CONTROL STRATEGIES FOR TWO PLAYER GAMES

Bruce Abramson

CUCS-234-86

Control Strategies for Two Player Games *

Bruce Abramson †

April 21, 1986

Abstract

Computer games have been around for almost as long as computers. Most of these games, however, have been designed in a rather ad hoc manner, because many of their basic components have never been adequately defined. This paper points out some deficiencies in the standard model of computer games, the minimax model, and outlines the issues that a general theory must address. Most of the discussion is done in the context of control strategies, or sets of criteria for move selection. A survey of control strategies brings together results from two fields: implementations of real games, and theoretical predictions derived on simplified game trees. The interplay between these results suggests a series of open problems that have arisen during the course of both analytic experimentation and practical experience as the basis for a formal theory.

1 Introduction: Computer Games, Why and How?

In 1950, when computer science was still in its infancy and the term "artificial intelligence" had yet to be coined, Claude Shannon published a paper

*This research was supported in part by the National Science Foundation under grant IST-85-15302.

†Department of Computer Science, Columbia University
and Computer Science Department, University of California at Los Angeles

called *Programming a Computer for Playing Chess* [Sha50]. He justified the study of chess programming by claiming that, aside from being an interesting problem in its own right, chess bears a close resemblance to a wide variety of more significant problems, including translation, logical deduction, symbolic computation, military decision making, and musical composition. Skillful performance in any of these fields is generally considered to require thought [1], and satisfactory solutions, although usually attainable, are rarely trivial. Chess has certain attractive features that these more complex tasks do not: the available options (moves) and goal (checkmate) are sharply defined, and the discrete model of chess fits well into a modern digital computer. He then went on to outline the basics of this model and describe a method by which chess could be implemented on a computer.

The discrete model to which Shannon referred is called a game tree, and it is the general mathematical model on which the theory of two-player zero-sum games of perfect information is based [NM44]. Chess belongs to this class of games; it is perfect information because all legal moves are known to both players at all times, and it is zero-sum because one player's loss equals the other's gain. At the top of the chess tree is a single node which represents the initial setup of the board. For each legal opening move, there is an arc leading to another node, corresponding to the board after that move has been made. There is one arc leaving the root for each legal opening, and the nodes that they lead to define the setups possible after one move has been made. More generally, a game tree is a recursively defined structure which consists of a root node representing the current state and a finite set of arcs representing legal moves. The arcs point to the potential next states, each of which, in turn, is a smaller game tree. The number of arcs leaving a node is referred to as its *branching factor*, and the distance of a node from the root is its *depth*. If b and d are the average branching factor and depth of a tree, respectively, the tree contains

[1] Exactly which tasks require thought or intelligence is among the most widely debated issues of our time, and is clearly beyond the scope of this paper. Because this is primarily a survey, I have chosen to accept any author's claim that his system is intelligent, and take no responsibility for perceived inconsitencies in the use of the term throughout the paper.

approximately b^d nodes. A node with no outgoing arcs is a *leaf*, or terminal node, and represents a position from which no legal moves can be made. When the current state of the game is a leaf, the game terminates. Each leaf has a value associated with it, corresponding to the payoff of that particular outcome. Technically, a game can have any payoff, (say a dollar value associated with each outcome), but for most standard parlor games, the values are restricted to Win and Loss (and sometimes Draw).

In two-player games, the players take turns moving, or alternate choosing next moves from among the children of the current state. In addition, if the game is zero-sum, one player attempts to choose the move of maximum value, and the other that of minimum value. A procedure which tells a player which move to choose is a strategy for controlling the flow of the game, or a *control strategy*. In principle, the decision of which choice to make is a simple one. Any state one move away from the leaves can be assigned the value of its best child, where best is either the maximum or minimum, depending on whose turn it is. States two moves away from the leaves then take on the value of their best children, and so on, until each child of the current state is assigned a value. The best move is then chosen. This method of assigning values and choosing moves is called the *minimax algorithm*, and defines the optimal move to be made from each state in the game. An example of the minimax algorithm is shown in figure 1 on the tree of Nim, a simple game which plays an important role in the mathematical theory of games [BCG82]. Most interesting games, however, generate trees which are too large to be searched in their entirety, and thus an alternative control strategy must be adopted. The checkers tree, for example, contains roughly 10^{40} moves, and the chess tree in the neighborhood of 10^{120}. In these games, the tree is searched to some limit (generally set by the computational power available), and domain specific heuristic information is applied to tip nodes at the search frontier to return an estimated *static evaluation function*. The control strategy must rely on these estimates.

When the tip values are exact, the tree is *complete*. Otherwise, the tips are internal nodes, and the tree is *partial* [2]. Complete trees are well

[2]Technically, it is possible to have internal nodes with exact values, as well. However, once the outcome of a game is known, the game is effectively over. Thus, any node with

understood but rarely applicable. Applications must rely on partial trees, which are invariably implemented in an ad hoc manner. Partial game trees have appeared primarily in two subfields of artificial intelligence, game programming and the analysis of heuristic search methods. The emphases of these fields are quite different. Game programming is concerned with the development of computer programs that play specific games well, hopefully at or beyond the level of a human expert. Thus, the models that have been studied are "real" games, (mostly chess), and the metric for success is performance vs. machine or human opponents. Heuristic analysis, on the other hand, is concerned with investigating the accuracy of heuristic techniques, when compared to some ideal. Since the trees generated by interesting games tend to be both too large for the ideal to be calculated, and too complex to be dealt with analytically, simplified models and "artificial" games have been defined. Both fields have contributed interesting results about control strategies. Nevertheless, the interplay between them has been minimal; games that have actually been implemented have not been analyzed, and theoretical predictions have not been considered when the implementations were designed. Because of this, very little about the general theory of partial game trees is known.

This paper presents a survey of the various control strategies that have been used. Some basic game tree search procedures are described in section 2. Section 3 discusses strategies that have been implemented or proposed in the context of real games, while section 4 outlines some theoretical results derived on simple models and strategies motivated by these results. Section 5 discusses areas in which interaction between the applied and theoretical aspects of the field could be beneficial to both, and section 6 suggests some directions for future research.

---

a known exact value can be treated like a leaf, and the distinction between complete and partial trees remains valid.

2 Background: Basic Game Tree Searching Procedures

Shannon's analysis included the description of two families of control strategies for the chess tree, type A and type B. A type A strategy behaves as if the tip nodes are leaves, and applies minimax to the estimates calculated by the static evaluator. This involves a full-width fixed-depth search (consideration of all possibilities up to a set distance away from the root), and uses heuristics only in assigning the values to the tips. Because the values it minimaxes are estimates, this technique does not always make the optimal move, and thus should be distinguished from minimax on complete trees, which does. For the sake of clarity, throughout the rest of this paper, minimax on partial trees will be referred to as the *minimax procedure*; the term algorithm will be reserved for complete trees. When only the term minimax is used, it applies equally to both. The underlying assumption behind the use of this procedure is that the estimates are reasonably accurate; the succcess of the strategy depends on the validity of this assumption. Type B strategies, on the other hand, only consider reasonable moves. Heuristics are used not only to calculate tip values, but also to decide which moves are worth considering. Throughout most of the history of chess programming, the general feeling was that whereas type A is easier to implement, type B shows greater promise for expert performance.

In the years since 1950, two important observations have led to innovative techniques which are now standard: there is an easily recognizable class of obviously incorrect moves, and a preset search depth may not fully exploit the computational resources available. These observations led, respectively, to the development of $\alpha - \beta$ *pruning* and *iterative deepening search*. The exact origins of $\alpha - \beta$ are disputed, but the first paper to discuss it in detail was probably [EH63]. $\alpha - \beta$ prunes by recording boundaries within which the minimax value of a node may fall. α is a lower bound on the value that will be assigned to a maximizing node, and β an upper bound on the value of a minimizing node. Descendants whose minimax values fall outside the range are pruned, and their subtrees can be ignored. To insure that the correct (minimax) choice is not missed, α and β start at minus and plus infinity, respectively, and are updated as

the tree is traversed. Figure 2 shows an example of $\alpha - \beta$ pruning. The sensitivity of $\alpha - \beta$ to the order in which nodes are examined was first pointed out in [SD69]. The procedure's behavior under several different orders was analyzed in [FGG73] [KM75], where it was shown that in the average case, $\alpha - \beta$ cuts the effective branching factor from b to $\sqrt{b}$, and allows the search depth to be doubled. The asymptotic optimality of $\alpha - \beta$ over the class of all game searching algorithms, in terms of the number of nodes searched, was proved in [Pea82]. Parallel implementations for even further speedup were surveyed in [MC82]. Two more recent pruning techniques, SSS* [Sto79] and SCOUT [Pea80] have been shown to occasionally prune more branches than $\alpha - \beta$ [RP83] [CM83]. These instances, however, are few and far between, and the gain is generally inconsequential. Furthermore, SSS* requires much more space and is harder to implement. For these reasons, neither SSS* nor SCOUT has ever been used in a successful performance oriented game program.

Unlike the pruning algorithms, which are unique to two-player games, iterative deepening is a completely general search paradigm. Iterative deepening allows the search to proceed until a preset time, rather than a preset depth, is reached. This is accomplished by first performing a full-width (or $\alpha - \beta$) search to depth 1, then to depth 2, then to depth 3, etc. When the procedure times out, it makes the move that was singled out as best by the deepest search completed. The advantage of using iterative deepening in games was first demonstrated by Chess 4.5 [SA77], one of the most powerful chess programs of the 1970's. Although the technique was shown to be a time (number of nodes expanded) and space (amount of bookkeeping required) optimal tree search in the context of one-player games [Kor85], the analysis is equally applicable to two-player games.

