The Expected-Outcome Model of Two-Player Games

Bruce Abramson

Submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in the Graduate School of Arts and Sciences
COLUMBIA UNIVERSITY
1987

ABSTRACT

The Expected-Outcome Model of Two-Player Games

Bruce Abramson

Long before computer games became popular recreations, mathematicians viewed games as models of decision making. The general understanding of decisions, however, has been impeded by the ambiguity of some of the basic components of game-tree search. In particular, the static evaluation function, or determination of a node's merit based on directly detectable features, has never been adequately defined. The *expected-outcome* model proposes that the appropriate value to assign a node is the expected value of a game's outcome given random play from that node on. This proposal is supported by both analytic proofs and experimental evidence: the model's optimality on a class of simple game-trees is derived analytically, and the popular statistical techniques of random sampling and regression analysis produce efficient expected-outcome estimators in real games. Overall, the expected-outcome model of two-player games is shown to be precise, accurate, easily estimable, efficiently calculable, and domain-independent.

Contents

List of Figures

List of Tables

Acknowledgments

This dissertation involved a great deal of time and effort, and I'm grateful to many people for both helping hands and shoulders to cry on.

First and foremost, I'd like to thank my thesis advisor, Richard Korf, who gave me support and encouragement throughout my graduate career, and who invited me to join him at UCLA when he decided to move west. A close second, however, would have to be Jonathan Gross. As my Columbia liaison and official advisor, Jonathan made it possible for me to remain a Columbia student while in residence at UCLA.

Over the years, I've gotten useful tips from many of my colleagues, in particular Othar Hansson, Andrew Mayer, and Mordechi Yung, who were helpful and interested beyond the call of duty. I've also lucked out with five great officemates, Jed Schwartz, Jon Rynn, Russell Mills, Arthur Goldberg, and Maria Pozzo, who've helped create a series of comfortable working environments.

Then, of course, there are my parents, grandparents, sisters, brother-in-law, friends, lovers, and all the folks who alternately let me ramble on about my work and pretend that it didn't exist, and put up with all of my neuroses, (only some of which can be blamed on this thesis).

Finally, if there's anyone else out there that I talked to between mid-1983 and mid-1987, thanks. I made it.

Part I
Introduction

1 Prelude: Why Study Games?

The mathematical study of games predates digital computers by several decades [NM44]. With the coming of the computer age, the marriage of games, as decision making models, to computers, as decision making machines, was swift and natural. In 1950, with computer science in its infancy and the term "artificial intelligence" as yet unborn, Claude Shannon's *Programming a Computer for Playing Chess* [Sha50] begat the computer game. In this classic work, Shannon justified chess programming as a valid scientific pursuit 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; in any of these fields, skillful performance requires thought, and satisfactory solutions, although generally attainable, are rarely trivial[1]. In addition, he pointed out that chess has an attractive feature that its more complex relatives lack: both the options (moves) and the goal (checkmate) are sharply

[1]The idea of studying simplified models is characteristic of most basic research in artificial intelligence. Battlefield management and stock-market investment are two examples of decision-making in (semi-)adversarial settings. One of the motivations behind studying games is that they model the decisions and the adversary, without worrying about the complexities of the particular domain.

defined.

The original objective behind game programming, then, was to understand how decisions are made. To this end, researchers have discovered some general algorithms that have led to the development of powerful, albeit domain-specific programs, perhaps culminating in special purpose chess architectures that were able to compete at the master level [BE86] [CT82]. Despite the superiority of these machines to any existing software-based programs, their implicit requirement of a machine per domain is clearly unacceptable. Thus, the understanding (and hopefully the eventual automation) of the decision-making process requires an ongoing investigation of the underlying principles, definitions, and algorithms for searching simple models such as game-trees.

One particular key question faced by decision makers that has never been adequately addressed is how potential domain configurations can be quantified. Without a reasonable assignment of values, comparison of, and choices among alternative possible states, is impossible. In game-theoretic terminology, this assignment is known as a *static evaluation function*. In the past, all game evaluators have concentrated on defining aspects of a specific domain that relate to the ultimate goal of winning. No effort has been expended seeking an invariant trait of the genus *ludus* (game) that could serve as the paradigm of evaluators.

The focal point of this research has been the determination of an unambigu-

ous, domain-independent model of the static evaluation function. The proposed definition, the projected outcome of a game given random play, leads to the *expected-outcome* model of two-player evaluators, in which the relative merit of game-tree nodes, rather than board positions, is considered. Since game-trees are general mathematical models while game boards are domain-specific state descriptions, this distinction immediately increases a function's extensibility from a specific game to the class of two-player zero-sum games of perfect information. In addition, the introduction of the random-play assumption suggests a radical rethinking of virtually every component of game-tree search. The resultant model of the game-tree is essentially probabilistic in nature, rather than deterministic. Node values become random variables, and search procedures attempt to maximize expected values. In the long run, this model may be applicable to a wide variety of domains, and may begin bridging the gap between simple parlor games and their more complex relatives. For the moment, however, the redefinition of node values in game-trees as probabilistic entities helps resolve one of the most basic and ambiguous components of two-player search systems by giving a general characterization of static evaluation.

2 Overview: What Lies Ahead?

This document is divided into three parts. Part I, the Introduction, explains the fundamental problems that motivated the research. In particular, section 3 contains background information which briefly outlines some previous artificial intelligence research efforts on game-trees. The thrust of this survey is that the ambiguity of static evaluation sends a ripple of problems through the search system: functions can't be compared, backup algorithms can't be justified, and frontiers can't be set intelligently. Although each of these shortcomings has been discussed in the past, the effects that they have on the performance of game-playing programs is frequently unidentifiable; it is not difficult to ignore one or more of them when designing a program that plays a specific game. Since the primary objective of this work was the development of a domain-independent static evaluator, however, all three had to be considered.

Part II, the Model, opens with the definition of a node's *expected-outcome* value as the expected value of the leaves beneath it. The basic model is outlined in section 4. Unlike all previous models, these functions are probabilistic in nature. Game-trees have long been viewed in a strictly deterministic context; leaves have exact values corresponding to their payoffs, and internal nodes have exact values determined by the minimax algorithm [NM44]. Under the expected-outcome model, only leaf values are exact. Internal nodes are viewed as random

variables whose expected values depend on the distribution of leaves beneath them. This definition immediately offers a criterion for evaluator comparison: the more closely a function approximates the expected value, the stronger it is. In addition, the obvious method of approximating expected values is random sampling. Rather than performing a full-width search as deeply as possible, sampling strategies rely on as many full-depth searches as time permits. By searching a randomly chosen subset of the paths between the node being evaluated and the leaves, full-depth searches suggest some directions in which valid justifications of backup algorithms and criteria for setting search frontiers may lie. These are outlined briefly in section 7. Unfortunately, although the new model addresses some major issues left hanging by the standard definitions, it also introduces its own set of problems. Perhaps the two most pressing charges against it involve the inaccuracy of the implicit random-play assumption and the relative inefficiency of random samplers. Since accuracy and efficiency are the two major concerns in system (e.g., model or program) design, these are potentially damning objections.

In an attempt to dispel any skepticism brought about by these (or other) difficulties, the rest of part II scrutinizes both the rationality of the random-play assumption and the validity of the expected-outcome model. The studies described in section 5 answer four questions, all in the affirmative:

- Is the model's ideal implementation optimal on simplified game-trees?

- Do completely informed (exact) expected-outcome functions make good decisions in small games?
- Can estimated functions compete favorably within the realm of expert-designed evaluators?
- Is expected-outcome useful in the design of efficiently calculable static evaluation functions?

Section 5.1 analyzes the model's theoretical accuracy on some simple uniform game-trees. Of course, interesting games generate trees that are much too irregular to be analyzed, and thus the applicability of the model to real domains must be discussed empirically.

Section 5.2.1 considers the performance of exact expected-outcome functions on some small variants of tic-tac-toe and Othello. Since optimal moves are calculable in small trees, there is an absolute standard against which a function's performance can be judged. Thus, these experiments not only investigate the relative standing of expected-outcome and some expert-designed evaluators with respect to the frequency with which they move correctly, but determine each function's absolute decision quality, as well. These tests, however, are limited to games that are small enough to be searched exhaustively. Thus, section 5.2.2 studies the performance of an estimated expected-outcome function that com-

putes the average value of a randomly sampled subset of leaves beneath each node. To compare the levels of play achieved by generated and designed functions, the estimator is pitted directly (no lookahead) against an expert-designed evaluator on the full (8-by-8) game of Othello. Unfortunately, the cost of implementing the random sampler is prohibitive, thereby motivating section 5.2.3's use of regression analysis on the sampler's estimates to produce an *efficient* expected-outcome function. This technique automatically learns coefficients that define polynomial evaluators for Othello and chess; tournament play among a group of related functions indicates that even under this approximation, twice removed from the original definition, the model still performs well.

These experiments bring the reader up to date on the expected-outcome model. Part III lists some of its potential future extensions, summarizes what has been done already, and recaps the major contributions of the research.

3 Background: What is Known Already?

Although many key questions in the study of two-player games have never been adequately addressed, there are quite a few that have been. In order to fully appreciate the place that games hold in artificial intelligence research, it is necessary to backtrack a bit and examine work that has been done on them in the past. Throughout this discussion, the importance of developing a well un-

derstood model for the static evaluator will be stressed. It will be shown that without a reasonable definition of evaluation functions, a series of ambiguities that pervades the search system leads to many of the major problems that have been identified by game researchers.

3.1 Preliminaries

One of the most widely applied problem-solving techniques in artificial intelligence is the heuristic search of state-space graphs. Nodes in a state-space graph represent states of the world, and arcs indicate the legal transformations between states. In a graph, every node is unique; no two nodes describe the same world. Sometimes, however, space considerations make it necessary to consider a state that can be arrived at through two (or N) distinct sets of legal transformations as two (or N) nodes. This type of graph is called a *tree*. Games are modeled with a special class of trees called *game-trees*, in which the nodes are board configurations, the arcs legal moves, and the players take turns moving. Game-trees are the general mathematical model on which the theory of *two-player zero-sum games of perfect information* is based [NM44]. Many popular parlor games, such as chess, checkers, Othello, tic-tac-toe, and Go belong to this class of games; they are perfect information because all legal moves are known to both players at all times, and zero-sum because one player's loss equals the other's gain. An

example of the game-tree of 5-stone Nim, (a simple game which forms the basis of much of the mathematical theory of games [BCG82]), is shown in figure 1.

At the top of a game-tree is a single node, known as the *initial state,* (or *root*), 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 initial state 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 that 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 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 is over. 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 that tells a player which move to choose is a strategy for controlling the flow of the game, or a *control strategy.* In principle, the procedure of assigning values to nodes and then choosing a move 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 made. This method of assigning values and choosing moves is called the *(complete) minimax algorithm,* and it defines the optimal move to be made from each state in the game [NM44].

3.2 Heuristic Evaluation Functions

The applicability of optimal control strategies, such as complete minimax, is dependent on a tree's size. Some state-space graphs, such as the Nim tree, are small enough for every node to be examined. Interesting problems, however, tend to define graphs which are too large to be searched exhaustively[2]. Thus, some

[2]For example, complete trees have been estimated at 7^{60} ($\approx 10^{36}$) nodes for Othello, 10^{40} for checkers, 10^{120} for chess, and 361! ($> 10^{650}$) for Go.

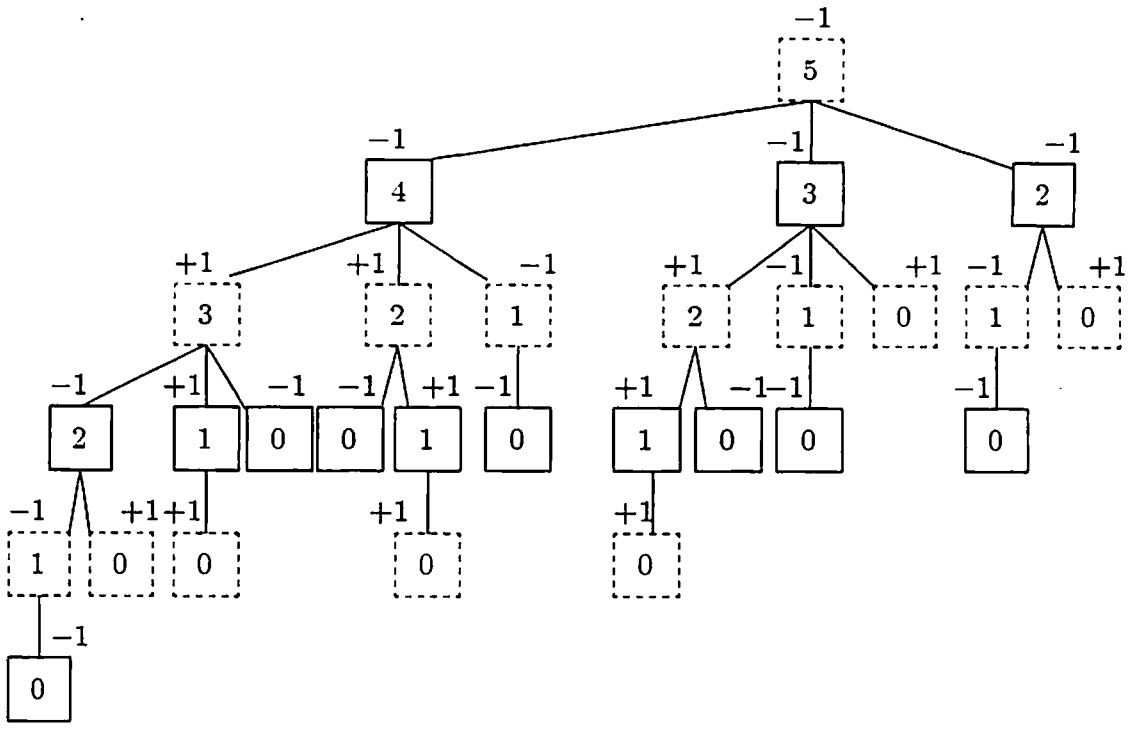


Figure 1: The game of Nim is played with piles of stones. In this version, the players take turns removing 1, 2, or 3 stones from the initial pile, (in this case, of five stones), and the player removing the last stone loses. The numbers in the squares indicate the size of the pile at the beginning of the player's turn. If MAX (dashed box) victories are denoted by $+1$, and MIN (solid box) victories by -1, then the complete minimax algorithm assigns the values shown outside the boxes. Since the root value is -1, 5-stone Nim is a forced win for MIN.

nodes must be ignored. Guidelines that indicate which nodes should be ignored and which should be examined result in a systematic search of the graph. Perhaps the simplest guideline suggests extending search depth uniformly to view as many nodes as time permits; in trees whose leaves are too far from the root for exhaustive searches to be performed, search to some limit, (generally set by the computational power available), and study the *tip nodes* at the search frontier instead of the leaves. When the tip nodes are leaves, the values are exact and the tree is *complete*. Otherwise, the tips are internal nodes, and the tree is *partial*[3]. Complete trees are well understood but rarely applicable — applications must rely on partial trees. Since tip nodes in partial game trees do not have immediately discernible payoffs associated with them, additional guidelines are necessary to decide on the proper heuristic estimate to give them, which paths should be searched further, and how the estimates should combine to recommend a move. The most desirable heuristics are easy to calculate, highly accurate, applicable to many problems, and lead to good solutions. Heuristic search theory is, in large part, the study of necessary tradeoffs among these features and the design of heuristics that combine them in the desired proportions.

