Analysis of Alpha-Beta Pruning. Artificial Intelligence, **6**, pp. 293-326.

Korf, R. 1985. Depth-First Iterative-Deepening: An Optimal Admissible Tree Search. Artificial Intelligence, **25**, pp. 97-109.

Lister, L. 1990. An Analysis of the Conspiracy Numbers Algorithm. M.Sc. thesis, Department of Computing Science, University of Alberta.

Marsland, T.A., Reinefeld, A., and Schaeffer, J. 1987. Low Overhead Alternatives to SSS*. Artificial Intelligence, **31**, pp. 185-199.

McAllester, D.A. 1985. A New Procedure for Growing Mini-Max Trees. Technical Report, Artificial Intelligence Laboratory, Massachusetts Institute of Technology.

McAllester, D.A. 1988. Conspiracy Numbers for Min-Max Search. Artificial Intelligence, **35**, pp. 287-310.

Palay, A.J. 1982. The B* Tree Search Algorithm - New Results. Artificial Intelligence, **19**, pp. 145-163.

Reinfeld, F. 1945. Win At Chess. Dover Books.

Rivest, R.L. 1987. Game Tree Searching by Min/Max Approximation. Artificial Intelligence, **34**, pp. 77-96.

Schaeffer, J. 1986. Experiments in Search and Knowledge. Ph.D. thesis, Department of Computer Science, University of Alberta.

Schaeffer, J. 1990. Conspiracy Numbers. Artificial Intelligence, in press. *Also in* Advances in Computer Chess V. *Edited by* D. Beal. Elsevier Science Publishers, pp. 199-218, 1989.

Schruffer, G. 1988. Minimax-Suchen: Kosten, Qualitat und Algorithmen. Ph.D. thesis, Technical University Braunschweig.

Slate, D.J., and Atkin, L.R. 1977. Chess 4.5 - The Northwestern University Chess Program. *In* Chess Skill in Man and Machine. *Edited by* P.W. Frey. Springer-Verlag, New York, pp. 82-118.

Stockman, G.C. 1979. A Minimax Algorithm Better Than Alpha-Beta? Artificial Intelligence, **12**, pp. 179-196.

Figure Titles:

Fig. 1. A Minimax Search Tree.

Fig. 2. Conspirators.

Fig. 3. Conspiracy Numbers.

Fig. 4. Conspiracy Numbers Algorithm.

Fig. 5. Compromise vs Storage Intensive.

Fig. 6. Effect of Range Size on Tree Size (Ordering 100%).

Fig. 7. Effect of Ordering on Tree Size (Value Range = 33).

Fig. 8. Distribution of Nodes.

Fig. 9. Round Robin Version Results.

Fig. 10. Likelihood Version Results.

Fig. 11. Likelihood Variants Results.

Fig. 12. Best Move Cutoff Results.