The importance of these algorithms is twofold. In their pure form, they have become part of the standard implementation of type A strategies, crucial to the development of some very powerful programs to be discussed in section 3.1. In this context, they are merely enhancements to the minimax procedure; they make the same choice faster. Thus, their contribution to the study of decision quality of control strategies is negligible. Various modifications to $\alpha - \beta$, however, have formed the basis of many of the type B strategies discussed in section 3.2. These modifications generally are not

guaranteed to return the minimax value, and raise some interesting issues related to decision quality.

3 Game Programming

Even partial game trees have their limits — programs that play backgammon [Ber80], Go, Scrabble, and poker, [Bra83] have used significantly different models. Their practical applicability seems to be restricted to perfect information games with "manageable" branching factors, such as chess, checkers, kalah, and Othello. Although some interesting machine learning experiments have been run using checkers as the example domain [Sam63] [Sam67] [Gri74], the game that has generated the most interest, from Shannon's article on, is chess. In 1975, Newborn wrote that "all the chess programs that have ever been written and that are of any significance today are based on Shannon's ideas," and "improvements in programs are due primarily to advances in computer hardware, software, and programming efforts, rather than fundamental breakthroughs in how to program computers to play better" [New75]. To a great extent, this is still true in 1986. Nearly all control strategies contain an element of the minimax procedure; the major distinction between them is whether all paths are searched to the same depth (type A) or not (type B), and if not, what the criterion for expanding nodes is.

At the heart of every type A strategy lies a fixed-depth, full-width minimax search. In addition to α–β and iterative deepening, a *secondary search* is frequently used to augment the basic strategy. A secondary search is exactly what it sounds like: a second search started at a point other than the root. The reasons for using this technique will be discussed in section 3.1. Although secondary searches technically violate the fixed-depth characterization, strategies which use them can still be classified as type A. Fixed-depth, full-width minimax is comparatively easy to implement and conceptually simple to justify — as the estimates approach exact, the procedure approaches optimal performance. Errors occur only in static evaluation. The major drawback to type A strategies is the amount of computation required for a full width search; even with the help of a pruning algorithm, the large branching factor in most games limits the search

to a relatively small portion of the tree. Type B strategies, on the other hand, constitute a rather large and diverse family. Their common feature is that they generate all possible next moves, but expand only the most promising ones. The distinction between generation and expansion may be a bit subtle. A node is *expanded* when its children are *generated*. In a full-width search, first the root is expanded by generating all of its children. Then all nodes at depth 1 are expanded, generating all depth 2 nodes, etc. In a selective search, all chidren of the root are generated, but only some are expanded. A wide variety of heuristics have been suggested as means for selecting only the most promising nodes for expansion. The major difficulty encountered is in devising criteria which are good enough to result accurate play.

One of the first computer chess matches, designed in part to compare the performances of the two strategy types, pitted the Kotok-McCarthy type B program against the ITEP type A program in four games. When ITEP searched 3 ply, both games were draws. When it searched 5 ply, it won both [New75]. The problem with the Kotok-McCarthy program was that it was not sufficiently selective in its choice of nodes to forward prune. In the words of former world chess champion Mikhail Botvinnink, "The rule for rejecting moves was so constituted that the machine threw the baby out with the bath water" [Bot70]. Thus, the domination of a type A program over a type B program does not necessarily indicate that selective strategies are inferior to those relying on brute force, it simply highlights the relative difficulty in implementing them.

3.1 Type A Strategies: Full-Width Minimax Searches

Many of the earliest chess programs used type A strategies and were able to achieve modest performance [New75] [Ber78]. The major difficulty these programs faced was that practical computational limits were reached while search was still shallow. The standard measure of search depth is *ply*, or the number of consecutive moves considered. In an average chess game, each player has between forty and fifty moves. A complete search, then, would have to exceed 80 to 100 ply. In most of these early programs, only three or four ply were searched. In addition, artificial termination of search at a

uniform depth implies that anything not detectable at the search frontier is effectively nonexistent. Thus, these programs generally failed to realize when they were in the midst of a complex tactical maneuver, such as a material trade in chess. This problem is known as the horizon effect, and is a necessary consequence of the decision to terminate search uniformly [Ber73]. The first method proposed to alleviate the effect was a secondary search, with the selected tip as root [Gre67]. This approach, however, does not remove the horizon, it merely extends it.

Positions which are not affected by the horizon are called *quiescent*, or quiet, because there is no imminent threat that will radically shift the game from what was anticipated at the horizon. The importance of applying the static evaluator only to quiescent positions was pointed out by Shannon; the issue of how to determine which positions are quiescent, however, is still largely open. Some attempts to resolve the problem are discussed in [Kai83]. Beal proposed consistency as a means for detecting quiescence. A node is consistent if its static value is the same as its backed up value from a one ply search [Bea80]. He later modified this *consistency search* to *locked-value search* [Bea82]. A value is locked if it has two children with the same best value. If this value is not correct, both of the children must have been evaluated incorrectly. Although there is no guarantee that this approach helps detect quiescence, it is generally safe to assume that single errors are more likely to occur than double errors. Thus, a locked value is less likely to be an anomaly brought about by the horizon effect than a non-locked value. A more common approach to quiescence detection and correction is to perform some sort of secondary search beyond the frontier for positions which include captures, checks, or move promotions, and to consider all other positions quiescent. Many programs, including Chess 4.5, used this method.

Despite these difficulties, many successful programs have used type A strategies. One of the general assumptions underlying them all is that the deeper the search, the better the performance. Although it was believed at one point that the limits of brute force search would be reached long before a computer could play master level chess [Ber73] [Bot84], state of the art technology has resulted in special purpose architectures that have done just that. The first such machine, Belle, relied almost totally on

speed to become the first computer to achieve master rating [CT77] [CT82]. Speed allowed Belle to search to a previously unacheivable eight ply, make more accurate decisions, and apparently avoid the horizon effect [Ber81]. IAGO, an Othello program which plays at about world championship level, also used a standard full-width, $\alpha - \beta$ search with iterative deepening. The development of IAGO stressed analysis of positions, and resulted in a very strong static evaluation function. This function, combined with the relatively small tree of Othello (relative to chess, that is) accounts for the program's success [Ros82]. Chess 4.5, which introduced iterative deepening, also included a hash table to avoid redundant checks. When a node is encountered, its value is entered in the table. If it is reached a second time, the subtree beneath it need not be searched. The avoidance of redundancy allows deeper searches to be performed without requiring additional time.

The *M & N procedure* attempts to improve performance by making better use of the information gathered at the tips, rather than by speedup. This is done by finding the minimax value, and adding a bonus function to it. This bonus is an experimentally derived function of the M maximum or N minimum values. Thus, the backed up value contains information about the best several choices, not only the single best. Using the game of kalah as their example domain, Slagle and Dixon showed that this procedure improves play to about the level that would be achieved by extending minimax an additional ply. Its major drawback is that pruning techniques become more complicated and less helpful [SD70]. The notion of saving multiple nodes was also used by Harris to devise *bandwidth search* [Har74]. The idea underlying this search is that making the optimal choice is not always necessary, as long as one which is not too far from optimal is guaranteed. If an evaluation function with constant bounded error can be found, any node whose value is within those bounds of the currently most promising one may, in fact, be best. The original domain of bandwidth search was one player games, where the idea of a constant bounded error was considered a weakening of the admissibility requirement. Whereas an admissible function never overestimates the true value, a function with constant bounded error never overestimates by more than e or underestimates by more than d. This scheme has the advantage of not discarding all moves

right away, thereby allowing for occasional error recovery. Its disadvantage is that it does not search for the minimax value, but chooses the first node found within $(e + d)$ of it. The algorithm fared well in Four Score (a 3-dimensional, 4-by-4 tic-tac-toe game) competition, but holds little promise for more complex games because of the difficulty of finding heuristics that satisfy the bandwidth conditions.

The first intelligent type A strategy is part of a new chess architecture, *SUPREM* (Search Using Pattern Recognition as the Evaluation Mechanism). SUPREM links two machines together, one smart but slow (the oracle), and one very fast (the searcher). The oracle is the source of game specific analyses, and is responsible for downloading preprocessed pattern recognition data to the searcher. This includes information about what representations must be maintained, how to update them incrementally, and the value of any instance of any representation. The searcher must then search quickly, update many pieces of information simultaneously, and combine information rapidly [BE86]. In SUPREM, the searcher actually performs a parallel search of *all* legal moves, indicating that the basic control strategy is type A. Unlike other type A programs, SUPREM is considered intelligent because it recognizes patterns. Unlike other intelligent programs, however, the information is used for evaluation, not strategy. This combination of guided knowledge and rapid search is powerful enough to form the basis of Hitech, the current North American Computer Champion, and highest rated chess computer to date (2250 as of early 1986). According to Berliner, who designed the system, Hitech's rating is rising rapidly, and may eventually reach the grandmaster level.