One of the major components of the theory of heuristic search is the evaluation

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

of nodes using only *static* information, (features directly detectable in the state) to determine their relative merit. Static evaluators have been studied in two fundamentally different contexts: single-agent (i.e., puzzles) and dual-agent (i.e., games). One essential difference between these settings is the objective of the search system. In one-player puzzles, the standard goal is to reach a specific state via the least expensive path, whereas the goal of a typical two-player game is to be in a winning state when the game ends. The intuitive single-agent evaluator, then, is an estimate of the cost of the cheapest path from a given state to the goal. This fairly rigorous definition offers a method for comparing the accuracy of any two functions proposed for the same task — the more accurate the estimate, the better the function — and has spawned many studies which relate the accuracy of an evaluator to the solution quality and algorithmic complexity of heuristic searches that use it [Pea84]. For example, one single-agent domain that has received a great deal of attention is the 8-puzzle. This puzzle consists of eight tiles numbered 1 through 8 arranged in a 3-by-3 grid, with one space in the grid left blank. The objective is to transform an arbitrary initial configuration into a given final configuration by sliding tiles either horizontally or vertically into the blank space. Each move, of course, shifts the blank to the location vacated by the most recently moved tile, and opens up new possibilities for the next move. The goal of the search is to effect the desired transformation in as few moves as

possible. A popular static evaluator, Manhattan Distance [Pea84], develops an estimate of the number of moves needed by ignoring the rule that only allows tiles to be shifted into the blank space. With this restriction gone, the number of moves needed to change one configuration to another is the sum, over all tiles, of the number of rows plus the number of columns separating a tile from its desired final position. For a detailed discussion of the 8-puzzle and Manhattan Distance, see [Pea84].

Since the goal of a two-player game is to win, a similar intuitive definition of the two-player evaluator should be an estimate of whether a given node will result in a winning state. Unfortunately, no firm understanding of what this means has ever been developed. The literature has been uniformly vague in its interpretation of game evaluation functions, describing them with a series of equivalent qualitative terms, including measures of a position's "worth"[Nil80], "merit", "strength" [Pea84], "quality"[Win77], or "promise"[Ric83]. For example, the best known two-player evaluation function is material advantage in chess. In this family of chess functions each chessman is assigned a certain weight, say one for pawns, three for knights and bishops, five for rooks, and nine for queens. On any board, each player's score is given by the weighted sum of his pieces. The value of the board is the difference between black's score and white's score. The justification behind using this evaluator is that when combined with several

other chess features, material advantage should correspond to board strength [Sha50]. This is, of course, rather imprecise; the closest thing to a precise task that has ever been proposed for two-player evaluators is that the ideal function should return a node's complete minimax value, and an heuristic function should estimate it. The difficulty with this definition is that there are no known general procedures for estimating minimax values, judging heuristic quality, comparing different functions, or learning static evaluators. Proponents of this proposal have claimed that minimax values can be approximated by considering the ease with which a game can be won [Sha50]. Exactly what this means, why it should be so, and how it could be calculated, however, are questions that have never been satisfactorily answered.

Without an adequate understanding of what an evaluator is attempting to estimate, it is impossible to devise a general guideline for evaluator design. This, in turn, points out another deficiency in the standard approach to two-player games: there is no known a priori method for determining function accuracy. In a game whose leaf values are restricted to WIN and LOSS, an ideal evaluator would return the appropriate value. Thus, the implicit task of such an evaluator is classification, and a function's strength should be directly proportional to its ability to correctly identify internal WINs and LOSSes. The obvious way to determine strength, then, is to divide the range of returned values into two

classes and calculate the percentage of nodes that were correctly identified. One problem with this approach is that it completely ignores the issue of decision quality, or the frequency with which the best move is made. Since the a priori notion of evaluator strength is only useful if it can serve as a useful predictor of the a posteriori notion of evaluator performance, the quality of a function's decisions is crucial to predicting its strength.

To see what can happen if decision quality is ignored, consider two evaluators for the same game, A and B. Function A classifies 90% of all nodes correctly, but most of the erroneous 10% are LOSSes which are given very high values (or WINs given very low ones). Any time one of these LOSSes is available it will be selected, despite the existence of many WINs which were correctly classified but given lower values. Function B, on the other hand, only classifies 60% of the nodes accurately, but most of its errors occur in fields of similar siblings. For example, when given a choice of eight WINs and two LOSSes, B may correctly classify only the two LOSSes and four of the WINs; four WINs will appear as LOSSes, but an optimal move will still be made. Thus, B's errors rarely affect play, and its decision quality is rather strong. If two such functions were pitted against each other, B would probably win more games, thereby reversing the predictions that were based on classification strength. This is illustrated in figure 2.

Although the above scenario appears somewhat contrived, it may actually

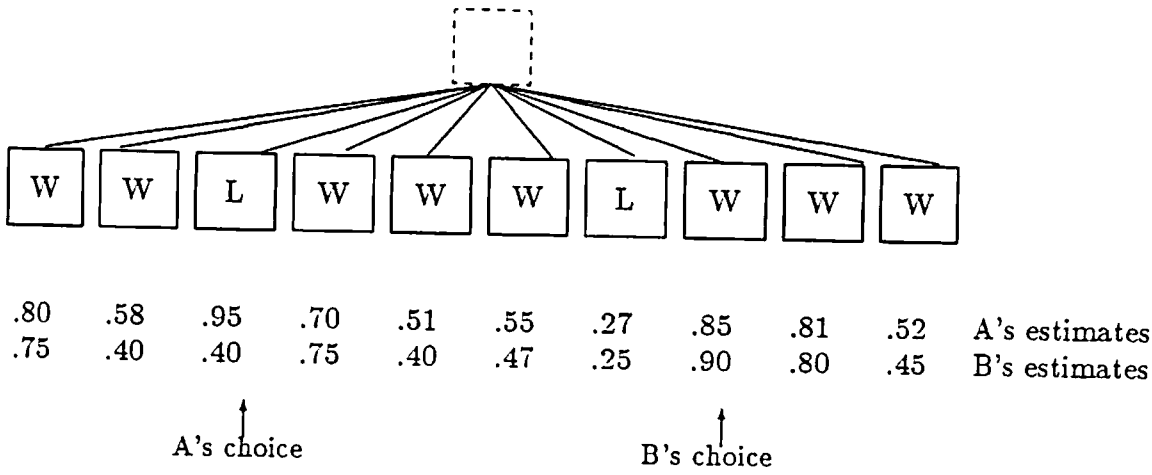


Figure 2: This demonstrates the problem with using classification strength as an a priori predictor of playing quality. There are ten available moves, eight wins and two losses. If both evaluators assume that values above .5 are wins and those below .5 are losses, function A classifies 90% correctly, and makes the wrong move. Function B, on the other hand, only gets 60% of the nodes right, but makes an optimal move, anyway.

arise from a realistic set of circumstances quite similar to those that cause a well known game-tree phenomenon, the horizon effect (the tendency to push foregone conclusions beyond the search frontier) [Ber73]. Errors of the type made by function A frequently arise from the evaluation of board positions that occur in the middle of a combination of moves or a series of exchanges. These nodes are not *quiescent*, and should not be evaluated statically. Evaluations that are made on non-quiescent nodes are highly unreliable because the static information is likely to change radically as soon as the horizon is extended [Sha50]. The classic non-quiescent position, for example, appears halfway through a queen trade in chess. Although the trade's completion may lead directly to a LOSS, mid-trade nodes tend to indicate very strong positions, in which a queen advantage suggests a WIN. As soon as play continues and the horizon is extended, however, the trade's completion will become evident, as will the error in static evaluation. Unfortunately, non-quiescent nodes are not always easily recognizable, and are thus frequently unavoidable. The problem of quiescence has been characterized as a necessary outcome of the standard definition of the evaluator. Since the evaluator is designed to estimate the value of a node as it relates to the goal, there is no logical point for terminating search other than reaching a goal node (leaf) [Bot84] [Ber73] [Ber79]. Thus, the search frontier must be set arbitrarily, and anomalies of the horizon abound.

Of course, nobody ever said that evaluation functions had to be comparable a priori; a posteriori operational comparisons are also possible. Defining an evaluator's strength as proportional to the strength of its play allows any two heuristic functions to be compared via head-to-head competition. There are, however, two major drawbacks to this approach. First, comparative studies fail to provide an absolute measure of heuristic function quality, and second, evaluator strength is not the sole determining factor of a program's decision quality and performance level.

3.3 Control Strategies

The performance of a game program is dependent on several components of its search system, including the static evaluator, backup algorithm, and search depth. Decisions are based on the recommendations of a control strategy, which specifies both how the static values assigned across the search frontier should be combined, and which move should ultimately be made. Complete minimax, for example, is the optimal control strategy on game-trees that can be searched exhaustively. When complete minimax is inapplicable, however, an alternative strategy is necessary. Unlike evaluation functions, which in the past have been addressed in a strictly domain-specific manner, much of the work done on control strategies has been domain-independent.

In his original analysis of chess programming, Shannon described two families of control strategies, type-A and type-B. A type-A strategy behaves exactly as if the tip nodes are leaves, and applies *partial 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 heuristic 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 complete minimax, which does. The assumption underlying the use of partial minimax is that the estimates are accurate; the success of the strategy depends on the validity of this assumption. This "face-value principle" of perfect estimates is generally taken for granted, not because it is believed to be true, but rather because no stronger assumptions are available [Pea84]. Type-B strategies, which do not perform full-width searches, but rather consider only reasonable moves, similarly rely on the face-value principle; in these systems, not only are heuristic *estimates* assumed to be accurate as static values, but *guidelines* of indeterminate value are accepted as valid criteria for deciding which lines of play are reasonable, as well.

In 1975, Monroe 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 com-

puter hardware, software, and programming efforts, rather than fundamental breakthroughs in how to program computers to play better" [New75]. For the most part, this remains true in 1987. Nearly all control strategies contain an element of Shannon's analysis; 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. There is, however, one highly significant (now standard) technique that was developed after Shannon's work, namely α-β pruning. The basis of this algorithm lies in the observation that there is an easily recognizable class of moves that will obviously not be selected by minimax. The exact origins of α-β are disputed, but the first paper to discuss it in detail was probably [EH63]. The α-β algorithm (see figure 3) prunes by recording boundaries within which the minimax value of a node may fall. The parameter α represents 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 must fall outside the range are pruned, and their subtrees can be ignored.

The α-β algorithm always chooses the same move as a minimax search to the same depth. Thus, when the tree is complete, the choice is optimal. In a partial tree, however, it is the face-value principle that justifies pruning. Since partial minimax implicitly assumes that the estimated tip values are accurate.

they can be pruned as easily as exact values. The move made, then, although under no promise of optimality, is guaranteed to remain the same. To insure that the minimax choice is not missed, α and β start at minus and plus infinity, respectively, and are updated as the tree is traversed.

To date, the efficiency of pruning algorithms has been the most frequently analyzed aspect of two player search systems. The sensitivity of α-β to the order in which nodes are examined was first pointed out in [SD69]. The algorithm's behavior under several different orders was analyzed in [FGG73] [KM75], where it was shown that in the average case, α-β cuts the effective branching factor from b to approximately $b^{3/4}$, and allows the search depth to be extended by 33%. The asymptotic optimality of α-β over the class of all game searching algorithms, in terms of the average branching factor, 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 be comparable to (and occasionally better than) α-β, [CM83] [RP83]. Although SSS*, on the average, expands between one-third and one-half of the nodes that α-β does, the speedup is not sufficient to offset the additional bookkeeping and storage costs incurred. SCOUT, on the other hand, not only fails to offer much in the way of a speedup over α-β, but is also less familiar and harder to implement. For these reasons, neither SSS* nor SCOUT has ever been used in a successful

performance oriented game program[1].

The primary importance of these pruning algorithms is that they increase the efficiency of partial minimax, allow larger portions of the tree to be searched, and as a result, make more accurate decisions. Although α-β-partial-minimax is far and away the most popular control strategy, it is by no means the only one. I have presented a survey of game-playing strategies in [Abr86a]. One common thread running through all previously proposed search strategies is their (at least partial) reliance on the face-value principle. Given this universal dependence on accurate estimates, the importance of determining a precise, useful aim for the evaluator should be obvious. Nevertheless, no such definition exists. The use of partial minimax to back up estimated information, for example, has never been justified analytically; it was originally used because it would be optimal if the information were accurate, and retained solely on the strength of empirical evidence. In fact, the few attempts at analytic justification that have been made have discovered that for an infinite class of game-trees, partial minimax propagates errors, rather than filtering them out [Nau83a] [Pea83] [Bea80]. According to these studies, deeper searches should lead to poorer play, thereby contradicting the algorithm's observable performance.

[1] An iterative algorithm based on SSS* has recently been developed [BB86]. This technique, which involves trading off space for time, retains a good deal of SSS*'s speedup without exceeding the available memory.