3.2 Type B Strategies: Selective Searches

One of the features common to all type B strategies is the use of domain specific information to select promising nodes and lines of play. This approach is believed to be the method used by human experts [Ber73] [Ber77b] [Bot70] [Bot84], and is thus interesting for two reasons. One, it is a more accurate model of cognitive processes, and may aid in the understanding of human decision making [NSS63]. Two, it muffles the combinatorial explosion by drastically reducing the effective branching factor, hopefully leading

to improved performance. The domain specific knowledge required to make type B strategies work can be infused in various forms. Michie identified three types of knowledge that are useful in game programs: rote memory (dictionary entries of board positions), theorems [3] Hash tables of the type used by Chess 4.5, the inclusion of standard book openings in Belle and other programs [CT77] [CT82] [Tho82], and the endgame library of PIONEER [Bot84], are all examples of successful uses of rote memory. In addition, large tables have proven very useful in machine learning experiments that develop better static evaluation functions for checkers [Sam67] [Gri74]. In terms of the design of intelligent search strategies, however, rote memory is not particularly helpful. Heuristics and pattern recognition, on the other hand, have each led to the development of some interesting strategies.

3.2.1 Forward Pruning Based Strategies

Heuristics are general guidelines built into the program. Bratko and Michie [BM80] wrote a program to play the chess endgame of KRKN (King and Rook vs. King and Knight) which included an advice table, a list of general heuristics like "avoid mate". This is a rather nonstandard use of heuristics, however. Typically, they are included in the evaluation function and the forward pruning criteria [Ber77a] [Ber77b], not in separate data structures. Forward pruning is a technique used by many type B strategies. Unlike $\alpha - \beta$, which only prunes nodes that will not be chosen, forward pruning techniques ignore all nodes which don't look very promising, thereby running the risk of missing the correct choice.

One heuristic that has been used to define a type B strategy is "only expand nodes which look at least as good as the current best." This heuristic defines a technique called *razoring* [BK77]. At first glance, this procedure sounds strikingly similar to $\alpha-\beta$. In fact, the only difference between them lies in the criteria used for determining the promise of a node. $\alpha-\beta$ relies on backed up minimax values, razoring on the static evaluation. Thus, while razoring prunes nodes that don't look good, $\alpha - \beta$ only prunes nodes that are not good. Razoring should be used in addition to $\alpha - \beta$, not instead of

[3] The term "theorems" is confusing, because the knowledge used is generally not a theorem in the mathematical sense. In my opinion, "heuristics" or "guidelines" is more accurate.

it. In the worst case, then, razoring will prune the same nodes as $\alpha - \beta$, with only the added cost of some additional evaluations. In the average case, however, razoring will prune nodes earlier than $\alpha - \beta$, narrow the branching factor more rapidly, and deepen search, all in exchange for occasionally missing the best choice. Preliminary experiments showed that in the exchange, razoring gained, on the average, an order of magnitude over $\alpha - \beta$ in a 4-ply tree, in terms of the number of nodes expanded. Razoring is illustrated in figure 3.

Another heuristic which has been considered is "start the search with an idea about the true value of the root." This heuristic leads to a procedure called *aspiration search*. Its development was motivated by the observation that $\alpha - \beta$ works best if the node that will eventually be returned by minimax is among the first nodes examined SlagDix69. The reason for this is rather straightforward: if the best alternative is considered first, the α and β values are quickly set to define a narrow range around the minimax value of the root, thereby resulting in a great deal of pruning. The predetermined upper and lower bounds, then, can serve the roles of α and β. If the procedure used to determine these bounds is fairly accurate, the search tree can be narrowed quickly. However, because the bounds don't start at plus and minus infinity, it is possible that they are wrong. Once again, the guarantee of returning the minimax value is lost. The use of this heuristic as a means of pruning absurd moves was discussed in [AAD75]. Figure 4 shows how aspiration search can be used to augment $\alpha - \beta$.

The B* algorithm [Ber79] uses a simple heuristic of a very different nature, "terminate the search when an intelligent move can be made." This algorithm was motivated by the desire to avoid the horizon effect by defining natural criteria for terminating search. The search proceeds in a best-first manner, and attempts to prove that one of the potential next moves is, in fact, the best. By concentrating only on the part of the tree that appears to be most promising, B* (and best-first searches in general), avoids wasting time searching the rest of the tree. Berliner's adaptation of best-first searches to game trees included the first modification to Shannon's original model. Instead of associating a single value with each node, B* uses two evaluation functions, one to determine an optimistic value, or upper

bound, and one for a pessimistic value, or lower bound. The search is conducted with two proof procedures, PROVEBEST, which attempts to raise the lower bound of the most promising node above the upper bounds of its siblings, and DISPROVEREST, which tries to lower the upper bounds of the siblings beneath its lower bound. The search terminates when the most promising choice has been proven best. Figure 5 illustrates the use of these procedures. Although B* sounds particularly appealing from both the speedup and cognitive modeling viewpoints, it does have its dark side. Like all best-first searches, a good deal of storage space is needed to keep track of the promise of each node on the generated-not-expanded (*open*) list, so that the focus of the search can shift as necessary. More significantly, though, is that B* is not guaranteed to terminate before time runs out. If this occurs, it loses the edge of making intelligent decisions, and has to choose whatever looks best at the time. The success of B* lies, to a great extent, in its ability to correctly select the most promising node and most efficient proof procedure. Several variations which focused on proof procedure selection have been studied [Ber79] [Pal82], and a scheme which selected them probabilistically was shown to be somewhat stronger than one which made deterministic choices.

The advantage of ranges, of course, is that they contain more information than point probabilities. Palay extended this reasoning one step further, and devised the idea of passing entire probability distributions. A distribution contains complete information about the likely location of a node's value, and thus retains considerably more information than just a range. He combined this idea with the control aspects of the B* algorithm to yield a powerful best-first search strategy, PSVB* [Pal85], and showed that by using distributions, an increase in efficiency of 91% over the use of ranges is possible.

An interesting idea that has recently been suggested as a selection criterion is "attempt to stabilize the value of the root." This heuristic was used to develop a procedure called *conspiracy search* [McA85]. The value of a node is stable if deeper searches are unlikely to have any major effect on it. In a conspiracy search, the root's stability is measured in terms of conspiracy numbers, the number of leaves whose values must change to affect its (the root's) value. If the number of conspirators required to change the

root value is below a certain threshold, the value is assumed to be accurate. At any given point during the search, the possible values of the root are restricted to the interval $[V_{min}, V_{max}]$, where V_{min} and V_{max} are the values of its minimum and maximum accessible descendants at the search frontier, respectively. To update the range, either prove that the minimizing player can avoid V_{max}, or that the maximizing player can avoid V_{min}. The decision of which to prove at each point can be made with the help of the conspiracy numbers. Unlike B*, there is no need to change the evaluation function (or to use multiple functions) to derive the interval — a single function will suffice. An example of conspiracy search is given in figure 6. Alone among the procedures discussed in this section, conspiracy search has been neither analyzed nor implemented. This is not due to any fundamental flaws in the procedure, but rather because it was only recently proposed. However, the heuristic on which it is based sounds reasonable, and hopefully forthcoming studies will determine its value as a control strategy.

3.2.2 Plan Based Strategies

Although pattern knowledge has not been used as extensively as heuristic knowledge in the design of type B strategies, several interesting systems have used patterns to plan in game domains. For example, Bramer described an optimal program for the chess endgame KPK (king and pawn vs. king) which used pattern knowledge [Bra80]. The PARADISE chess program (PAttern Recognition Applied to DIrecting SEarch), [Wil77] [Wil80] [Wil82], relies almost completely on pattern recognition to direct search. Like the B* algorithm, PARADISE expresses node values as ranges, attempts to prove that one move is the best, and terminates search based on knowledge, not parameters (such as depth or time). Unlike B*, however, PARADISE uses a large collection of plans, or sequences of moves to be made from various positions, to avoid the errors caused by the horizon effect. To compensate for the expense of maintaining the knowledge base, a small tree is searched. This is possible because the use of plans drastically reduces the branching factor. Information is communicated from one part of the tree to another using a "hodge-podge" collection of cutoffs that control the search, and indicate when searches along abandoned lines should be

resumed. Because of the rather ad hoc nature of these criteria, PARADISE does not make a major contribution to the theory of control strategies; its major contributions are to the fields of planning and pattern recognition.

One major issue raised by PARADISE, though, is how to generate plans to store in a data base. PARADISE's plans generally lead to a goal other than winning the game. Pitrat devised a general scheme for generating plans, and showed a few examples of its application to a simplified chess domain [Pit77] [Pit80]. His program is given a description of the initial state, and told to find a combination of moves that will lead to a specific goal. In order to succeed, the program must be given a less ambitious goal than "win the game." If the goal cannot be met as given, the program fails. Thus, it seems that the successful implementation of pattern recognition knowledge is directly related to the definition of inexact, or approximate tasks, which hopefully correspond to the ultimate goal of winning. Evidently, guiding plans towards a most promising node is considerably harder than guiding searches towards one. Whereas many search programs have performed reasonably well without defining specific goals, plans need them to succeed.