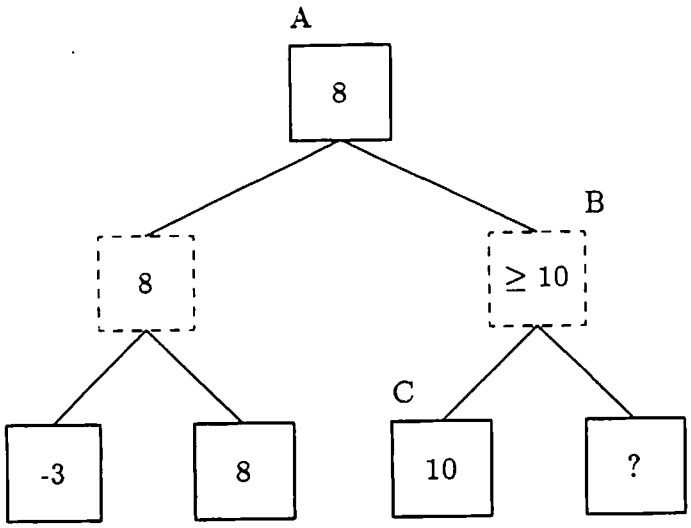


Figure 3: The α-β pruning algorithm is based on the observation that some nodes will obviously not be selected by minimax. Once a node has been removed from consideration due to the evaluation of one of its descendants, no further paths beneath it need be evaluated. For this reason, no evaluations were made on the node containing a ?. To see α-β in action, look at nodes A, B, and C. C is a leaf, statically evaluated at 10. This means that B (a MAX node) must have a value no less than 10. Since A (a MIN node) already has a child valued at 8, B will not be chosen, and its remaining child may be pruned.

This rift between predictions and observations calls the propriety of partial minimax into question. One of the algorithm's most obvious drawbacks is its immediate adoption of the single best child of each parent. If that one crucial estimate is wrong, the justification underlying the strategy's use is unfounded. It has been fairly well established that given the same search depth, strategies that account for multiple children make stronger decisions than minimax. Studies done on the M&N algorithm demonstrated that adding "some (experimentally determined) function of the M maximum or N minimum values" to the minimax value increases the accuracy of the decisions made [SD70]. This algorithm is not widely used, however, because it allows many fewer α and β cutoffs than minimax. The efficiency of α-β-partial-minimax allows deeper searches than M&N, which (apparently) more than compensates for the reduced accuracy. Nevertheless, it is possible that if the appropriate "experimental" addition could be determined analytically, a stronger pruning companion might lead to deeper M&N searches and even more accurate decisions. In addition, the discovery of *minimax pathology* [Nau83a] has led to the investigation of *product propagation* [Pea81], a control strategy that backs up the product of the values of a node's children. Despite the clearly inaccurate assumption of independence among sibling nodes required to justify this type of strategy, some surprising experiments have been run in which product propagation outplayed partial minimax [NPT83] [CN86] [CN87]. This

result alone should be enough to indicate that partial minimax is not always the strongest possible strategy, and highlight the importance of understanding static evaluation before devising backup strategies.

3.4 Summary

The study of game-tree search has been of continuous interest to artificial intelligence researchers. Early work on type-A strategies led to the partial minimax and α-β pruning algorithms for backing up node values, while quiescence analysis and type-B strategies considered the horizon effect and the need for intelligently set search frontiers. Despite the tremendous effort that has gone into understanding game-trees, the face-value assumption of accurate estimates has been universally accepted without a thorough investigation of its implications. Ironically, it may well be that this very principle is the source of most open problems in game-tree search; because of the ambiguity of static evaluators, functions can't be compared, the justification of backup algorithms relies heavily on the accuracy of estimated tip values, and search frontiers fall prey to the horizon effect. In addition, Newborn's 1975 observation that all successful chess programs were based on ideas developed in 1950 remains essentially true in 1987. This points out the importance of reassessing some of Shannon's original assumptions; in 1950, human labor was cheap, computation expensive, and probabilistic algorithms

[Rab76] undefined. Nearly four decades of technological advances should make new mathematical models — which may address some of these long-standing open problems — reasonable, and worthy of investigation.

Part II

The Model

4 Proposal: What is the Basic Model?

In a broad sense, the purpose of an evaluation function in a two-player domain is to indicate whether a given node on the search frontier will result in a victory. The standard assumption, forwarded by proponents of approximating minimax values, has been that this corresponds to an estimate of the outcome that would be arrived at by perfect play, even though it is universally acknowledged that play will not, in fact, be perfect. The implicit justification of the perfect-play assumption is that it is useful as a defense mechanism. After all, defense against a perfect opponent should work against an imperfect one as well. Nevertheless, this approach has some significant drawbacks: it tends to make defense against imperfect play considerably more difficult than necessary [Bra80], it assigns the evaluator a task that does not allow efficient estimation, and it fails to recognize the possibility of escaping from a forced LOSS position by later capitalizing on an opponent's error. This is not surprising. Perfection, as a deterministic all-or-nothing goal, frequently depends on omniscience — the concept of near-

perfection is not well defined, and the performance of algorithms that require complete information but are given only partial data is unpredictable. Thus, alternatives to the perfect-play assumption should be considered.

At the opposite end of the spectrum from perfection stands randomness. The interpretation of an evaluator based on the random-play assumption is straightforward. The value of a game is known only when play is over. Thus, only leaves should have exact values; internal nodes can be characterized as probability distributions with values that will be instantiated by the leaf eventually reached. Since the single most important parameter of any distribution is its mean, the values associated with an internal node should be the expected value of the leaves beneath it (if necessary, any set of non-numeric outcomes, such as {WIN,LOSS,DRAW}, can easily be quantified). This is precisely the outcome of the game under the random-play assumption, and gives rise to the *expected-outcome* model.

Definition: *Expected-Outcome Values*

The expected-outcome value of a game-tree node, G, is given by a player's expected payoff over an infinite number of random completions of a game beginning at G, or

$$EO(G) = \sum_{leaf=1}^{k} V_{leaf} P_{leaf},$$

where k is the number of leaves in the subtree, V_{leaf} is a leaf's value, and P_{leaf} is

the probability that it will be reached, given random play. It is important to note that P_{leaf} is not necessarily equal to $\frac{1}{k}$. The probability that a leaf will be reached is one over the product of its ancestors' branching factors; a node with no siblings is twice as likely to be reached as a node with one sibling. For example, consider a node N with two children, A and B, and three leaf grandchildren, one LOSS from A and two WINs from B, as shown in figure 4. Although the probability that a randomly selected leaf will be a WIN is two-thirds, the probability that random play from N will end in a WIN is one-half, and thus N's expected-outcome value is one-half. Leaves are only equiprobable if all nodes of equal depth are constrained to have identical branching factors.

While the assumption of random play may seem unrealistic, evaluation functions in two-player games are normally applied only at the frontier of the search. By definition, the frontier is the limit beyond which the program cannot gather any further data about the game-tree, and in the absence of such information, random play is the only practical assumption. While this approach stands in stark contrast to the usual one, its utility to specific domains is primarily an empirical question. Setting the issue of plausibility aside for a moment, the expected-outcome model has a number of attractive features. First, it is precise. Second, it allows heuristic quality to be measured in terms of proximity to this ideal; an estimated mean is a well-defined statistical quantity, whereas

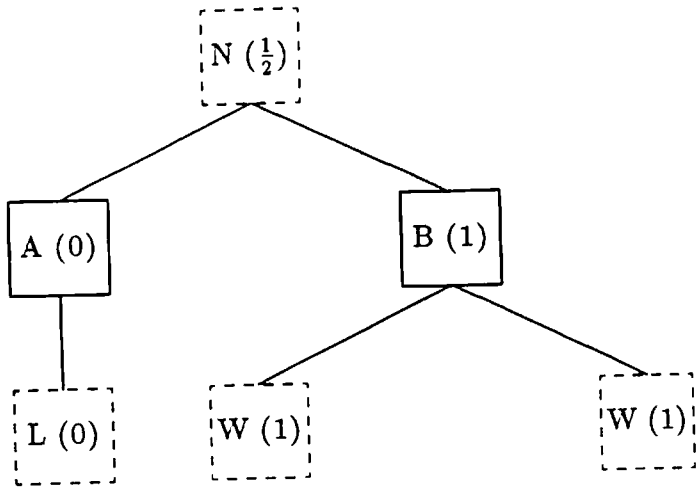


Figure 4: A node's expected-outcome value is defined as the expected payoff over an infinite number of random games that begin from it. In this picture, the three leaves are shown with their exact values of 0 for LOSS and 1 for WIN. Nodes A, B, and N are then assigned their expected-outcome values: all completions from A pay 0, and all completions from B pay 1. Since half of the games begun at N will pass through A and half through B, N is valued at 1/2.

an approximate minimax value is not. (Note that this also provides a method for comparing any two evaluators). Finally, and most significantly, it provides a practical means of devising heuristic functions — expected values can be approximated by random sampling. Along with their many advantages, of course, expected values (and other statistical parameters) do bear a serious onus: they can be very misleading when population sizes are relatively small. Thus, care must be taken not to rely too heavily on expected-outcome values in end-game play. In the end-game, however, exhaustive searches are frequently possible and special techniques are usually adopted anyway.

5 Support: Does the Model Work?

The expected-outcome model tends to elicit two immediate reactions, one positive, the other negative. The model has a very strong appeal because it is elegant, crisply defined, easily estimable, and above all, domain independent — its reliance on nothing more than a game's rules and outcomes makes it equally applicable to all two-player games. Its use of the random-play assumption, on the other hand, is at least mildly distressing; in addition to appearing even less accurate than the perfect-play assumption, it sacrifices the defensive edge gained by assuming a perfect opponent. Substantial supporting evidence will be necessary to dispel the apprehension caused by this approach. Like most heuristic

techniques, expected-outcome's performance in a specific game can only be determined empirically. Due to its claim of domain-independence, however, analytic support for the model should be derivable from generic game-trees.

There is a simple class of game-trees that has been analyzed fairly extensively, those with uniform branching factors and depths, and leaf values restricted to {WIN,LOSS}. It was on this class of trees that the minimax convergence [Pea80] and last player [Nau82b] theorems were derived, and minimax pathology has been studied in the context of one particular subclass [Nau82a] [Nau83b] [Nau83a] [Pea83] [Pea84] [Abr86b]. In another subclass, to be described more fully in section 5.1, ideal expected-outcome functions are provably (with probability approaching one) optimal. In addition, if the trees are a bit more complex, but some fairly reasonable conditions are met, expected-outcome retains its optimality. Real games, of course, generate trees that are too irregular to be treated analytically, and thus the utility of expected-outcome functions must be demonstrated experimentally. Although it is clearly impossible to verify the full extent of a model's applicability on a game-by-game basis, testing it on a variety of popular games should validate the claim that the utility of expected-outcome functions is not restricted to a single domain. An effective investigation of the expected-outcome model, then, must yield answers to four questions: When are the decisions it recommends optimal? If non-optimal, does the decision qual-

ity remain high? If exact information is unavailable, can an estimated function perform well? and Is it possible to implement the model efficiently? If all of these questions can be answered in the affirmative, the expected-outcome model will fulfill the purpose for which it was designed: a precise, useful, and domain-independent model for two-player static evaluators.

Before proceeding with the investigation, it is interesting to note that most previous research in computer game-playing has been characterized by a division between analysis and empiricism; it has been fairly rare for a technique motivated by the study of simple models to be immediately applied to real games. For the most part, analyses have been performed on simplified models. Quantitative conclusions have depended on the models studied, and few qualitative leaps to "real" games have been made. Actual programs, on the other hand, have implemented many heuristics that have never been analyzed. Although the *efficiency* of α-β and other backup algorithms has been studied [KM75] [RP83] [Pea84] [FGG73] [SD69], that of most forward pruning techniques used by type-B strategies has not been, and discussions of heuristic *accuracy* are virtually nonexistent. The different emphases of analysts and empiricists have effectively split the study of game-trees between two fields, heuristic analysis and game programming. Since expected-outcome can be discussed in both frameworks, it should be of interest to programmers and analysts alike. The model's simultaneous treatment by two

fields, however, may obscure the need for both the theorems of section 5.1 and the experiments of section 5.2. These sections demonstrate the utility of parameter-based evaluators (that is, functions that study the parameters of leaf-value distributions) in two different contexts: section 5.1 demonstrates that on a large subclass of a well-established analytic model, conclusions based on the perfect-play and random-play assumptions are identical. Thus, the expected-outcome model should be of theoretical interest to heuristic analysts. Section 5.2, on the other hand, culminates in the demonstration of a viable machine-generated static evaluator, that requires a minimum of expert input. This indicates that the model should be of interest to game programmers as well. In addition, by allowing a qualitative leap to be made between theory and practice, the introduction of "probabilistic game-trees" may suggest many areas in which further feedback between them is possible.

5.1 Analytical Evidence

Historically, the study of computer game playing has been an empirical field. Primarily due to the complexity of game-tree models, analyses have been few and far between. Games like chess were chosen as abstractions of the real world that were simple enough to allow computer simulation and experimentation. They remained, however, way too complex to allow mathematical analysis. Thus, a

new model had to be defined which was an abstraction of the game-tree. This section shows that on an infinite class of analyzable game-trees, the perfect-play and random-play assumptions lead to identical moves: section 5.1.1 discusses some of the properties of a popular analytic model called the (d, b, f)-tree, while sections 5.2.2 and 5.2.3 show how these properties imply the equivalence of the two assumptions.

5.1.1 An Analytic Game-Tree Model

The most frequently analyzed game-tree models to date have all been variations of the *(d,b,f)-tree* [Pea80] [Pea84], which are game-trees with uniform leaf depth d, uniform branching factor b, and leaf values assigned by a set of independent identically distributed random variables drawn from a common distribution function (p.d.f.), f (with the corresponding cumulative distribution function, or c.d.f., represented by F).

Although (d, b, f)-trees do not correspond to the trees generated by chess or other popular games, there is a class of games, called board splitting, that was designed to fit perfectly into the (d, b, f) paradigm [Pea84]. In these games, a square b^d-by-b^d board is covered with randomly distributed H's and V's, (where the distribution of H's and V's is described by f). 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, only one square remains. If that square contains an H, the horizontal splitter wins. Otherwise, the vertical splitter wins.

Many interesting results have been derived on (d, b, f)-trees and board splitting games, but of particular relevance to expected-outcome are the minimax convergence [Pea80] and last player [Nau82b] theorems. According to these theorems, the minimax value of the root of a (d, b, f)-tree is essentially (with probability approaching one) determined as soon as the parameters of the tree are set. In a tree with leaf values restricted to WIN and LOSS, for example, there is a threshold value, denoted by Ξ_b, such that if WIN leaves were generated with probability greater than Ξ_b, the tree's minimax value is almost certain to be a WIN. If WINs were generated with probability less than Ξ_b, on the other hand, the root value will be a LOSS. A general statement of the theorem is given below without proof. For a proof, see [Pea80] [Nau82b] [Pea84].

Theorem 1: *Minimax Convergence* [Pea84]

The root value of a (d, b, f)-tree with continuous terminal c.d.f. F converges (in probability), as $d \rightarrow \infty$ to the $(1 - \xi_b)$-quantile of F, where ξ_b is the solution in $(0.0, 1.0)$ to $x^b + x - 1 = 0$.