Botvinnik has identified the development of inexact goals to guide inexact search as one of the most important problems in the design of intelligent systems [Bot70] [Bot84]. This points out a fundamental flaw in the original definition of partial game trees: there is no clear understanding of what a static evaluator is attempting to estimate. Ostensibly, it should approximate the actual value of the node, (the value that would be returned by the minimax algorithm), although when a multi-valued function is used to estimate a binary valued one, it is unclear precisely what the estimate corresponds to. In the domain of chess, Botvinnik identified material advantage as the inexact goal. This problem was discussed in the context of the *chess master's method*, an ambitious control strategy that made heavy use of both heuristic and pattern knowledge [Bot82]. The pattern knowledge component appears in the form of a game specific action tree, which augments and directs the search tree. As the search tree grows, the action tree records purposeful moves, or goals that can be attained without exhausting resources. Heuristic knowledge is available through three general limitation principles: (i) if improvement is possible, it is contained in the

tree, (ii) new possibilities should be considered only if they promise an improvement, and (iii) only goals which don't exhaust the time limits should be considered. Botvinnik's scheme incorporates many of the ideas found in other control strategies. The limitation principles indicate a best-first approach to node expansion, and the action tree is similar to a dynamically constructed knowledge base of plans. These ideas were implemented in PIONEER, a system which was originally designed to play chess, but was most successful in the realm of planning maintenance repair schedules for power stations in the Soviet Union [Bot84].

The communication between the (game specific) action tree and (general) search tree parallels the relationship between the oracle and searcher in Hitech [BE86]. The difference between them lies in the role of knowledge. PIONEER uses it for strategic purposes, Hitech for evaluation. In either case, the lesson is the same: a functioning intelligent system needs both a general methodology and domain specific knowledge.

4 The Analysis of Heuristic Search

Throughout most of the 1950's, 60's, and 70's, all of the analytic work done on game trees dealt with determining the efficiency of node ordering and $\alpha - \beta$. During this period, a near-universal assumption was that the minimax procedure, although fallible, was the strongest conceivable control strategy, and its performance would improve directly with the length of lookahead and accuracy of the static evaluator. Alternative, or modified strategies, of the type discussed in section 3.2 were motivated by the desire to either model cognitive activities [NSS63] or improve performance faster than search depth could be extended [Ber73]. The perception of the minimax procedure has undergone a radical change in the 1980's. The advent of specialized chess architectures allowed lookahead to be lengthened, the effect of the horizon to be largely overcome, and computers to play master level chess. By 1985, there were even commercially available chess machines playing at or near the master level [Kop85]. At about the same time that these machines were convincing game programmers of the inherent power of full-width $\alpha - \beta$ minimax search, analyses of the procedure began questioning its theoretical accuracy.

One of the reasons that these analyses were so long in the coming is the complexity of the models. Games like chess were chosen as abstractions of the real world that were simple enough to allow computer simulation and experimentation. They were nowhere near simple enough to allow mathematical analysis. Thus, a new model had to be defined which was an abstraction of the game tree. The most frequently studied model to date is the (d,b,F)-tree [Pea80] [Pea84]. A (d,b,F)-tree has a uniform leaf depth d, a uniform branching factor b, and leaf values assigned by a set of identically distributed random variables drawn from a common distribution function, F. With the definition of the model and a family of "artificial" games which embody the simplifications, studies of (and challenges to) the accuracy of minimax became possible. In the 80's, then, the primary motivation behind defining alternative strategies to minimax has become the discovery of a procedure which is correct in some theoretical sense. This section is divided into two parts. The first outlines the results which caused minimax to fall into disfavor with theoreticians, and the second describes some control strategies for partial game trees which avoid these theoretical difficulties.

4.1 The Benefits of Lookahead

The aspect of the minimax procedure that has come under fire in the 80's is lookahead. Lookahead is generally assumed to be helpful because it results in the proper move being chosen more often. Over the years, an impressive body of empirical evidence has been amassed to support the validity of this claim, all in the form of successful programs which rely on it. Nevertheless, recent analyses have uncovered some surprising results.

The potential futility of looking ahead was addressed by Pearl, who derived the *minimax convergence theorem* [Pea80] [Pea84]. This theorem states that in a deep enough (d,b,F)-tree, the root value is essentially predetermined; the value is a function of b, and the variance a function of d. Specifically, as $d \rightarrow \infty$, if F is continuous, the minimax value of the root converges to the $1-\xi_b$-*quantile* of F, where ξ_b is the solution of $x^b+x-1=0$. If, on the other hand, F is discrete with values $v_1 < v_2 < \cdots < v_M$, and $1-\xi_b \neq F(v_i)$ for all i, the root's value converges to the smallest i sat-

isfying $F(v_{i-1}) < 1 - \xi_b < F(v_i)$. In a binary ($b = 2$) tree, for example, $1 - \xi_b = 1 - (\sqrt{5} - 1)/2 \approx .382$. Thus, if F is continuously distributed between 0 and 1, the root converges to .382. If F is restricted to the integers between 0 and 100, then $F(38) < .382 < F(39)$, and the root converges to 39. Perhaps the most interesting case occurs when F is a binary function, in which each leaf is a win ($v_W = 1$) with probability P, and a loss ($v_L = 0$) with probability $1 - P$. The convergence theorem for discrete distributions indicates that if $P > \xi_b$, the root will converge to 1, and if $P < \xi_b$ it will converge to 0. The only condition under which a fair game (one in which either player may win) is possible, then, is when the root fails to converge, or $P = \xi_b$. Nau pointed out that the minimax convergence theorem does not account for a widely observed phenomenon known as biasing (this can be observed in real games, not only (d,b,F)-trees) — the tendency for the player searching to perceive himself as winning if the search depth is odd, and losing if it is even [Nau82b]. He showed that this is due to errors in the static evaluator, and derived the *last player theorem*, which states that the value returned by the minimax procedure on a (d,b,F)-tree approaches ξ_b if one player moved at the bottom level of the search, but $1 - \xi_b$ if the last move belonged to the other. This theorem makes an important statement about the way lookahead values should be interpreted: values returned from alternating depths form two distinct sequences, and must be considered separately.

Minimax convergence indicates that there are instances in which lookahead is not helpful. The theorem can be viewed as an outcome of the weak law of large numbers: as the number of events in the sample space increases, the deviation of the observed outcome from the expected outcome decreases. In the case of (d,b,F)-trees, an event is the assignment of a leaf value, the observed outcome is the minimax value, and the expected outcome is $1 - \xi_b$. As $d \rightarrow \infty$ the number of leaves grows exponentially, indicating that the observed minimax value will always converge to the predicted value. Since the degree to which a node may deviate from $1 - \xi_b$ depends only on its height, (distance from the leaves), lookahead is more or less worthless high in the tree — all choices are roughly equivalent because the children of the root all converged to the same value. When that occurs, random play is just as effective as lookahead. On the other hand, when

play approaches the end of the game, there are too few leaves for the weak law of large numbers to apply, and the values may vary greatly from $1 - \xi_b$. At this stage of the game, lookahead may help determine these values more accurately.

An even stronger statement about lookahead is made by another recent discovery, *minimax pathology*, a phenomenon whereby the decisions made by minimax may become less reliable as lookahead length increases. Nau, (and independently, Beal [Bea80]), performed an error analysis on (d,b,F)-trees in which F was a uniform random distribution of binary values, and discovered that for an infinite class of game trees, as search depth increases, so does the probability that an incorrect move will be made [Nau83a].

Pathology was first demonstrated on a family of board splitting games developed by Pearl as simple games with all the properties of (d,b,F)-trees [Pea84]. In board splitting, a square b^d-by-b^d board is covered with 1's and 0's. The first player splits the board vertically into b sections, keeps one in play, and discards the rest. The second player splits the remaining portion horizontally, doing the same. After d rounds, (a depth of $2d$ ply), only one square remains. If that square contains a 1, the horizontal splitter wins. Otherwise, the vertical splitter wins. A board splitting game with a uniform random distribution of terminal values is called a P-game. A game with a clustering of similar values among neighboring leaves is called an N-game [4]. In all of these games, the board shrinks as play proceeds, making it possible to devise evaluation functions whose accuracy improve as the tree is descended. Nau used one such function, the percentage of 1's on a board, to show that P-games are pathological while N-games are not. This led him to conclude that the cause of pathology lay in the uniform random distribution of leaf values; since real games do not have this property, pathology has never been observed [Nau82a]. Similar conclusions about trees with clustering among their terminal values were reached in [Bea82] [BG82].

Pearl pointed out that minimax pathology is not simply a statistical aberration [Pea83]. The minimax procedure involves propagating functions

[4] An N-game board is set up by randomly assigning 1's and -1's to each branch in the game tree. If the sum of the branches leading from the root to a leaf is positive, a 1 is placed in the square corresponding to that leaf. Otherwise, the square gets a 0.

of estimates. In general, this is not the same as calculating the estimate of a function. The anomaly is not that P-games are pathological, but rather that chess is not! He performed an in depth error analysis of minimax, and discovered that if, as is frequently claimed, the power of lookahead lies in increased visibility, (more accurate static evaluations deeper in the tree), this increase must be at least 50% for each additional ply. Since this is almost never true in real games, he concluded that the 50% must be taken over all nodes found at the deeper level. In most games, certain positions qualify as *traps*, terminal positions that are located high in the tree, (thus called because they trap one player into an early loss). The presence of terminal nodes in the vicinity of the search frontier drastically increases the accuracy of their ancestors, and results in the necessary improvement. If the (d,b,F)-tree is modified by making every internal node a trap with probability q exceeding a certain threshold, $q \geq 1-\frac{(1+B)^{1-(1/B)}}{B}$, this improvement is reached, and pathology should be avoided [Pea84]. Abramson extended the idea of traps to any node whose W/L value could be determined exactly, or *f-wins* [Abr86]. He demonstrated experimentally that if f-wins occur with increasing densities at deeper levels in the tree, pathology can be avoided with an overall density considerably below the predicted threshold.