(Recall that the Q^{th}-quantile of a distribution is defined as the value y, s.t. Pr[A leaf has a value $\leq y$] $= Q$ and Pr[A leaf has a value $> y$] $= 1 - Q$).

It is important to note that as stated, the minimax convergence theorem is incomplete. To begin with, if leaf values are restricted to WIN (with probability p) and LOSS (probability $(1-p)$), the convergence is to 0 if $\xi_b > p$ and 1 if $\xi_b < p$. This, of course, is nothing more than a special case of the general theorem. In a binary distribution, the value at all quantiles $(1-Q) < p$ is 0, and otherwise, it is 1. Of greater significance, however, is that according to the *last player theorem* [Nau82b], these values are only correct if MAX has the last move (uniform leaf depth guarantess that the last move will always belong to the same player). When the "last player" is MIN, the minimaxed root values converge to a different set of quantiles, specified by replacing ξ_b with $(1-\xi_b)$. The complete theorem, then, would have to specify four sets of equations to account for minor discrepancies caused by restricting leaves to binary values vs. allowing them to cover an arbitrary range, and by shifting the last move from MAX to MIN. If a general threshold, Ξ_b, is defined by

$$\Xi_b = \begin{cases} 1-\xi_b & \text{if MAX is the last player} \\ \xi_b & \text{if MIN is the last player,} \end{cases}$$

however, it is possible to speak in terms of values x that fall either above or below the relevant Ξ_b-quantile of the c.d.f.[5]

In a nutshell, what these theorems say is that the root value of a (d,b,f)-tree

[5]Note that this means that binary distributions converge to 1 if $p > (1-\Xi_b)$ and to 0 if $p < (1-\Xi_b)$.

is dependent only on the parameters of the tree, not the actual arrangement of its leaves. The tree's depth is significant in two respects: it determines the last player and the rate of convergence. The branching factor and last player determine the quantile to which convergence will occur, and the leaf distribution function provides the value at that quantile. This convergence of minimax values is not terribly surprising. One of the general principles underlying the study of statistics is that as population sizes tend to infinity, behavior patterns become increasingly predictable. Minimax convergence simply states that game-trees are no exception; as they grow, their "personalities," or root values, approach a predestined norm.

Consider, for example, the case of binary valued leaves[6]. If the distribution of level-d nodes (leaves) is described by

$$f = \begin{cases} WIN(\text{for}MAX) & \text{with probability } p \\ LOSS(\text{for}MAX) & \text{with probability } 1-p, \end{cases}$$

then the distribution of root minimax values can be defined as

$\mathcal{F}_d(p) \stackrel{def}{=}$ Pr[Root (level-0) is a WIN for MAX, given p and d].

When d is small, $\mathcal{F}_d(p)$ is S-shaped — the S's left tail indicates values of p that are certain losses, its right tail certain wins, and its central diagonal encompasses

[6]In cases where f takes on an arbitrary set of leaf values, everything can be defined analogously: for all real leaf values x, $F(x) \stackrel{def}{=} \int_{-\infty}^{x} f(t)dt$, and
$\mathcal{F}_d(x) \stackrel{def}{=}$ Pr[A node d ply above leaves has a minimax value $\leq x$].
Clearly, this subsumes the binary case. To simplify the discussion, binomial distributions have generally been considered.

all values of p that may lead to either. As $d \to \infty$, however, $\mathcal{F}_d(p)$ approaches a step function, and the range of p values that may lead to either wins or losses shrinks to a point. In other words, if the value at the Ξ_b-quantile of F is 1, $\mathcal{F}_d(p)$ will converge to 1, if it is 0, convergence will be to 0. The only instance in which the value of the root may be either 0 or 1, then, is when the value at the appropriate quantile is in some neighborhood around Ξ_b. On an S-curve, this may involve a significant portion of the curve. In a step function, however, the "neighborhood" is a point, and unless p is set at *exactly* that point, it will either be above or below the relevant threshold. In practice, d does not have to be very large for the step functions to appear, because the rate of convergence is super-exponential in d [Pea84]. Furthermore, If d is not large enough for convergence to occur, the tree is probably small enough to permit exhaustive search, and the size of the leaf population is too small for statistical parameters to give useful information to decision makers, anyway.

The convergence of minimax values poses an interesting problem for researchers who wish to study issues related to decision quality on (d, b, f)-trees. Since a tree's value is strictly determined by its parameters for nearly all f, no errors can be made by any evaluator; with probability approaching one as the trees approach infinite depth, any two (d, b, f)-trees with identical b, d, and f will either both be forced wins or both be forced losses, and thus in the limiting case,

even a random function will choose "optimally" among a set of (d, b, f)-trees with identical parameters. In order to study issues related to decision quality on (d, b, f)-trees, then, some mechanism for introducing the possibility of error must be adopted. One solution, used in the study of minimax pathology, was to define a class of "fair" board splitting games [Pea84] [Nau82a]. Since board splitting games were designed as physical embodiments of (d, b, f)-trees [Pea84], the outcome of a game is preordained as soon as the probability p of generating a V is set; after all, the first move amounts to choosing among a set of b $(d-1, b, f)$-trees with identical parameters, or no choice at all. The game can be made fair, however, if $p = \xi_b$. Only in this case will the game's outcome remain undetermined — either player may win with a non-zero probability [Nau82a].

Although the class of fair board splitting games has served quite nicely as an illustration of issues related to minimax pathology [Nau82a] [Nau83b] [Nau83a] [Pea83] [Pea84] [Abr86b], it is not particularly useful in the context of expected-outcome. At any point in a board splitting game, the potential moves are all (d, b, f)-trees generated with identical d, b, and p. If the game is unfair, in the limiting case any evaluator will play optimally. If the game is fair, however, there will be some legitimate choices to be made between forced wins and forced losses. In these cases, the performance of expected-outcome should be suspect for two reasons. First, if the trees are even reasonably large, (say depth eight in

a binary tree, or 2^8 leaves), all of the expected-outcome values will fall in a very small range, and thus not offer much in the way of discriminating power. Second, because the game was made fair, the moves were all drawn from the small group of trees whose minimax values are not determined by their parameters. Preliminary experiments run on binary trees indicate that in this particular case, not only does expected-outcome fail to perform well, but as the trees grow, it performance deteriorates. This is not completely unexpected — evaluation functions that choose among game-trees based on their respective leaf-value distributions can not discriminate effectively among trees designed with identical parameters because the same value will be assigned to each option. As the trees grow, the discriminating power of expected-outcome diminishes to the point where all decisions are made by a (secondary) tie-breaking rule. This difficulty suggests that fair board splitting games are not a useful vehicle for introducing the possibility of error in the context of the decision quality of expected-outcome, and motivates the need for a somewhat different approach.

5.1.2 Decisions Based on Leaf Distributions

The issue at hand, of course, is not the convergence of root values, but rather the decision quality of expected-outcome evaluators. The relationship between these issues, however, is rather straightforward: in (d, b, f)-trees, minimax values

are determined by the parameters used to *design* the trees and the *observed* expected-outcome values indicate precisely what these parameters are. This suggests investigating whether given the choice of b moves, $m_1, m_2, ...m_b$, representing the roots of b (d, b, f)-trees with identical depths and branching factors but different leaf distributions, expected-outcome will select the best move. The resultant model for studying the decision quality of expected-outcome is a tree constructed by generating b (d, b, f)-trees with identical b's and d's, but different f's, and linking them together to a single parent. The lone decison in such a tree is which subtree should be chosen. Although this may seem like a trivial test, it does have the desired effect of a (d, b, f)-tree model in which it is both possible to distinguish among subtree parameters and make real choices. As a simple corollary to the minimax convergence theorem, knowledge of the relevant parameters will lead to perfect decisions. Since the significance of (d, b, f)-trees lies in the implications that they have to real games, any techniques derived in the context of simple, analytically tractable models should serve primarily as motivation for studying their implementations in progressively more realistic domains to determine how much of their power remains as the simplifications are dropped. With this in mind, then, the correspondence between assuming randomness and perfection on (d, b, f)-trees is enough to indicate that expected-outcome and related distribution-based functions are worth studying even in domains in which

convergence can not be expected to occur.

On (d, b, f)-trees, however, convergence does occur, and determining the decision quality of expected-outcome is quite simple — it is optimal wherever root values converge. The convergence theorems indicate that in the limiting case, a tree's minimax value is completely determined by the parameters that went into designing it, namely d, b, and f. Expected-outcome, as a statistical measure of distribution means, is governed by the law of large numbers, which states that as the size of a population grows, the observed mean approaches the distribution's actual mean at a rate linear in the population size [Fel57]. In the case of (d, b, f)-trees, this means that as the trees get deeper, expected-outcome values (observed means) become increasingly accurate measures of f's theoretical mean. Since the size of the leaf population is given by b^d, the rate of increased accuracy is exponential in d. Thus, as $d \rightarrow \infty$, knowledge of a move's expected-outcome value will be sufficient to determine its minimax value.

Proving the optimality of distribution-based-evaluators given converged root values, then, simply involves combining the law of large numbers with minimax convergence: as larger leaf populations are observed, the empirical estimations of the distribution's parameters approach their true values. Since the tree's minimax value is strictly determined by the distribution used in its design, f, the observations also become progressively more accurate predictors of the root

values. Thus, as the trees grow, knowledge of the outcome given random play (assuming binary valued leaves) accurately predicts the outcome given perfect play. With this information, not only can optimal moves always be made, but a perfect correspondence between optimality given perfection and given randomness becomes evident.

Binary Valued Leaves The utility of expected-outcome functions to (d, b, f)-trees whose outcomes are limited to WIN and LOSS should be immediately obvious. The probability that random play on a (d, b, f)-tree with binomially distributed leaves will lead to a win is defined as the probability that a random path from the root to the leaves will terminate at a WIN leaf. In a uniform tree, this is exactly the probability that a randomly chosen leaf will be a WIN, or p. Since the mean of a binomial distribution is given by the parameter p, ideal expected-outcome functions calculate the de facto values of $\hat{p}$ for each of the b moves, and select the best one. Without loss of generality, order the moves such that $\hat{p}_1 \geq \hat{p}_2 \geq \cdots \geq \hat{p}_b$, (where $\hat{p}_i$ is the expected-outcome value of move m_i). If the players both based their decisions on expected-outcome values, MAX would choose m_1 and MIN m_b. By the law of large numbers, as the trees grow, the expected-outcome values $\hat{p}_i$ will approach the corresponding theoretical parameters, p_i, that were used in generating the trees. In other words, $\forall i \ \lim_{d \to \infty} \hat{p}_i = p_i$. Assuming that none of the p_i are set in the transition region

(that is, all root values converge), then, there are only two possibilities (for MAX, say):

(a) If $\hat{p}_1 > (1 - \Xi_b)$, then $lim_{d\to\infty}\mathcal{F}_d(p_1) = 1$. This means that m_1 is a WIN, and thus optimal for MAX.

(b) If $\hat{p}_1 < (1 - \Xi_b)$, then as $d \to \infty, \mathcal{F}_d(p_1) = \mathcal{F}_d(p_2) = \cdots = \mathcal{F}_d(p_b) = 0$. Since all of the moves are LOSSes, they are all equally "optimal".

This leads to the following corollary to the convergence theorems:

Corollary 1: *Optimality of Binary Expected-Outcome*

Let $m_1, m_2, ..., m_b$ be (d, b, f)-trees with identical branching factors and depths, and binomially distributed leaf values with expected-outcome values of $\hat{p}_1 \geq \hat{p}_2 \geq \cdots \geq \hat{p}_b$, respectively, where all roots converge. Then,

$lim_{d\to\infty}$ (Pr[m_1 is optimal for MAX] = Pr[m_b is optimal for MIN]) = 1.

Arbitrary Leaf Values One of the benefits of performing analyses on trees with leaf values restricted to WIN and LOSS is that given a move's true value, it is fairly simple to determine whether or not it is optimal. Under any set of circumstances a move whose value is WIN is optimal, and, unless there are no WINs available, one labelled LOSS is not. When leaves are allowed to take on an arbitrary set of values, however, any value may be optimal at any time. Thus, even knowledge of a move's true minimax value is insufficient to determine its

optimality; the values of all of its siblings must be known as well. An ideal minimax evaluator, then, would rank the moves according to their minimax values, $M_1, M_2, ..., M_b$, and (correctly) recommend M_1 to MAX and M_b to MIN. How closely does the ideal expected-outcome ranking of observed distribution means, $\hat{\mu}_1, \hat{\mu}_2, ..., \hat{\mu}_b$, correspond to the optimal one? For the moment, consider one of expected-outcome's relatives, *Q-outcome*. Both functions belong to a family of evaluators that assigns each node a value corresponding to some statistical parameter of its leaf distribution. In the case of expected-outcome, the parameter is μ, with Q-outcome it's the Q^{th}-quantile[7]. Thus, Q-outcome ranks the moves $\hat{q}_1, \hat{q}_2, ..., \hat{q}_b$, in order of their Q-quantiles, and recommends m_1 to MAX and m_b to MIN. Once again, $\forall i \lim_{d\to\infty} \hat{q}_i = q_i$. The same argument that was used to demonstrate the optimality of binary expected-outcome may be used to validate Q-outcome, where $Q = \Xi_b$.

For any real value x, there are three significant ranges in which $F(x)$ can fall:

(a) If $F(x) = \Xi_b$, then by definition, x lies on the Ξ_b-quantile of F. Call this point x^*. Note that x_i^* is the value assigned to q_i by Ξ_b-outcome.

(b) If $F(x) > \Xi_b$, $lim_{d\to\infty}\mathcal{F}_d(x) = 0$. Any $x > x^*$, then, is essentially guaranteed to exceed the root value.

[7]The mean, μ, does not necessarily correspond to any particular quantile. One notable exception, to be considered in section 5.1.3, is the family of normal curves, in which the mean is the median, thereby equating expected-outcome and .5-outcome.

(c) If $F(x) < \Xi_b$, then $lim_{d\to\infty}\mathcal{F}_d(x) = 1$. In other words, any x that falls on a quantile smaller than Ξ_b is almost certain to be less than or equal to the root value. Thus, any $x < x^*$ gives a lower bound on the root value, and the closer x is to x^*, the tighter the bound.

This leads to a second simple convergence corollary:

Corollary 2: *Optimality of Q-Outcome*

Let $m_1, m_2, ..., m_b$ be (d, b, f)-trees with identical branching factors and depths, and leaf distributions specified by p.d.f's $f_1(x), f_2(x), \cdots, f_b(x)$, respectively.

Order the moves by (observed) Ξ_b-outcome. Then

$lim_{d\to\infty}$ (Pr[m_1 is optimal for MAX] = Pr[m_b is optimal for MIN]) = 1.

5.1.3 Non-uniform Game-tree Models

The optimality of leaf-distribution-based evaluators suggest simple, optimal strategies for all games played on (d, b, f)-trees: if f describes a binary random variable, choose the first move for which $\hat{p} > (1-\Xi_b)$. Otherwise, rank the moves according to Ξ_b-outcome and select the best one. Unfortunately, (d, b, f)-trees are highly unrealistic models, and almost never correspond to real world (or even real game) situations. Several complications arise when the models being studied move from the simple to the complex. First, exact statistical parameters are as difficult to calculate as complete minimax values. Their advantage lies in being easier to

estimate, and with estimation comes error. In the binary case, these errors mean that a node's likelihood of being classified correctly is directly proportional to its distance from the threshold; although two moves with theoretical parameters of p_1 and p_2, $p_1 > p_2 > (1 - \Xi_b)$ may be equally good options, if the values are estimates, $\hat{p}_2$ is less likely to be classified correctly than $\hat{p}_1$, and is thus a weaker move. This justifies expending the effort necessary to evaluate all nodes, rather than just selecting the first one that looks good.

A second, and more significant, difficulty introduced by complex models arises from the absence of uniform branching factors and depths. The uniformity of b is a necessary factor in the definition of Ξ_b, and even if b is constant, an unknown depth and unidentified last player make it impossible to know whether nodes should be ranked according to $(1 - \xi_b)$-outcome or ξ_b-outcome. Without this information, there is no way for a player using a Q-outcome function to know which quantile of the leaf-distribution to study. These difficulties strongly imply that the optimal (d, b, f)-tree strategies are not widely applicable. Therefore, some modifications must be made to insure that play will be at least reasonable. Perhaps the most obvious idea is to set $Q = .5$; the median is equidistant from the two candidate quantiles, and likely to give useful, albeit nonoptimal information, regardless of the value of Ξ_b. The use of the median to evaluate nodes, in turn, suggests considering other central tendencies of leaf distributions, in particular,

their means. Unlike medians, means are not restricted to actual leaf values, and thus have significantly greater discriminating power. This strength is particularly important when the number of possible leaf values is small.

This defense of expected-outcome on (d, b, f)-trees with non-binary leaf values is rather weak. It is possible, however, to show that under a fairly realistic set of circumstances, the ordering imposed on a set of moves by expected-outcome is identical to that imposed by Ξ_b-outcome. Consider the case of two normally distributed random variables, X_1 and X_2, that correspond to leaf-value distributions for two moves in a game-tree, m_1 and m_2, with expected-outcome values defined by $\hat{\mu}_1 > \hat{\mu}_2$, respectively. In order to discuss the relationship between the evaluators, it is important to recall that quantiles are defined in terms of area under a distribution curve. On normal curves, there is a measure related to the c.d.f. known as the *z-score*. Given an area under a normal curve, a quantile function returns the x-intercept of the area's upper bound. The z-score returns the same value, but in units of standard deviations from the mean. Thus, if a point $x > \mu$ is described by both quantile Q and z-score z, then $F(x) = \mu + z\sigma$, and the area under the curve from $-\infty$ to $(\mu + z\sigma)$ is Q.

The general formula for calculating the z-score of a point x on a normal curve with mean μ and standard deviation σ is:

$$Z(x) = \frac{(x - \mu)}{\sigma}.$$

The function $Z(x)$ is uniquely invertible; for any z-score, the value of

$$x = Z^{-1}(z) = z\sigma + \mu$$

can be calculated. A quantile-based evaluation function inputs its relevant parameter z_Q into the two Z^{-1} functions, (one for each distribution being considered), receives as output the values $x_1 = Z_1^{-1}(z_Q)$ and $x_2 = Z_2^{-1}(z_Q)$, and chooses the best move.

Lemma 1:

If two evaluators, Q_a-outcome and Q_c-outcome, recommend different moves, then there is some z-score y, $z_{Q_a} < y < z_{Q_c}$, for which the two leaf distributions share a common x-intercept.

Proof:

If a function inputting z_{Q_a} selects m_1, then the value at the Q_a-quantile of the m_1 curve exceeds the the one at Q_a-quantile of the m_2 curve, or $Z_1^{-1}(z_{Q_a}) > Z_2^{-1}(z_{Q_a})$. If, at the same time, an evaluator relying on z_{Q_c} opts for m_2, then $Z_2^{-1}(z_{Q_c}) > Z_1^{-1}(z_{Q_c})$.

By a trivial corollary of the intermediate value theorem, Z_1^{-1} and Z_2^{-1} must cross at some point y, $z_{Q_a} < y < z_{Q_c}$. Thus, $Z_1^{-1}(y) = Z_2^{-1}(y)$, and the two c.d.f.'s share a common x-intercept. □

By the definition of Z^{-1}, this means that $y\sigma_1 + \mu_1 = y\sigma_2 + \mu_2$. Solving for y

yields

$$y = \frac{\mu_1 - \mu_2}{\sigma_2 - \sigma_1}.$$

Since y is the point at which the Z functions cross, any two evaluators that consider z-scores on the same side of y will reach the same conclusion; any two that examine opposites sides of y will disagree. This immediately confirms the intuition that to be problematic, the distribution with the smaller mean must also have a larger variance, because $y > 0$ immediately implies $\sigma_2 > \sigma_1$. It also leads to the following theorem:

Theorem 2: *Limited Optimality of Normal Expected-Outcome*

Let $M_1, M_2, ..., M_b$ be an optimal ordering of (d, b, f)-trees with identical branching factors and depths, and normal leaf distributions specified by means $\mu_1, \mu_2, \cdots, \mu_b$, and standard deviations $\sigma_1, \sigma_2, \cdots, \sigma_b$, respectively.

If $\forall M_i, M_j : (i < j) \Rightarrow (\mu_i - \mu_j) > z_{\Xi_b}(\sigma_j - \sigma_i)$, then

$lim_{d \to \infty}$ Pr[expected-outcome will make optimal moves] $= 1$.

Proof:

Any two evaluators, such as expected-outcome and Ξ_b-outcome, will agree over a set of moves if and only if they agree on a choice between any two elements of the set. Lemma 1 reduces the problem to determining how often expected-outcome and Ξ_b-outcome will fall on different sides of y. The equivalence of the median and mean on normal curves casts expected-outcome as a quantile-based

function for which $z_Q = 0$. Disagreement between the evaluators will occur if and only if $0 < y < z_\Xi$. Since $y = (\mu_1 - \mu_2)/(\sigma_1 - \sigma_2)$, which by assumption is greater than z_{Ξ_b}, expected-outcome and z_{Ξ_b}-outcome will agree on a choice between any two moves. According to corollary 2, Ξ_b-outcome makes optimal moves in the case described, and thus expected-outcome will, as well. □

In general, agreement between any two evaluators can be promoted by assuming that $(\sigma_2 - \sigma_1) < z_\Xi(\mu_1 - \mu_2)$. According to [Nau82b], $lim_{b\to\infty}\xi_b = 0$. For example, $\xi_2 = .382$, while $\xi_{50} = .056$. Z-scores grow slowly, however, so that in a tree with a branching factor of 2, (.618)-outcome sets $z_\Xi \approx 0.3$, and with $b = 50$, (.944)-outcome sets $z_\Xi \approx 1.6$. Since z-scores greater than three never occur, if $(\mu_1 - \mu_2) > 3(\sigma_2 - \sigma_1)$, ideal expected-outcome will always make the same move as ideal Ξ_b-outcome, for any b. Disagreement between the evaluators, then, implies one of two occurrences: either there is a very large discrepancy in variances, or a very small difference in means. At least in the latter case, Even an incorrect decision on the part of expected-outcome will probably not be catastrophic.

The remaining question, then, is will the leaf distributions beneath sibling nodes in a game-tree have similar variances? If the leaf values are generated randomly using different distribution functions, there is no reason to assume that they should. In a real game, however, trees are built and leaf values assigned according to the rules of the game. It has been fairly well established

that sibling nodes in real game-trees have tightly correlated values, primarily because the static information they contain differs by at most one application of the game's rules (i.e., one move) [Nau83b]. Since the subtrees beneath them are generated by a consistent rule set, the distributions will probably be quite similar. Differences among their means, then, are much more likely to occur than differences among their variances, and thus the assumptions necessary to vindicate expected-outcome users are reasonable.

The justification behind using expected-outcome to rank moves in games with arbitrary leaf values, then, is a combination of three factors. First, it is impossible to calculate the value of Q that would guarantee optimal ordering. Second, distribution means contain more information than any other single statistical parameter. Third, and most significantly, under a rather reasonable set of assumptions, namely normally distributed leaf values with larger differences among sibling means than variances, the ordering imposed by expected-outcome will be correct. Thus, the use of expected-outcome functions even in game-trees with arbitrary distributions of leaf values remains theoretically sound.

The support offered by (d, b, f)-trees to expected-outcome, then, can be summed up as follows: In these games, minimax values are (with high probability) determined by a set of theoretical parameters. Expected-outcome functions observe the actual occurence of these parameters, and indicate their true theoretical

values. As the trees grow, both relationships become increasingly strong, and expected-outcome functions become powerful indicators of the moves that would be made by a minimax oracle. Although the proofs all depend on simplifications of the model, the qualitative implications of this result suggest studying expected-outcome in a series of progressively more realistic domains.

5.2 Empirical Evidence

Real games introduce an even further complication: the entire discussion so far has been predicated on the assumption that all available subtrees have identical structures. Although it is not unreasonable to assume that siblings will frequently be structured similarly, they will rarely, if ever, be identical. This realism may shatter any remaining claims regarding the optimality of expected-outcome functions. Unlike most previous work done on (d, b, f)-trees, however, the expected-outcome model does have qualitative implications that should be directly applicable to real games. In general, heuristics are used in domains in which optimal solutions are either unnecessary or not effectively computable. Instead, satisfactory solutions are sought. Without making any pretense of optimality, then, expected-outcome functions can be applied to real games to determine their merit.

Since the ultimate criterion by which an evaluator is judged is its performance

in actual competition, three sets of experiments were run to verify both the rationality of the random-play assumption and the strength of the expected-outcome model in real-game settings. The first set generated some small complete game-trees, calculated the exact numbers of WINs, LOSSes, and DRAWs beneath every position, and compared the completely informed expected-outcome function with a well-known game-specific evaluator. In this context, expected-outcome's decision quality was superior to that of the more standard functions. While these results are encouraging, they are limited to games that are small enough to be searched exhaustively. In the second set of experiments, a regulation-sized game was used to test the accuracy of estimated expected-outcome functions by basing its decisions on the average value of randomly sampled paths. A player relying on this sampler was pitted directly (no lookahead) against a standard evaluator, with the result that expected-outcome significantly outplayed its opponent. Unfortunately, the cost of random sampling is prohibitive. Thus, the objective of the final set of experiments was to produce an *efficient* estimator. By performing a regression analysis on the sampler's estimated expected-outcome values, a learning preprocessor was able to automatically generate the coefficients in polynomial scoring functions. Once again, the results were positive: a player using the learned coefficients played as well as one using coefficients that had been designed by an expert, even though the learning procedure had no infor-

mation about the game other than its rules, leaf values, and an easily observable set of game features. Taken as a whole, this series of experiments offers strong empirical support for the expected-outcome model.

Before proceeding with the details of these experiments, it is important to point out that because the theoretical development of the model occurred concurrently with the empirical testing, several marginally different implementations of expected-outcome were studied. One particular deviation from the final model that warrants mention is the assignment of values to DRAWs in the random sampling (section 5.2.2) and Othello learning (section 5.2.3) experiments. These tests were run under the assumption that DRAWs can be ignored completely (i.e., they were not counted as leaves). Although this treatment of DRAWs may be justifiable on rational grounds, (no one plays to draw while they can still win, and thus, at least during the mid-game, DRAWs have minimal impact on decisions), it is not consistent with the final definition of expected-outcome, and should be avoided in the future. Fortunately, the experiments ignoring DRAWs all involved Othello trees, in which the miniscule percentage (less than .5%) of DRAW leaves effectively avoids the issue of proper quantification. Tests run on chess trees, in which between one-third and one-half of the leaves were DRAWs, showed that conclusions reached under the two formulations were essentially equivalent; DRAWs had a very marginal impact on the learned coefficients, and

would thus only infrequently affect the decisions made.

5.2.1 Decision Quality

The first step in investigating a model's propriety is determining its *decision quality*, or the frequency with which it recommends correct moves. In the case of expected-outcome, this is tantamount to inquiring how often the move with the largest (or smallest, as appropriate) weighted average of WIN leaves beneath it is, in fact, optimal. Since optimal moves are defined by complete minimax searches, their calculation is contingent upon knowledge of the entire subtree beneath them. Thus, for this first set of experiments, only fairly small games could be studied. Moreover, in order to compare the decision quality of expected-outcome with that of a more standard function, popular games with small variants were needed. Tic-tac-toe and Othello were chosen. Tic-tac-toe (naughts-and-crosses in the UK) is a game that should be familiar to everyone; Othello, although of growing popularity, may not be. The standard game is played on an 8-by-8 board. The playing pieces are discs which are white on one side and black on the other. Each player, in turn, fills a legal vacant square with a disc showing his own color. Whenever the newly placed disc completes a sandwich consisting of an unbroken straight line of hostile discs between two friendly ones, the entire opposing line is captured and flipped to the color of the current mover. A move

is legal if and only if at least one disc is captured. When neither player can move, the one with the most discs is declared the winner. (For a more detailed description, see [Fre80] [Mag79] [Ros82]).

These games contributed four small variants for study, although only two of them, 3-by-3 tic-tac-toe and 4-by-4 Othello, actually have game-trees that are small enough to generate entirely. The other two, 4-by-4 tic-tac-toe and 6-by-6 Othello, were chosen because they are small enough for large portions of their trees to be examined, yet large enough to offer more interesting testbeds than their smaller cousins. For each of these medium-sized games, ten initial configurations were generated by making a set number of random moves, twenty for 6-by-6 Othello and seven for 4-by-4 tic-tac-toe. The entire trees beneath these setups — involving the last nine moves in tic-tac-toe and the last twelve in Othello — were examined.