Michon suggested a more realistic model than the (d,b,F)-tree, the recursive random game (RRG) [Mic83]. From every position in an RRG, there are n legal moves, $(n = 0, 1, 2, ...)$, with probability f_n. The value f_0 indicates the probability that a node is a leaf, in which case it is randomly assigned either W or L. He used RRG's to analyze both pathology and quiescence. In terms of quiescence, non-quiescent positions in most games were shown to correspond to positions with relatively few options, or small branching factors. In terms of pathology, he showed that games with uniform branching factors are bound to be pathological, while games whose branching factors follow a geometric distribution are not. The various remedies to pathological behavior combine to give a solid explanation of the phenomenon: pathology occured because of oversimplifications in the original (d,b,F)-tree model. The removal of *any single* uniformity assumption resulted in a nonpathological tree: N-games targeted the uniform distribution, traps the uniform depth, and RRG's the uniform branching

factor. When f-wins removed the uniform terminal density as well, pathology became even easier to avoid.

Pathology, minimax convergence, and the last player theorem all contribute to the understanding of the minimax procedure in general, and lookahead in specific. Because these results were all derived on simplified models, it's unclear exactly how applicable their direct mathematical implications are to real games. Nevertheless, they do reveal some important points that should be as true for chess and checkers as they are for board splitting:

- Lookahead is not always beneficial.
- Improved visibility, in and of itself, is not a sufficient explanation of why lookahead is helpful (when it is).
- Values returned by lookahead to different depths should only be compared if the same player moved last in both cases.
- The more uniform the tree, the more likely it is that minimax is not the proper control strategy to be using.

4.2 Alternatives to Minimax

Perhaps the most significant outcome of the phenomena discussed in the previous section is that for the first time, the sanctity of minimax was taken to task. These challenges to the accepted standard have motivated the design of several non-minimax control strategies. To understand these strategies, it is important to recall two of the basic assumptions underlying the optimality of minimax: perfect play by both players, and accurate information (e.g. accurately evaluated tips). Since, for the most part, neither of these conditions ever holds, alternative control strategies may lead to better performance.

Pearl suggested the method of *product propagation*. This strategy assumes that the static evaluator returns the probability that a node is a forced win. If an internal node is a forced win for player I, all of it's children must be forced losses for player II, and vice versa. In other words, Pr[h is a win node] = Π(1−Pr[h' is a win node | h' is a child of h]). These

alternating products propagate the probabilities back up the tree [Pea81]. Nau showed that when this control strategy is used instead of minimax, pathology disappears [Nau83b]. Tzeng proved that given a (d,b,F)-tree with independent sibling values and an evaluation function that does, in fact, return the probability of forcing a win, product propagation will outperform any equally informed algorithm. Among the assumptions inherent to product propagation, however, is the independence of sibling nodes. Since this is clearly not true in real games, there is no reason to assume that this strategy is even reasonable in most interesting domains. Nevertheless, Nau, Purdom, and Tzeng ran some Monte Carlo experiments that demonstrated that even on N-games, (which have interdependent sibling values), product propagation played well against minimax. A strategy that averaged the values returned by product propagation and minimax (by simply adding them and dividing by 2), outperformed either strategy alone [NPT83] [TP83]. The outcomes of these experiments, however, were frequently not significant enough (statistically) to reveal anything conclusive.

Ballard developed a control strategy for searching game trees with chance nodes, **-minimax*, which assigns each chance node the average of its children's values [Bal83]. He and Reibman contended that the problem with minimax is that it erroneously assumes perfect play on the part of the opponent. They modified *-minimax to *minimax in the presence of error*. In this system, each player assigns his opponent an expected strength. This strength is used to determine subjective probabilities indicating the likelihood that a given move will be chosen. Minimax corresponds to a strength of 1 (perfect play), and *-minimax chance nodes to a strength of 0 (random play). Imperfect play should lie somewhere between the two, and can be modeled by calculating a weighted sum of the subjective probabilities [RB83]. Empirical studies performed on (d,b,F)-trees with correlated sibling values showed that this strategy outperformed the addition of a ply to minimax. Another alternative strategy is *minimum variance pruning* [Tru79]. In this strategy, nodes are assigned probability density functions (pdf's) describing the likely location of the minimax value, and the subtree with the minimum variance is expanded first. The motivation underlying this procedure is similar to that behind conspiracy search and locked value search: stable values are likely to be accurate, and small variances indicate

stable values. Although this strategy was proposed by Truscott in 1979, it has been neither fully developed nor tested, and is thus of unkown value.

5 Discussion: Relating Theory to Practice

To date, none of the strategies described in the previous section have been successfully implemented in real games. At the same time, none of the heuristics used to design type B strategies have been successfully analyzed. There are good reasons for both of these; the theoretical strategies tend to require a full-width search (pruning techniques have yet to be devised), and have generally been developed using assumptions which are not valid in real games. Determining the accuracy of a type B strategy, on the other hand, would probably require a model too complex to be analyzed. Because of their different orientations, there has been minimal interplay between the results of heuristic analysts and game designers. This isolationist tendency has, in turn, allowed game programming to flourish without ever developing a firm theoretical groundwork. There is overwhelming evidence that the minimax procedure is a reasonable control strategy, that its performance will improve with greater programming innovations, and that it will play excellent chess on fast enough machines. There is no evidence that it is in any sense correct. The discovery of pathology indicates that there are instances, albeit specialized ones, in which the traditional assumptions of minimax are false. The subsequent resolution of the phenomenon through the imposition of nonuniformity implies very definite strengths and weaknesses of the procedure; the fewer distinguishing characteristics among the nodes, the worse the performance of minimax. Viewed in this way, pathology suggests a point which should be directly applicable to real games: use minimax when there is a clear choice. Otherwise, if all values are clustered around some intermediate range, use another strategy, perhaps a weighted sum or modified M & N.

This split between theory and practice, although understandable, is somewhat disturbing. Experience dictates that the dismissal of unobserved theoretical predictions as irrelevant is unwise. In operations research, for example, the simplex method has long been used to efficiently solve linear programming applications. The proof of a worst case exponential run-

ning time, and the artificial construction of examples that caused it to run poorly, motivated other models and approaches, to the point where one has been developed which not only appears to be theoretically correct, but promises to have serious commercial potential as well [Kar84].

The incorporation of probability distributions into B*-like algorithms in [Pal85] is encouraging. Although probabilities were used in almost all the analytic studies, prior to this work they played a more or less inconsequential role in the implementations. The original motivation for Palay's use of distributions was to more accurately assess the probable location of a node's true (minimax) value than could be done with either single numbers or ranges. However, a radically different interpretation of probability distributions is possible. Point values imply the existence of a true value which is being estimated. A range implies an unknown exact value, but one that can be bounded. In sharp contrast, the use of probability distributions implies that there is no true value. Instead, the nodes are random variables which will be instantiated at different values with varying probabilities. The resultant model of partial game trees is probabilistic, rather than deterministic, in nature. In a probabilistic context, minimax is almost certainly non-optimal. The assumptions that should go into devising a control strategy for probabilistic trees include imperfect play and an unwillingness to commit to anything beyond the next move. Implicit in the minimax procedure is the statement that *if* play reaches node X, node Y (the best of X's children) *will* be chosen. If node X is not the current node, this represents a premature commitment which may or may not make sense in a deterministic domain, but certainly does not in a probabilistic one.

6 Areas for Future Investigation

Every component of partial game trees leaves many problems open. The trees represent a mathematical model that has been extensively used, but rarely studied. Throughout the course of this survey, virtually every definition in the model was shown to be vague, and every assumption was questioned at least once. The areas of difficulty that relate most directly to the design of control strategies (in no particular order) are:

- The static evaluator: Is an inexact goal necessary, or can search truly be guided effectively toward a number whose meaning is vague? Are there features of the tree which can serve as inexact goals, or must the information be game specific?

- The role of knowledge: How much game specific information is really required to design a successful program? Should control strategies be defined for trees (the general model) or for games (the specific)? Can planning be used effectively to play games at expert levels, or only to augment more standard search techniques? Could the idea of an advice table be extended to include human interaction? Would a domain like chess lend itself to a knowledge-based expert system, in which strategy decisions were based on responses learned from experts?

- The limitations of minimax: Does the leap of type A systems beyond the master level indicate that there are no limits? Can the technology be developed to push these systems to grandmaster level?

- The mathematical model: What is the correct model for two-player zero-sum games of perfect information? Is it probabilistic or deterministic in nature? What is the optimal control strategy for making decisions in this model? How closely does this model approximate real games? What is the relationship between partial and complete game trees?

- The role of lookahead: Why is lookahead advantageous? Under what conditions will it not help? What is the correct criterion for the termination of search? How can quiescent nodes on the search frontier be recognized? Can the horizon effect be avoided completely, and if so, how?

Hopefully, within the next few years, a new general theory of partial game trees will begin to answer some of these questions. This type of theoretical groundwork will have profound ramifications in both the analysis of heuristics and the design of games, and should be actively pursued by practitioners of both fields.

References

[AAD75] G.M. Adelson-Velskiy, V.L. Arlazarov, and M.V. Donskoy. Some methods of controlling the tree search in chess programs. *Artificial Intelligence*, 6:361–371, 1975.

[Abr86] Bruce Abramson. An explanation of and cure for minimax pathology. In Laveen Kanal and John Lemmer, editors, *Uncertainty in Artificial Intelligence*, North Holland, 1986.

[Bal83] Bruce W. Ballard. The *-minimax procedure for trees containing chance nodes. *Artificial Intelligence*, 21:327–350, 1983.