For each game studied, every node in the tree (beneath the initial configuration) was considered by four functions: complete minimax, expected-outcome, a previously studied standard, and worst-possible-choice. The decisions recommended by these evaluators were compared with the optimal move, or the move recommended by minimax, and a record was kept of their performance. Minimax, by definition, never made an error, and worst-possible-choice served as a record of all errors possible. Unlike complete minimax, expected-outcome did not use a

backup algorithm to combine leaf values; its decisions were based strictly on evaluations of a node's immediate descendants. Finally, the standard evaluators were taken from published literature (see the appendix for a detailed description) and calculated using only static information: the open-lines-advantage for tic-tac-toe described in [Nil80], and an Othello weighted-squares function based on the one in [Mag79]. The number of errors made by each function is shown in table 1, as is the number of decision nodes in each tree. Since most evaluators recognize leaves as a special case, decisions made in the presence of leaves is not indicative of a function's true quality. Thus, only nodes whose best successor (as selected by minimax) were not leaves, were considered decision nodes. Figures 5 and 6 graph the relative percentage of possible errors made by each of the imperfect functions (i.e., expected-outcome, open-lines-advantage, weighted-squares).

A quick glance at the results reveals several interesting points, perhaps the most significant of which is the evaluators' relative error-frequency — in tic-tac-toe, expected-outcome made roughly one-sixth as many errors as open-lines-advantage, and in Othello about one-third as many as weighted-squares. Although the relative number of incorrect decisions in a search space is a fair basis for comparing two evaluators, the absolute accuracy of the functions depends on the percentage of possible errors that they make. In all instances tested, expected-outcome made only a small percentage of the errors possible. In terms

Decision Quality				
Game	Decisions	Errors		
		Standard	E-O	Possible
3x3 Tic-tac-toe	69945	4384	676	35859
4x4 Tic-tac-toe	201179	1924	368	52492
	188316	2138	328	63983
	135123	2510	414	28204
	130929	2427	394	40942
	125303	1672	994	40219
	124439	3214	1188	38735
	108301	4115	382	35539
	85887	1613	224	30479
	72227	2619	328	23336
	56395	2118	334	24627
4x4 Othello	69308	3396	1632	6632
6x6 Othello	1151130	46221	19513	106136
	1119937	11945	3528	29257
	856265	59076	14982	114240
	840106	66310	20837	150864
	834440	38052	14053	84448
	826710	83046	22676	175716
	458663	26000	10319	59547
	377671	14712	5074	35221
	210004	11104	5393	24117
	196446	11551	3768	22428

Table 1: This table contains the output of the decision quality experiments. The decisions column records the number of nodes whose best successor, as selected by minimax, was not a leaf. The number of possible errors was determined by a function that always selected the successor with the worst minimax value; if it did not err, no error was possible. For the other two evaluators, a move was considered an error if its minimax value was non-optimal.

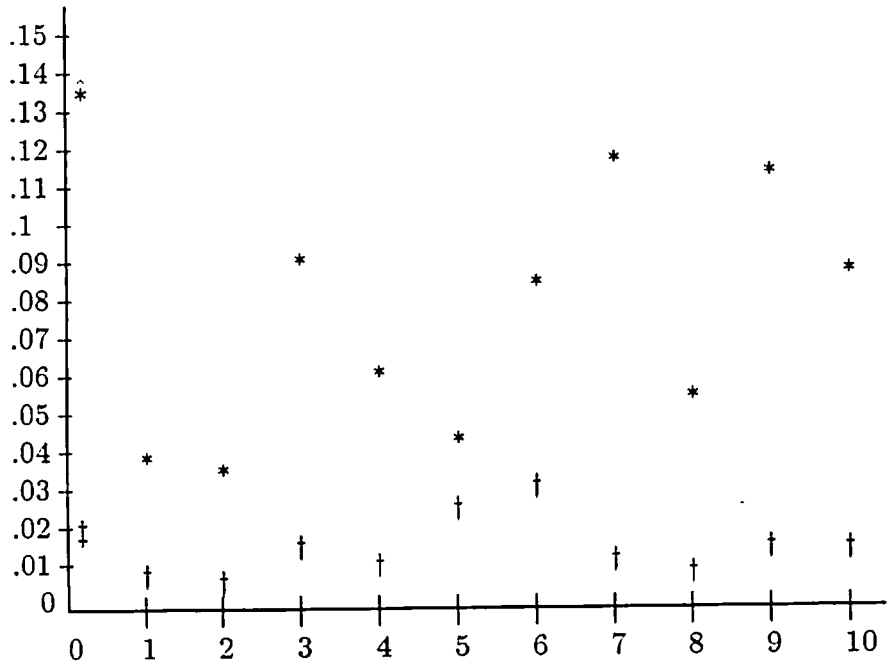


Figure 5: The percentage of possible errors made by open-lines-advantage (*) and expected-outcome (†) on eleven tic-tac-toe trees. Entry 0, (⁑ and ‡, respectively), corresponds to the 3-by-3 tree. The other ten entries are complete subtrees beneath randomly generated 4-by-4 tic-tac-toe positions with seven markers already on the board. Approximate average accuracy: open-lines-advantage 92.3%, expected-outcome 98.6%.

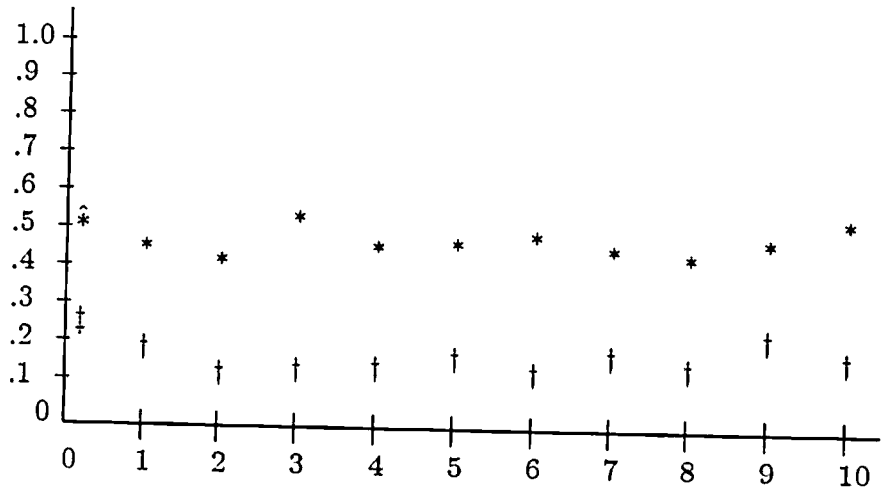


Figure 6: The percentage of possible errors made by weighted-squares (*) and expected-outcome (†) on eleven Othello trees. Entry 0, (* and ‡, respectively), corresponds to the 4-by-4 tree. The other ten entries are complete subtrees beneath randomly generated 6-by-6 Othello positions with twenty-four discs already on the board. Approximate average accuracy: weighted-squares 58.2%, expected-outcome 83.4%.

of the standard evaluators, the results are consistent with observed performance. The open-lines-advantage function for tic-tac-toe is known to be fairly strong [Nil80]. When implemented with sufficient lookahead, it can both force a draw and take advantage of its opponent's errors. (A simple set of tests showed that in the 3-by-3 case, a one-ply lookahead is generally sufficient). Weighted-squares, with a mean accuracy of roughly 58% on trees with an average branching factor around seven is reasonable, but not overwhelmingly effective, as an Othello evaluator. This is not surprising. A thorough analysis of the game showed that weighted-squares strategies were overly simplistic, unable to account for issues like mobility and stability, and not sensitive to the differences between opening, mid-game, and end-game strategies. An evaluator that took these items into consideration formed the basis of IAGO [Ros82], a world-championship level program[8]. Since expected-outcome consistently outperformed both of its expert-designed competitors, it is possible to conclude that, at the very least, expected-outcome evaluators are strong in tic-tac-toe and reasonable in Othello.

In addition to performing well, the expected-outcome model offers a means of

[8]IAGO is more than just a sophisticated static evaluation function. Among other things, it considers phase-of-game, completely solves end-game positions, and uses a precomputed table to determine a configuration's "edge-stability". This is the most significant component of its evaluator, and the algorithm used to derive the table relies on a weighted-squares formulation. Because of its complexity and extensive use of table look-up, I did not feel that IAGO was a particularly appropriate challenger for the early testing stages of a new model. The purpose of these experiments was simply to test the decision quality of a new model of evaluation functions: any expert-designed function, such as weighted-squares, should provide a useful comparison.

quantifying some long-standing observations that have generally been discussed qualitatively. For example, the decision quality of all tested functions deteriorated greatly as the tree was descended. Most previously studied evaluators have claimed not to be tailored to end-game play, without really explaining why. In the case of expected-outcome, the reason is clear: statistical parameters like distribution means are only useful for large populations. Since end-game nodes have relatively few descendants, the utility of expected-outcome to the end-game should be suspect. The basic point made by these experiments, however, is that in all cases tested, expected-outcome not only made fewer errors than the standard functions, but chose the optimal move with relatively high frequency. This indicates that guiding play in the direction of maximum win percentage constitutes a reasonable heuristic. Thus, the expected-outcome model has passed its first test: exact values generally lead to good moves.

5.2.2 Random Sampling Strategies

According to the the decision quality results, if complete information is available, moving in the direction of maximum win percentage is frequently beneficial. Of course, these are precisely the cases in which optimal moves are always computable. Since probabilistic (and for that matter, heuristic) models are only interesting when knowledge is incomplete, some method of estimating expected-

outcome values based on partial information is necessary. The obvious technique is random sampling. Expected-outcome values, by their very definition, represent the means of leaf-value distributions. In the second set of experiments, a sampler-based estimate of expected-outcome was pitted against a weighted-squares function in several matches of (8-by-8) Othello. Although Othello, like most parlor games, actually ends in one of {WIN,LOSS,DRAW}, ratings in tournament play are adjusted based on the final victory margin. For the moment, only the friendly interpretation of the game will be considered. Victory margin statistics are useful for comparing player strength, and will be used in section 5.2.3 to learn evaluation functions. For the remainder of this section, however, an Othello game will be considered either won, lost, or tied. These experiments, like those which investigated decision quality, were designed as pure tests of evaluator strength — neither player used any lookahead. The aim of these tests, then, was to show that sampler-based functions can compete favorably with those designed by experts, at least in terms of playing strength. As far as efficiency goes, they are not comparable. The sampler consistently required orders of magnitude more time than the static player. Efficiency, however, will be considered later. The purpose of these matches was simply to investigate the competitive abilities of estimated expected-outcome.

matches of regulation-size (8-by-8) Othello. To show the relationship between estimation quality and performance level, several different samplers were tested. Those that were able to derive estimates fairly quickly were soundly trounced by weighted-squares. The performance of two time consuming samplers, however, helps stress both the potential value of expected-outcome and the potential inefficiency of random sampling. The first, and weaker of the two, set $\epsilon = .05$, $N = 8$, and stopped checking for convergence if none had been found by the time 256 samples were taken. For the most part, this sampler needed between one and ten minutes per move, as opposed to a maximum of two seconds for weighted-squares. Their accuracy, however, was nearly identical. Four 100-game matches were played, and in each, the competition was about even; in no case did either evaluator win enough games to make a viable claim of superiority. From the sampler's viewpoint, the win-loss-draw scores of the four matches were 46-48-6, 41-53-6, 48-49-3, and 54-41-5, for a total score of 189-191-20. The second sampler made several further sacrifices of efficiency for accuracy to ensure improved performance. In this implementation, N was set to 16, a maximum of 1024 samples were taken, and progressively smaller discrepancies (ϵ) were allowed between successive $\hat{\mu}_i$'s[11]. The resulting program ran roughly one-tenth

[11]For each sample size between 16 and 1024, the number of expected wins out of 1024 games was predicted. If two consecutive estimates were within the given range of tolerable error, convergence was assumed. Between 16 and 32 samples, the tolerable error was 0. For 32 and 64 it was 16, for 64 and 128, 24, for 128 and 256, 20, for 256 and 512, 18, and for 512 and 1024, 9.

as fast as the first one, but with much greater success. In a 50-game match, this sampler defeated its weighted-squares opponent 48-2, a victory so overwhelming that its superiority should be evident.

Veteran Othello players may feel that the number of victories alone is insufficient to accurately gauge the relative strength of two players. Perhaps of even greater significance is the margin of victory — the single most important feature in determining a player's USOA (United States Othello Association) rating[12]. Over the first 400 games, weighted-squares outscored the weaker sampler 13,295 to 11,748, for an overall advantage of 1547 discs. As expected, it did not fare as well against stronger opposition; its 50 game total of 894 discs was 1,079 shy of the 1,973 racked up by the stronger sampler. To effectively study the victory margins, a random variable that represents the disc differential,

$$\mathcal{D} = \text{(weighted-squares' score)} - \text{(sampler's score)},$$

must be defined. Assuming that $\mathcal{D}$ is distributed approximately normally, standard tests regarding the means of normal distributions can be used to answer some questions about the functions' relative strengths. First, using the disc-differential criterion, does the weaker sampler still appear equivalent to weighted-squares? In other words, given that $\mathcal{D}$'s mean and standard deviation for the

[12]During tournament play, ratings are adjusted to reflect the projected margin of victory in a game between two rated players. Although it is not the only factor considered when determining a player's rating, disc differential is a highly significant one. As a general rule, every ten rating points separating two players adds a disc to the anticipated margin of victory. The interval between player classes is about 200 points, for a projected differential of 20 discs [Ric81].

400-game match were experimentally estimated at $\tilde{\mu}_{\mathcal{D}} = 3.87$ and $\tilde{\sigma}_{\mathcal{D}} = 14.69$, respectively, what is the probability that the true value $\mu_{\mathcal{D}} = 0$? The two-tailed hypothesis testing procedure indicates that this probability is smaller than .005 — it is fairly safe to say that the first sampler is not quite as good as weighted-squares. The test also indicates, however, that with probability .95, the actual difference between them is $\mu_{\mathcal{D}} \in [2.80, 4.94]$. Given the USOA rating system, a projected victory margin of 2 to 5 discs is not particularly significant. Thus, it is clear that although the functions are not identical, their caliber of play is essentially equivalent. Second, what does the rating system have to say about the stronger sampler? For its 50 game match, the experimental estimates were $\tilde{\mu}_{\mathcal{D}} = -21.58$ and $\tilde{\sigma}_{\mathcal{D}} = 11.21$ (negative means favor the sampler). The possibility of function equivalence is not even worth considering; the 95% confidence interval for $\mu_{\mathcal{D}}$ is $[-24.97, -18.19]$, rating the second sampler about a full class better than weighted-squares[13].

These studies show that efficiency considerations aside, sampler-based functions can compete within the realm of expert-designed evaluators. It is important, however, to keep the results in their proper perspective. As a demonstration of the world's best Othello evaluator, they are woefully inadequate — the absence

[13]The correspondence between player strength and disc-differential suggests designing samplers that estimate expected-disc-differential-outcome. Since this idea was developed long after the initial sampling experiments were run, I felt that its implementation was not crucial. In the next section, however, an Othello function learned to approximate expected-disc-differential-outcome is shown to be superior to two that were learned to estimate expected-outcome.

of lookahead makes the games unrealistic, the difference in computation times skews the results, and the competition is not as strong as it could be. Their sole purpose was to establish estimated expected-outcome as a function that can play at least on par with those designed by experts, and the data clearly substantiates the claim. Expected-outcome functions, then, do appear to make useful decisions in interesting settings. Given no expert information, the ability to evaluate only leaves, and a good deal of computation time, they were able to play better than a competent (albeit non-masterful) function that had been hand-crafted by an expert. Thus, the second challenge has been met: in the absence of perfect information, an expected-outcome estimator made reasonably good decisions.

5.2.3 Learning Expected-Outcome Functions

Although the results of the random sampling experiments are strongly positive as far as they go, they do leave a good deal to be desired. As things stand, the estimation of expected-outcome values gives evaluation functions a clear sense of purpose. It does not, however, provide function designers with any useful tips. In fact, the clearly unacceptable time discrepancy between the samplers and their static opponent points out the difficulty of sampling in competition-oriented settings. This situation may satisfy theoreticians, who are primarily interested in understanding models, but it is anathema to programmers, who

would like to use said models to design stronger programs. Thus, efficiency considerations must now be addressed.

Like most products, evaluation functions incur costs in two phases of their development, design and implementation. The inefficiency of sampler-based functions is accrued during implementation; their design is simple and cheap, because an effective sampler need only understand the game's rules and be able to identify leaves as WINs, LOSSes, or DRAWs. Static evaluators, on the other hand, rely on detailed game-specific analyses, frequently at the cost of many man-years of work. To help reduce these design costs, a variety of automatic tools that improve static evaluators have been developed, the simplest of which attempt to determine the relative significance of several given game features. Techniques of this sort are called *parameter learning* [Sam63] [Sam67] [Gri74] [CK86], and should be applicable to learning the relationship between game features and expected-outcome values. The use of parameter learning introduces two deviations from the ideal expected-outcome function. The first, reliance on an estimated estimate is easily justified by efficiency considerations and empirical evidence. The second, and perhaps more problematic, involves the use of expert-selected domain features in what claims to be a domain-independent model. It is important to point out, however, that the experiments described in this section are significant primarily from an implementation point of view. A model's theoretical

accuracy is not effected by the (in)efficiency of its implementations. Although the model itself remains domain-independent, then, expertise helps increase its efficiency to the point where it may be used to make real-time decisions. Thus, even though reliance on predetermined game features limits a function's conformity to the domain-independent ideal, scoring polynomials are the backbone of most competitive game programs, and if done properly, the learned functions should combine the statistical precision and uncomplicated design of sampler-based functions with the implementation efficiency of static evaluators. The next set of experiments involved learning static expected-outcome estimators of just this sort.

Static evaluation functions contain two components — a finite feature set, $F_j, j = 1, \ldots, n$, and a corresponding coefficient set, $C_j, j = 1, \ldots, n$ — which combine to specify a unique value for any given board, $V = \sum_{j=1}^{n}(C_j F_j)$. Probably the best known example of a feature set is used by material functions in chess, P (pawn), R (rook), N (knight), B (bishop), and Q (queen)[14]. Any assignment of values to the pieces represents a valid evaluator. As a further example, the weighted-squares function that was studied in the decision quality and random sampling experiments used a feature set consisting of ten equivalence classes of squares (figure 7), and a coefficient set defined by the weights

[14]In general, these functions work with material advantage, and thus the feature value (not the coefficient) for P, say, would be given by (Black Pawns − White Pawns).

assigned to each class. Each set of game features defines an infinite family of evaluators; differing coefficients define distinct, albeit related, functions, and assign boards different values. Once again, the absence of a well-defined model of evaluation functions makes it difficult to either design appropriate coefficient sets or understand the static values. The expected-outcome model, however, suggests both a clear objective and a definite interpretation: determine the relationship between the expected-outcome value, (EO), and the game features, and set the coefficients such that $V = EO = \sum_{j=1}^{n}(C_j F_j)$.

The simplest method for deriving the appropriate weights is *regression analysis,* a statistical technique for developing predictors of a single *dependent* random variable, Y, given the values of some more easily observable *independent* variables, $X_j, j = 1, \ldots, n$. The input into a regression procedure is a matrix with $(n + 1)$ columns and m rows, $m > n$. Each row corresponds to an observed instance of the problem. Within a row, the entry in the j^{th} column represents either the observed value of X_j, if $1 \leq j \leq n$, or the calculated value of Y, if $j = (n + 1)$. Since a system of simultaneous equations with more equations than variables is overspecified, (assume w.l.o.g. that no two rows differ by only a multiplicative constant), there is no set of coefficients such that $\forall i, (1 \leq i \leq m), \sum_{j=1}^{n}(C_j X_{ij}) = Y_i$. This forces the definition of an estimated variable, $\hat{Y}$, by $\forall i, (1 \leq i \leq m), \sum_{j=1}^{m}(C_j X_{ij}) \stackrel{\text{def}}{=} \hat{Y}$. The purpose of a regression

1	2,9	4	5	5	4	2,9	1
2,9	3,10	6	6	6	6	3,10	2,9
4	6	7	8	8	7	6	4
5	6	8	8	8	8	6	5
5	6	8	8	8	8	6	5
4	6	7	8	8	7	6	4
2,9	3,10	6	6	6	6	3,10	2,9
1	2,9	4	5	5	4	2,9	1

Figure 7: These equivalence classes of squares were defined implicitly by the weighted-squares function outlined in the appendix. Squares adjacent to vacant and occupied corners belong to distinct classes. The ability to recognize these ten (and only these ten) game features is the characteristic of the weighted-squares evaluator family. Each member of the family defines its own set of weights.

procedure is to find the set of coefficients that produce the "best" estimate, or the set that minimizes the sum of the squared errors, $\sum_{i=1}^{m}(Y_i - \hat{Y}_i)^2$. Consider, for example, the problem of fitting a line to a series of points. In general, if there are more than two points involved, no straight line will pass through all of them. It is possible, however, to find the closest straight line approximations by minimizing the sum of the squared errors. In addition to line fitting, regression techniques have become quite popular among economists and other social scientists, and were first successfully implemented as a method of developing polynomial scoring functions in [CK86]. A related statistical technique, factor analysis, may be able to retain the performance characteristics of regression while requiring less in the way of expert input. This idea will be discussed in section 7.

Othello To find a member of the weighted-squares family that estimates expected-outcome values, a regression procedure can be used to learn square-class weights that approximate EO. Since the exact expected-outcome value is not computable in interesting games, the regression's dependent variable, $Y = EO$, must be determined by a random sampler. The first stage of the learning experiments was implemented by setting the dependent variable, $Y = \hat{EO}$, to an estimate determined by the stronger of the samplers described in the previous section. Recall that this sampler predicted the number of games that black would win if 1024 were completed randomly, and stopped when it converged. 5000 random initial

configurations were generated, with no attempt made to introduce the expertise necessary to determine whether they represented "realistic" positions. Each row of the resultant 5000-by-11 input matrix contained the square-class[15] and $\hat{EO}$ values of a different board. These configurations were devised by randomly choosing a start depth between 0 and 60, (with 30 being the most likely and probabilities decreasing symmetrically as distance from the mid-game increased), and then making the specified number of random moves. The resulting board served as the instance recorded in the matrix. Two independently generated matrices were used to derive evaluators L1 and L2 of table 2. The development of two different coefficient sets should come as no surprise; regression analysis is concerned only with finding best fits to the observed instances. Coefficients yielding good fits need not be unique, the difference of a few key observations may shift the choice of "best" between them, and larger input matrices might force them to converge to a single set. Nevertheless, since most of the values in the two sets are fairly similar, (as is the performance level that they generated), the coefficients both define reasonable evaluators, and both warrant investigation.

Evaluator L3 was learned using the same technique, but a different interpretation of expected-outcome. As mentioned before, the USOA rating system is highly dependent on a player's victory margin. This suggests studying

[15]For each square in the class, a black disc added +1 and a white disc -1, to the square-class value.

Coefficients for Weighted-Squares Functions					
Square-class	M	A	L1	L2	L3
1	64	64	33.57	56.16	56.77
2	-30	-30	-4.83	-7.38	-5.59
3	-40	-40	-3.23	-6.20	-36.46
4	10	10	3.84	6.63	5.88
5	5	5	3.00	4.19	4.73
6	2	-3	1.79	3.13	2.75
7	5	2	2.10	3.38	2.43
8	1	1	1.00	1.00	1.00
9	5	5	10.77	13.40	17.54
10	5	-2	1.62	3.87	1.67

Table 2: The coefficients used by each of five weighted-squares evaluators. Function M was copied directly from [Mag79]. A is a modification of M which accounts for my personal experience. Functions L1, L2, and L3, were learned by regression analysis as static estimators of expected-outcome values. L1 and L2 considered leaves as either WINs or LOSSes, and attempted to estimate the percentage of WINs beneath each node, while L3 approximated a node's expected-disc-differential.

expected-disc-differential-outcome, the first application of expected-outcome to a leaf distribution with a large range of potential values (specifically, all integers in $[-64, 64]$). Recall that even on (d, b, f)-trees, expected-outcome may not be optimal in instances of arbitrary leaf distributions. However, since no notion of ξ_b can conceivably be defined for Othello, it should be reasonable to assume that the disc-differentials are distributed normally. In any event, this assumption was tested empirically, and shown to be correct.

The regression was performed by SAS[16], a popular commercially available statistical applications package, which, in addition to performing regression analyses, also returns an analysis of the variance (ANOVA) over the instances reported in the input matrix. ANOVA data characterizes the probable accuracy of the regression's determined parameters and the predictive value of the independent variables on the dependent one. Without getting sidetracked into the specifics of the regression, there are a few points that are worth mentioning. First, any time a regression procedure is used on a feature set, F, to predict the value of a dependent variable, Y, two crucial questions arise: do the individual elements of F contribute to Y? and does the model described by F effectively predict Y? SAS provides answers to both: with probability $> .9999$, each of the square-classes contributes to $\hat{EO}$, and, in all three cases tested, the overall model has a fit somewhere in the $(0.6, 0.7)$ range, (where 0.0 indicates no relation and 1.0 is a perfect predictor). Second, it is important to consider confidence in a set of coefficients before they are applied to some further problem, such as the design of an evaluation function. As probabilistic entities, the parameters returned by SAS, in this case the coefficients, should be reported along with their standard errors (standard deviations). According to the SAS analysis of square-classes, the standard errors were all fairly small, and thus, the determined coefficients

[16]SAS is the registered trademark of SAS Institute Inc., Cary, N.C., U.S.A.

are sufficiently reliable to be incorporated with confidence into an evaluator.

Measures of model fit between 0.6 and 0.7 indicate that weighted-squares functions provide reasonable, albeit not overwhelmingly powerful predictors of expected-outcome values. This is directly analogous to the assertion that weighted-squares functions can play up to a certain level, but for championship play, other factors must be considered [Ros82]. The correspondence between a feature set's ability to predict expected-outcome values and the quality of play that it generates, offers another piece of evidence that the expected-outcome model does, in fact, provide an accurate characterization of static evaluator strength.

The accepted test of evaluation functions, however, is how well a program that uses them plays. Two of the weighted-squares functions shown in table 2, (M and A), were designed by experts[17], while the others (L1, L2, and L3) were learned by regression analysis. Competition among them demonstrated how well coefficients learned to approximate expected-outcome values fare against those developed by experts as measures of "node strength", for the same feature set. Prior to setting up a tournament, however, it is interesting to consider whether these are, in fact, the same goal. Have previous evaluators actually (subconsciously) been designed to approximate expected-outcome? A comparison of the designed and learned functions do not give a conclusive answer. In many respects, the coefficient sets

[17]Function M was copied directly from [Mag79]. Function A is a modification of M to account for my personal experience. Most of the tests run indicate that A is slightly stronger than M.

are similar; they all recognize the supreme importance of corners, the relative advantage of edge squares over non-edges, and the danger of taking a square near a vacant corner. They differ greatly, however, in many of their actual weights. While this strongly indicates the existence of a realtionship between the precise model of expected-outcome and the intuitive notion of node strength, the values are hardly close enough to substantiate a claim that they are identical.

The second stage of the learning experiments consisted of a series of Othello matches. Unlike the functions studied in the decision quality and random sampling experiments, all weighted-squares evaluators are efficiently calculable. This finally allows the lookahead ban to be lifted, and more realistic games to be studied. The rules of the tournament were simple. Every pair of functions met in one 400-game match, broken down into four 100-game sets, one each with lookahead length fixed at 0, 1, 2, and 3. The first six moves of a game were made randomly, and the evaluators took over at move seven. When the game was completed, the functions swapped colors, and replayed all but the first six moves. This was done to insure a wide variety of games without introducing an unfair bias. Thus, in each set, games were played beginning from each of fifty different initial setups. Over the course of 400 games, *only the expected-disc-differential function (L3) consistently outplayed its opposition; none of the others, either learned or designed, were able to do so.* It is interesting to note that there are only minor

Othello Tournament: 400 games per match							
					Scores (number of discs)		
Match	Function	W	L	D	Total	μ	σ
1	M	181	212	7	11906	29.765	10.06
	A	212	181	7	12467	31.1675	10.99
	Difference (A − M)	—	—	—	561	1.4025	19.77
2	M	196	193	11	12088	30.22	12.82
	L1	193	196	11	11750	29.375	11.05
	Difference (L1 − M)	—	—	—	−338	−0.845	21.78
3	M	191	198	11	12112	30.28	12.93
	L2	198	191	11	11863	29.6575	11.50
	Difference (L2 − M)	—	—	—	−249	−0.6225	22.88
4	A	213	178	9	12267	30.6675	12.17
	L1	178	213	9	11613	29.0325	9.34
	Difference (L1 − A)	—	—	—	−654	−1.635	19.70
5	A	200	184	16	12154	30.385	12.38
	L2	184	200	16	11411	28.5275	9.39
	Difference (L2 − A)	—	—	—	−743	−1.8575	19.77
6	L2	203	186	11	12538	31.345	12.33
	L1	186	203	11	11869	29.6725	11.30
	Difference (L2 − L1)	—	—	—	−669	−1.6725	22.25
7	L3	252	141	7	13564	33.91	11.42
	M	141	252	7	10856	27.14	11.52
	Difference (L3 − M)	—	—	—	−2708	−6.77	21.99
8	L3	215	176	9	12554	31.385	10.26
	A	176	215	9	11481	28.7025	11.52
	Difference (L3 − A)	—	—	—	−1073	−2.6825	20.39
9	L3	260	132	8	13743	34.3575	11.57
	L1	132	260	8	10537	26.3425	9.94
	Difference (L3 − L1)	—	—	—	−3206	−8.015	19.99
10	L3	269	124	7	13917	34.7925	12.29
	L2	124	269	7	10187	25.4675	10.55
	Difference (L3 − L2)	—	—	—	−3730	−9.325	21.01

Table 3: A match-by-match recap of the tournament among five weighted-squares evaluators.