[BCG82] Elwyn Berlekamp, John H. Conway, and Richard K. Guy. *Winning Ways.* Academic Press, 1982. In two volumes.

[BE86] Hans Berliner and Carl Ebeling. The suprem architecture: a new intelligent paradigm. *Artificial Intelligence*, 28:3–8, 1986.

[Bea80] D.F. Beal. An analysis of minimax. In M.R.B. Clarke, editor, *Advances in Computer Chess 2*, Edinburgh University Press, 1980.

[Bea82] D.F. Beal. Benefits of minimax search. In M.R.B. Clarke, editor, *Advances in Computer Chess 3*, Edinburgh University Press, 1982.

[Ber73] Hans J. Berliner. Some necessary conditions for a master chess program. In *Proceedings of the 3rd International Joint Conference on Artificial Intelligence*, pages 77–85, 1973.

[Ber77a] Hans Berliner. A representation and some mechanisms for a problem-solving chess program. In M.R.B. Clarke, editor, *Advances in Computer Chess*, Edinburgh University Press, 1977.

[Ber77b] Hans Berliner. The use of domain-dependent descriptions in tree searching. In *Perspectives on Computer Science*, Academic Press, 1977.

[Ber78] Hans J. Berliner. A chronology of computer chess and its literature. *Artificial Intelligence*, 10:201–214, 1978.

[Ber79] Hans Berliner. The b* tree search algorithm: a best-first proof procedure. *Artificial Intelligence*, 21:23–40, 1979.

[Ber80] Hans J. Berliner. Backgammon computer program beats world champion. *Artificial Intelligence*, 14:205–220, 1980.

[Ber81] Hans J. Berliner. An examination of brute force intelligence. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence*, pages 581–587, 1981.

[BG82] I. Bratko and M. Gams. Error analysis of the minimax principle. In M.R.B. Clarke, editor, *Advances in Computer Chess 3*, Edinburgh University Press, 1982.

[BK77] J.A. Birmingham and P. Kent. Tree searching and tree pruning techniques. In M.R.B. Clarke, editor, *Advances in Computer Chess*, Edinburgh University Press, 1977.

[BM80] I. Bratko and D. Michie. An advice program for a complex chess programming task. *The Computer Journal*, 23:353–359, 1980.

[Bot70] M.M. Botvinnik. *Computers, Chess, and Long Range Planning.* Springer-Verlag, 1970. trans. A. Brown.

[Bot82] M.M. Botvinnik. Decision making and computers. In M.R.B. Clarke, editor, *Advances in Computer Chess 3*, Edinburgh University Press, 1982.

[Bot84] M.M. Botvinnik. *Computers in Chess: Solving Inexact Search Problems.* Springer-Verlag, 1984. trans. A. Brown.

[Bra80] M.A. Bramer. An optimal algorithm for king and pawn against king using pattern knowledge. In M.R.B. Clarke, editor, *Advances in Computer Chess 2*, Edinburgh University Press, 1980.

[Bra83] M.A. Bramer, editor. *Computer Game Playing: Theory and Practice.* Ellis Horwood Limited, 1983.

[CM83] M.S. Campbell and T.A. Marsland. A comparison of minimax tree search algorithms. *Artificial Intelligence*, 20:347–367, 1983.

[CT77] J.H. Condon and K. Thompson. Belle. In P.W. Frey, editor, *Chess Skill in Man and Machine*, Springer-Verlag, 1977.

[CT82] J.H. Condon and K. Thompson. Belle chess hardware. In M.R.B. Clarke, editor, *Advances in Computer Chess 3*, Edinburgh University Press, 1982.

[EH63] D. Edwards and T. Hart. *The Alpha-Beta Heuristic.* Technical Report 30, MIT AI Memo, October 1963. Originally published as the Tree Prune Algorithm, Dec. 1961.

[FGG73] S. Fuller, J. Gaschnig, and J. Gillogly. *Analysis of the Alpha-Beta Pruning Algorithm.* Technical Report, Carnegie-Mellon University, 1973.

[Gre67] R.D. Greenblatt. The greenblatt chess program. In *Proceedings of the 1967 Fall Joint Computer Conference*, pages 801–810, 1967.

[Gri74] A.K. Griffith. A comparison and evaluation of three machine learning procedures as applied to the game of checkers. *Artificial Intelligence*, 5:137–148, 1974.

[Har74] Larry R. Harris. The heuristic search under conditions of error. *Artificial Intelligence*, 5:217–234, 1974.

[Kai83] H. Kaindl. Quiescence search in computer chess. In M.A. Bramer, editor, *Computer Game Playing: Theory and Practice*, Ellis Horwood Limited, 1983.

[Kar84] N. Karmakar. A new polynomial time algorithm for linear programming. In *Proceedings of the 16th Annual ACM Symposium on the Theory of Computing*, pages 302–311, 1984.

[KM75] Donald E. Knuth and Ronald W. Moore. An analysis of alpha-beta pruning. *Artificial Intelligence*, 6:293–326, 1975.

[Kop85] Danny Kopec. Chess computers. *Abacus*, 2:10–28, 1985.

[Kor85] Richard E. Korf. Depth-first iterative-deepening: an optimal admissible tree search. *Artificial Intelligence*, 27:97–109, 1985.

[MC82] T.A. Marsland and M. Campbell. Parallel search of strongly ordered game trees. *Computing Surveys*, 14:533–551, 1982.

[McA85] David A. McAllester. *A New Procedure for Growing Min-Max Trees.* Technical Report, MIT Artificial Intelligence Laboratory, July 1985.

[Mic83] Gerard P. Michon. *Recursive Random Games: A Probabilistic Model for Perfect Information Games.* PhD thesis, University of California at Los Angeles, 1983.

[Nau82a] Dana S. Nau. An investigation of the causes of pathology in games. *Artificial Intelligence*, 19:257–278, 1982.

[Nau82b] Dana S. Nau. The last player theorem. *Artificial Intelligence*, 18:53–65, 1982.

[Nau83a] Dana S. Nau. Decision quality as a function of search depth on game trees. *JACM*, 30:687–708, 1983.

[Nau83b] Dana S. Nau. Pathology on game trees revisited, and an alternative to minimax. *Artificial Intelligence*, 21:221–244, 1983.

[New75] Monroe Newborn. *Computer Chess.* Academic Press, 1975.

[NM44] John Von Neumann and Oskar Morgenstern. *Theory of Games and Economic Behavior.* Princeton University Press, 1944.

[NPT83] Dana S. Nau, Paul Purdom, and Chun-Hung Tzeng. *Experiments on Alternatives to Minimax.* Technical Report, University of Maryland, October 1983.

[NSS63] Allen Newell, J.C. Shaw, and H.A. Simon. Chess playing programs and the problem of complexity. In E. Feigenbaum and J. Feldman, editors, *Computers and Thought*, McGraw-Hill, 1963.

[Pal82] Andrew J. Palay. The b* tree search algorithm — new results. *Artificial Intelligence*, 19:145–163, 1982.

[Pal85] Andrew J. Palay. *Searching With Probabilities.* Pitman, 1985.

[Pea80] Judea Pearl. Asymptotic properties of minimax trees and game-searching procedures. *Artificial Intelligence*, 14:113–138, 1980.

[Pea81] Judea Pearl. Heuristic search theory: a survey of recent results. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence*, pages 24–28, 1981.

[Pea82] Judea Pearl. The solution for the branching factor of the alpha-beta pruning algorithm and its optimality. *CACM*, 25:559–564, 1982.

[Pea83] Judea Pearl. On the nature of pathology in game searching. *Artificial Intelligence*, 20:427–453, 1983.

[Pea84] Judea Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving.* Addison Wesley, 1984.

[Pit77] Jacques Pitrat. A chess combination program which uses plans. *Artificial Intelligence*, 21:275–321, 1977.

[Pit80] J. Pitrat. The behaviour of a chess combination program using plans. In M.R.B. Clarke, editor, *Advances in Computer Chess 2*, Edinburgh University Press, 1980.

[RB83] Andrew L. Reibman and Bruce W. Ballard. Non-minimax search strategies for use against fallible opponents. In *Proceedings of the 8th International Joint Conference on Artificial Intelligence*, pages 338–342, 1983.

[Ros82] Paul S. Rosenbloom. A world-championship-level othello program. *Artificial Intelligence*, 19:279–320, 1982.

[RP83] Igor Roizen and Judea Pearl. A minimax algorithm better than alpha-beta? yes and no. *Artificial Intelligence*, 21:199–220, 1983.

[SA77] David J. Slate and Lawrence R. Atkin. Chess 4.5 — the northwestern university chess program. In P.W. Frey, editor, *Chess Skill in Man and Machine*, Springer-Verlag, 1977.

[Sam63] A.L. Samuel. Some studies in machine learning using the game of checkers. In E. Feigenbaum and J. Feldman, editors, *Computers and Thought*, McGraw-Hill, 1963.

[Sam67] A.L. Samuel. Some studies in machine learning using the game of checkers ii — recent progress. *IBM J. Res. Dev.*, 11:601–617, 1967.

[SD69] James R. Slagle and John K. Dixon. Experiments with some programs that search trees. *JACM*, 16:189–207, 1969.

[SD70] James R. Slagle and John K. Dixon. Experiments with the m & n tree-searching procedure. *CACM*, 13:147–154, 1970.

[Sha50] Claude E. Shannon. Programming a computer for playing chess. *Philosophical Magazine*, 41:256–275, 1950.

[Sto79] G.C. Stockman. A minimax algorithm better than alpha-beta? *Artificial Intelligence*, 12:179–196, 1979.

[Tho82] K. Thompson. Computer chess strength. In M.R.B. Clarke, editor, *Advances in Computer Chess 3*, Edinburgh University Press, 1982.

[TP83] Chun-Hung Tzeng and Paul W. Purdom. A theory of game trees. In *Proceedings of the 8th International Joint Conference on Artificial Intelligence*, pages 416–419, 1983.

[Tru79] Tom Truscott. Minimum variance tree searching. In *Proceedings of the 1st International Symposium on Policy Analysis and Information Systems*, pages 203–209, 1979.

[Wil77] David Wilkins. Using chess knowledge to reduce search. In P.W. Frey, editor, *Chess Skill in Man and Machine*, Springer-Verlag, 1977.

[Wil80] David Wilkins. Using patterns and plans in chess. *Artificial Intelligence*, 14:165–203, 1980.

[Wil82] David Wilkins. Using knowledge to control tree searching. *Artificial Intelligence*, 18:1–51, 1982.

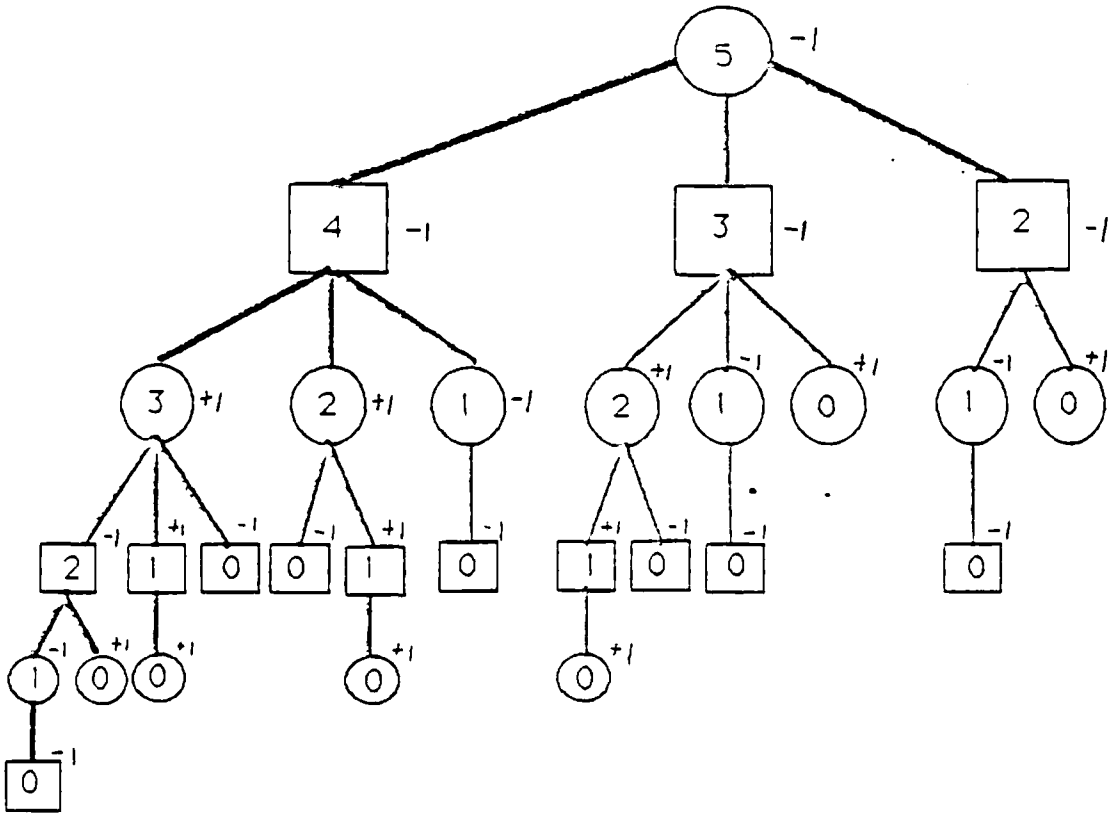

Figure 1: The game of Nim is played with 5 stones. The players take turns removing 1,2, or 3 stones from play. The player removing the last stone loses. Thus, a circle containing a 0 is a win for circle (Max), and a square with a 0 is a win for square (Min). Max wins are denoted by +1, Min wins by -1, and the minimax value of each node is drawn to the right of the node. Since the root has a minimax value of -1, Nim is a forced win for the player moving second (Min).

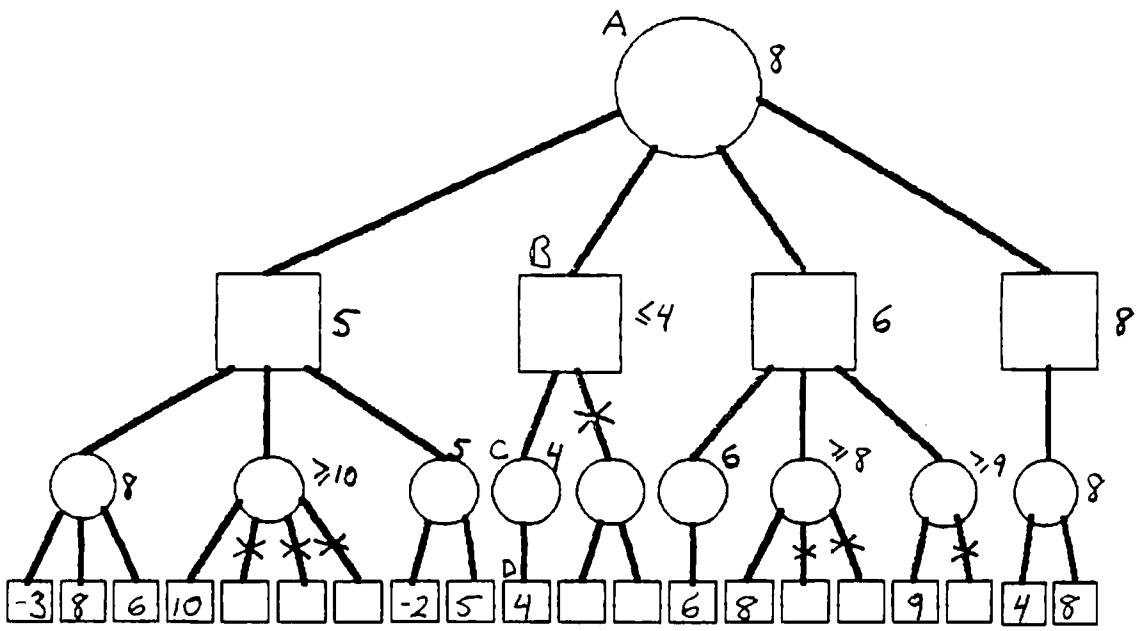

Figure 2 - Minimax Search with α-β pruning: Static values are drawn inside the nodes, minimax values outside and to the right. Branches with X's through them have been pruned. To understand how a branch is pruned, consider nodes A,B,C, and D. D is a leaf, statically evaluated at 4. Since D is the only child of C, C gets a minimax value of 4. This means that B, (a Min node), must have a value no greater than 4. Since A, (a Max node) already has a child valued at 5, B will not be chosen, and its remaining children can be pruned.

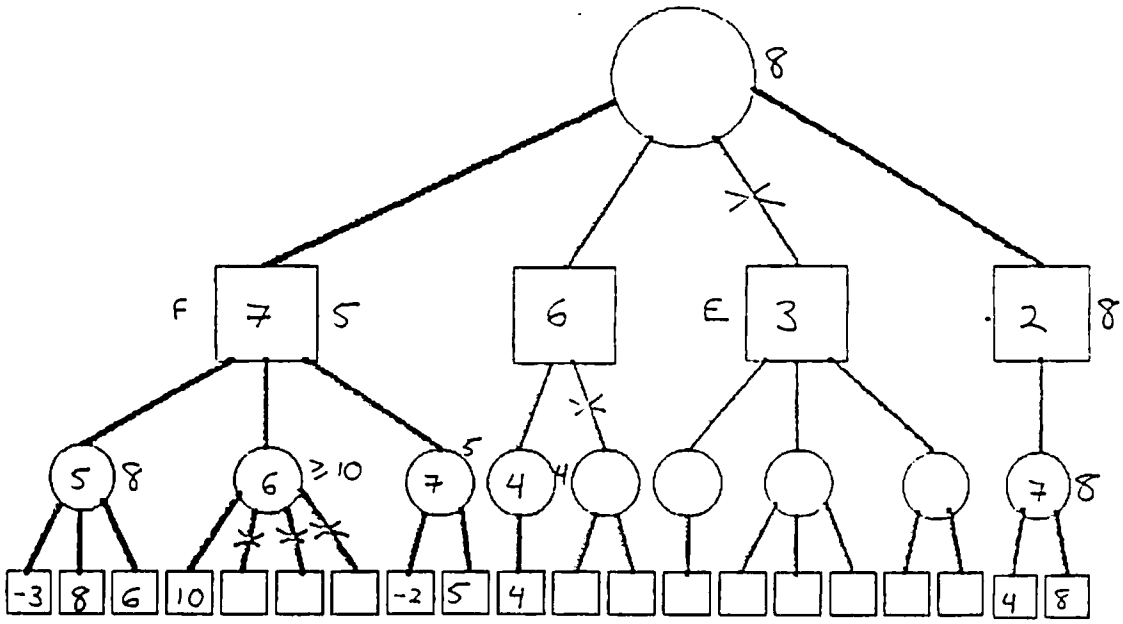

Figure 3: This is what happens when razoring is added to the pruning in example 2. Node E's static evaluation was 4, which is worse than node F's backed up value of 5. Thus, it was pruned

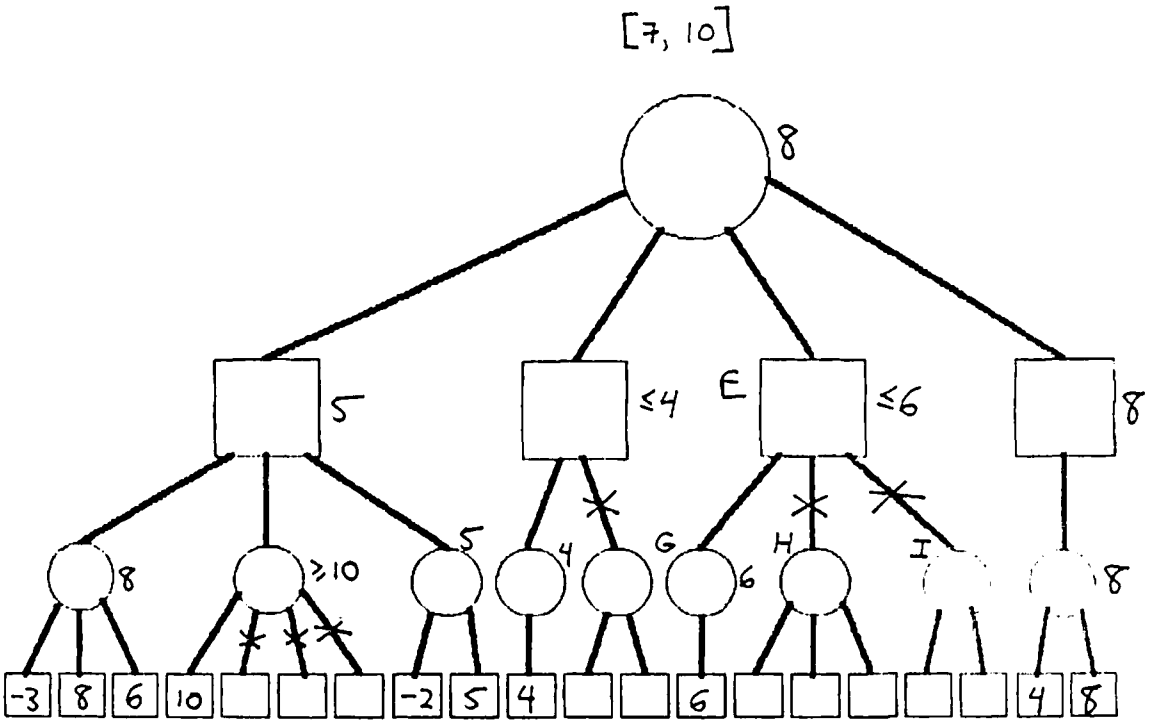

Figure 4. The α-β pruning of figure 2 is now augmented with an aspiration search. The precomputed range of possible values for the root is [7, 10]. When node G is expanded, it becomes clear that the value of node E will not exceed 6. Since this falls outside the range of possible values, nodes H and I can be pruned.

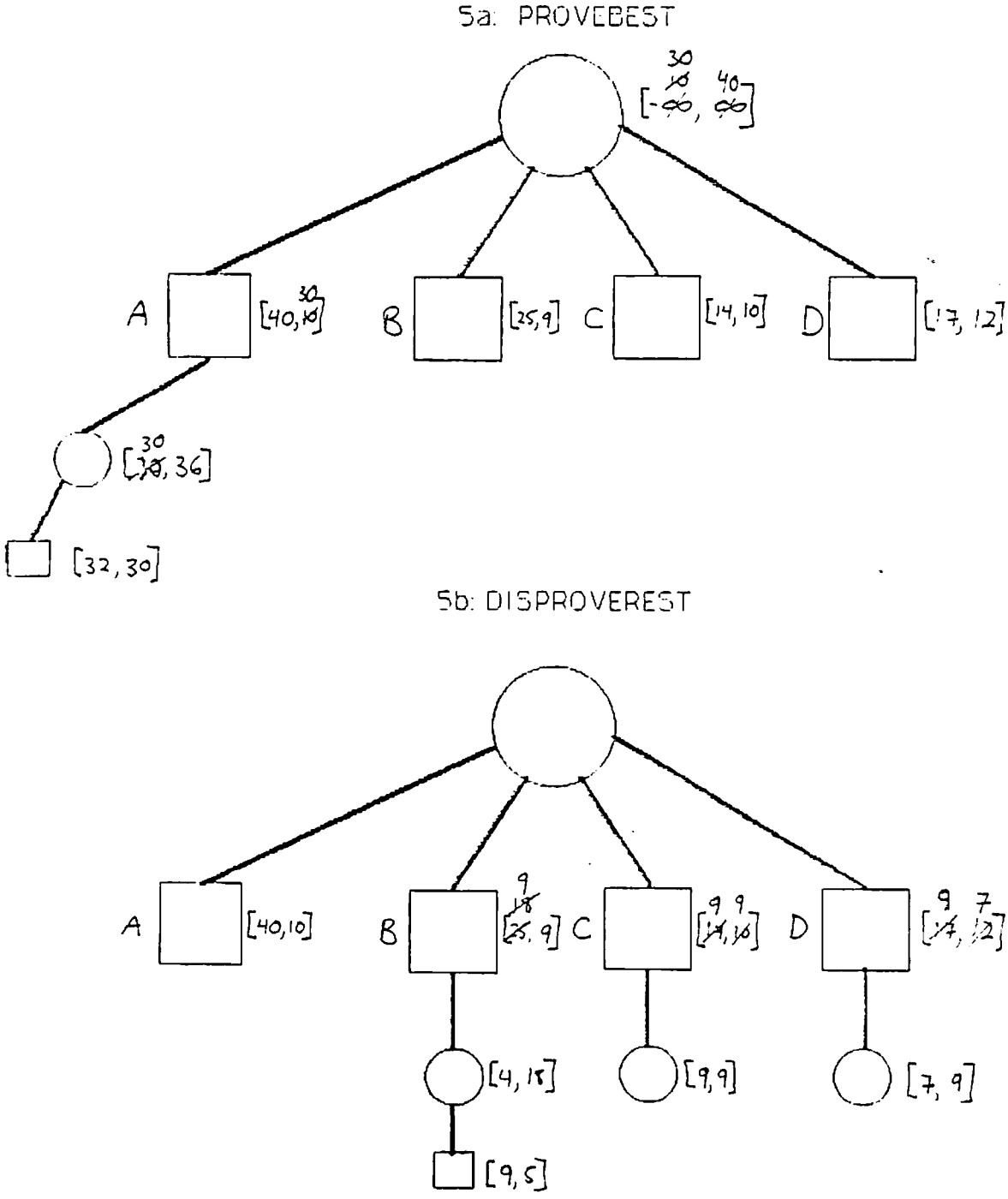

Figure 5 - The proof procedures of B*: Node A has the highest upper bound, and is thus the most promising node. In 5a, A is expanded until its lower bound is greater than the upper bounds of nodes B,C, and D. In 5b, the upper bounds of B,C, and D are pushed below the lower bound of A.

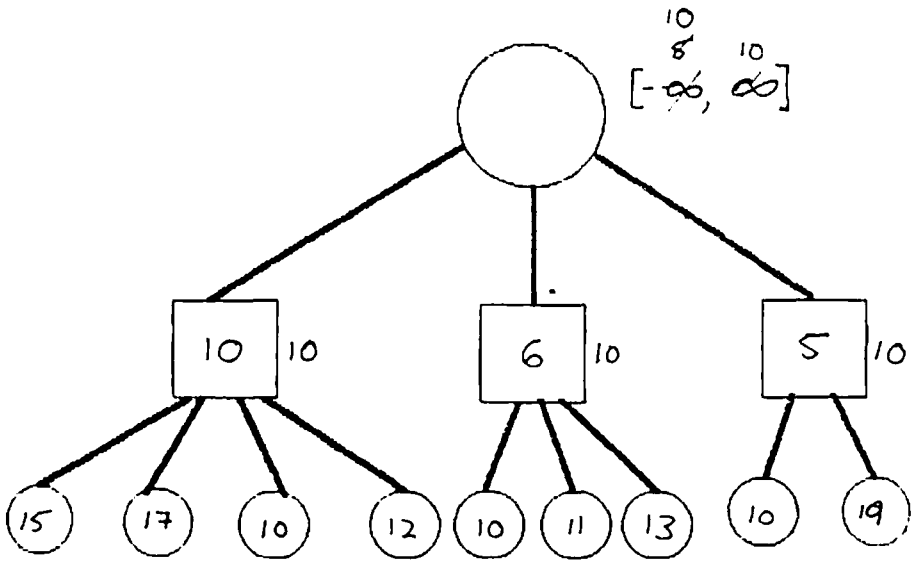

Figure 6 - Conspiracy search. In this example, a node is considered stable if 3 leaves must change to affect it. When the tree is built out to depth 1, the range of possible values is [5,10]. When built out another level, all three of the root's children get backed up values of 10. In order for the root value not to be 10, then, 3 leaves must change (1 grandchild from each child). Thus, the root is considered stable at 10.