4 Match (1600 Game) Totals						
Function	W	L	D	Points For	Points Against	Difference
L3	996	573	31	53778	43061	+10717
A	801	658	41	48369	47484	+875
M	709	855	36	46962	49644	−2682
L2	709	846	45	45999	50152	−4153
L1	689	872	39	45769	50636	−4867

Table 4: Totals from the weighted-squares tournament.

differences between L3's coefficients and those used by its competitors. Like L1 and L2, L3 felt that the experts both undervalued classes 2 and 9, overvalued class 4, and straddled class 10 (M was too high, A too low). It agreed with the expert analysis of class 3, however, and agreed with M on class 6 and A on class 7. Apparently, the five coefficient sets are all close enough to play comparably. L3's slight edge may be attributable to choosing correctly among some of these more controversial class values.

In the matches among A, M, L1, and L2, not only were the scores fairly close, (see tables 3 and 4), but the disc-differential statistics were, as well. An analysis of the victory margins shows that with probability .95, no two of these functions would be rated more than 35 USOA points apart. Since roughly 200 points (actually, 207 [Ric81]), separate the player classes, the rating spread is rather insignificant — it should be clear that all four functions are essentially equivalent. L3, on the other hand, was noticeably stronger than the other functions, albeit not overwhelmingly so; its victory margins ranged from 39 to 145, and its pro-

jected ratings were 12 to 109 points above its competitors. The disc-differential statistics also offer some specific characterizations of relative evaluator strength (see table 5). In each match, the player who scored the greater number of total discs (not the one with the most victories) is assumed to be the stronger of the two. Hypothesis testing techniques for normal distributions indicate both the significance of the distinction between players and, (with 95% confidence), the actual difference in their playing strengths. In match 3, for example, function M appeared to be very slightly stronger than function L2. The significance of this advantage, however, was only .4 — not terribly convincing. In fact, with probability .95, M's true advantage is between -1.07 (actually, a disadvantage of 1.07 discs) and 2.32. Overall, it is unlikely that the functions are of identical strength. It is clear, however, that their playing ability is essentially equivalent.

The superiority of L3 over L1 and L2 adds credibility to the claim that the conditions needed for the use of expected-outcome with arbitrary leaf distributions are reasonable; the performance of all three learned functions shows that machine generated, expected-outcome based coefficients are at least as good as those designed by experts. One potential counter-claim, of course, is that a function's strength is derived primarily from its feature set, not its coefficient set. If this is true, any two members of the same family should perform comparably, and it's not surprising that the new functions competed favorably with the old

Analysis of the Othello tournament				
Match	Players	Advantage	Significance	.95 Range
1	M,A	A	.85	$[-0.06, 2.86]$
2	M,L1	M	.6	$[-0.81, 2.50]$
3	M,L2	M	.4	$[-1.07, 2.32]$
4	A,L1	A	.9	$[0.13, 3.14]$
5	A,L2	A	.94	$[0.33, 3.38]$
6	L2,L1	L2	.86	$[0.03, 3.31]$
7	L3,M	L3	$>.99$	$[5.18, 8.36]$
8	L3,A	L3	.99	$[1.17, 4.19]$
9	L3,L1	L3	$>.99$	$[6.52, 9.51]$
10	L3,L2	L3	$>.99$	$[7.74, 10.91]$

Table 5: For each match, the player with the higher total score is considered stronger. The significance column gives the (approximate) probability with which the perfect parity hypothesis can be rejected. The .95-range column calculates the actual difference between the players, given the experimental data. With probability .95, the true difference lies within the given range.

ones. To squelch any doubts that may arise along these lines, some further family members were generated. A, M, L1, and L2 each played an additional match against a weighted-squares cousin with a randomly generated set of coefficients. All four random functions were trounced — they rarely won at all, and would be rated at least a player class behind the four that had been intelligently designed. With its strong showing in the tournament, the expected-outcome model has met the third challenge: an efficiently calculable estimator played fairly well.

Chess The efficient implementation of expected-outcome estimators in Othello suggests the first general guideline for designing two-player evaluators. Nevertheless, although the model was defined in a domain-independent manner, its utility has only been demonstrated in two domains, tic-tac-toe and Othello. Most aspects of these games are quite different, yet they do share certain gross characteristics: in both games, the playing pieces (markers or discs) are differentiated only by owner, and once played, are fixed in position. A convincing demonstration of the model's generality should involve a game that is as dissimilar to the first two as possible. The obvious choice is chess; computer chess has generated a larger corpus of expert literature than all other games combined, and thus there are many conventional wisdoms that can be used as bases of comparison (discussions of absolute accuracy are obviously out of the question).

One challenge posed by chess to expected-outcome is to learn material values

for the chess pieces. This problem has been studied extensively, dating back at least to the 18^{th} century, when Euler suggested that a piece's value should be proportional to its average mobility. Over the years, a general consensus has developed regarding the appropriate values, although there have been many variations on the basic theme. Mikhail Botvinnik's chess program, PIONEER [Bot84], used the values

$$P=1,\ R=5,\ N=3,\ B=3,\ Q=9$$

as part of its evaluation module. Other experts may value a bishop slightly higher than a knight or assign ten to a queen, but the modifications are minor; for all intents and purposes, Botvinnik's coefficients are representative of expert-designed material functions. As a standalone static evaluator, material is known to be rather weak [Sha50] [Lev76] [Bot84]. In addition to ignoring positional and phase-of-game considerations, material evaluators tend to produce essentially random openings and end-games. Nevertheless, the design of an expected-outcome based material evaluator is a sophisticated enough task to test the model's applicability to chess, and the plethora of available expertise should make the results' interpretation fairly noncontroversial.

Chess pieces, like square-classes, constitute a feature set for which coefficients must be learned to define an evaluation function. Once again, learning-by-regression is an appropriate method for coefficient set design. Regression analy-

sis has been used in the past as a method for learning material values, although its previous implementation was iterative, as a progressive coefficient improver [CK86]. These experiments, by way of contrast, use the same single regression technique that was so successful at learning weighted-squares functions for Othello. As in the Othello experiments, the regression's dependent variable, $Y = \hat{EO}$, was determined by a random sampler that predicted the number of games that black would win if 1024 were completed randomly from the given random initial configuration, and stopped when it converged[18]. Since the relative tree sizes and DRAW-leaf densities of the two games cause chess samplers to be orders of magnitude slower than Othello samplers, fewer initial configurations could be generated. In this case, the starting depth was set (with uniform probability) between 0 and 125, and the appropriate number of random moves were made. The resultant 350-by-6 matrix[19], however, was large enough for the regression to be fairly confident in the coefficients that it developed, albeit slightly less so than it was for the square-class weights; the standard errors were all small, (relative to the reported parameters), and in every case tested, all pieces contributed to $\hat{EO}$ with probability $> .99$. Another interesting statistic reported by SAS involves

[18]Unlike the Othello samplers, these did consider DRAWs. This will be discussed in detail momentarily.

[19]Additional configurations were later generated to verify the stability of the coefficients. A regression run on a 1000-by-6 matrix showed only slight perturbations in the relationships learned. I did not feel that these minor deviations warranted re-running any of the other experiments.

the proximity of the tightness (or perhaps more precisely, the looseness) of fit for the model to the observed strength of material functions. Recall that ANOVA data indicates the predictive value of the model, in this case material, on the dependent variable, or expected-outcome. Whereas the model fit measures were all between 0.6 and 0.7 for Othello square-classes, the chess material functions landed in the neighborhood of 0.35. This indicates that the predictive value of the material model on expected-outcome is poor. Once again, then, the statistically determined relationship between a feature set and the expected-outcome value corresponds closely to the empirically observed relationship between that feature set and its quality of play [Sha50] [Lev76] [Bot84].

Before a chess sampler could be implemented, one issue that was essentially glossed over in the Othello studies had to be addressed: the proper interpretation of DRAWs. Whereas random DRAWs are quite rare in Othello, the distribution of DRAW leaves in the chess tree is rather dense. According to the definition of the expected-outcome model, DRAWs should probably count as half-wins. Meanwhile, gaming intuition suggests that neither player finds DRAWs attractive while they can still win[20]. Since this essentially describes everything but the end-

[20]The relative density of DRAWs in the two games suggests another potential problem. It should be possible to construct games in which virtually every path leads to a DRAW. When reasonable players compete, however, only a very small subset of the moves are considered, and thus many actual games will terminate as either WINs or LOSSes. In this type of game, the number of samples needed to differentiate between moves, may be prohibitive. As theoretical constructs, such games simply indicate that sampling is an heuristic technique that can be foiled by adversary arguments. Whether any real games fall into this class, however, is a question that

game, and expected-outcome values of end-game positions are suspect anyway, DRAWs may, in fact, be nothing more than noise. To determine what effect, if any, the different interpretations would have on the results, the regression was run twice. The resultant functions are not shown on any consistent scale; since the coefficients in a set are only significant as they relate to each other, (and not as they relate to coefficients in another set), the functions are all scaled to best demonstrate the relationships learned.

- Chess1: $EO = \frac{(WINS+.5DRAWS)}{(WINS+LOSSES+DRAWS)}$,

  P=4.0, R=4.8, N=2.5, B=3.3, Q=9.6.

- Chess2: $EO = \frac{(WINS)}{(WINS+LOSSES)}$,

  P=4.0, R=4.9, N=3.0, B=3.7, Q=10.4.

Several aspects of these functions are interesting. First, regardless of the treatment of DRAWs, the piece values learned were about the same. Since DRAWs tend to look equally (un)appealing to both players, it is encouraging to find that they have minimal impact on expected-outcome values. Second, and more significant, are the coefficient sets themselves. As presented above, the numbers were scaled to illustrate their success at learning valid relationships

can only be resolved empirically. To date, all that can be stated conclusively is that this is not a problem in tic-tac-toe, Othello, or chess.

among the four major pieces; other than the quadrupled pawn values, both sets are remarkably close to the one proposed by Botvinnik, and certainly well within the acknowledged ranges.

The interpretation of these coefficient sets is not clear. The proximity of four of the learned coefficients to the predicted ones indicates that there must be some relationship between expected-outcome and the criteria applied by experts. The inflated pawn values, on the other hand, are problematic. Whereas the experts have agreed that pawns are the least valuable pieces on the board, the experimental results indicate that they are worth more than knights and bishops. This implies that expected-outcome is unable to capture some of the game's subtleties. One potential explanation for pawn inflation, however, has been provided by the very experts who relied on unit pawn values — the history of material functions implies that the difficulty may lie not in the learned coefficients, but rather with the original idea of appraising pieces according to their relative mobility. The primary purposes of a major piece are to move, control parts of the board, and attack opposing pieces; their relative strengths should depend heavily on their relative mobility. Pawns, on the other hand, serve different purposes — setting up defensive blockades and getting promoted to queens. (Although pawn promotion may occur only rarely during actual play, the threat is ubiquitous, and strategic play must be prepared to defend against unchallenged pawns in

open columns). Game designers have long recognized that position independent, (specifically row-independent), pawn values are universally less meaningful than equally consistent values for the other pieces, and have generally compensated by including a special *pawn formation* component in their evaluators [Lev76] [Sha50] [Bot84]. In addition, a previous attempt to automatically generate material evaluators fell prey to the same difficulty of overvalued pawns, albeit to a somewhat lesser extent [CK86].

The general (expert) consensus about pawn values, then, is that they are not really equal to one; assigning them a unit value is only reasonable if they are upgraded as their importance (mostly positioning) grows. There are instances, for example, when pawns are the most valuable pieces on the board — an unchallenged pawn in rank six is worth more than an idle rook (say, one not immediately threatening a checkmate). Precisely what a reasonable "average" pawn value would be (if such a thing is definable), has not been widely discussed, and is not a matter of consensus. Nevertheless, it is intuitively unappealing to think that pawns may be more valuable (on the average) than either knights or bishops, as the learned coefficient sets suggest. One potential explanation of this anomaly is that since the sampler was implemented with the assumption that all promoted pawns would become queens, the determined coefficient of 4.0 is actually for a pawn/queen, not just for a pawn. To test this hypothesis, an ad-

ditional set of experiments, in which pawns that reached the opposite side were removed from the board, was run. The coefficient sets learned for the resulting game of chess-without-promotion were:

- Chess3: No pawn promotion, $EO = \frac{(WINS+.5DRAWS)}{(WINS+LOSSES+DRAWS)}$,
  P=1.0 R=4.4 N=1.5 B=2.4 Q=6.8.

- Chess4: No pawn promotion, $EO = \frac{(WINS)}{(WINS+LOSSES)}$,
  P=1.0 R=5.6 N=2.0 B=3.0 Q=7.5.

This approach successfully deflated the relative importance of pawns, but it also perturbed the relationship among the other pieces. Although the R, N, B, and Q coefficients learned without pawn promotion are not drastically outside the mainstream, they would probably be considered a weaker set than those learned when pawns were promoted. This is, of course, understandable. The accepted values were designed for games with pawn promotion; losing pawns when they cross the board could significantly alter game-playing strategies. Ultimately, there is no real solution to the problem of inflated pawn values, other than to include additional factors in the regression matrix. For example, if rather than including a separate component for pawn formation, expert functions included seven values for pawns (one for each row), the learning procedure might be as successful at determining these accepted values as it was at learning the major

piece weights.

Since the coefficients assigned to pawns are acknowledged to be less meaningful than those assigned to the major pieces, and the learned major-piece weights all approximated Botvinnik's expert set, it should be reasonable to assume that the evaluators will play comparable chess. The recognized weakness of material functions, however, renders conclusions based on the performance of programs that rely on them dubious, regardless of coefficient set. Nevertheless, it is important to compare the learned coefficients with a more accepted set, and (due in part to the absence of a valid criterion for comparison), the only way to do this is to pit them against each other in a series of games. A tournament was run involving eight evaluators drawn from three sources (see table 6). The expert-designed Chess0 was borrowed from PIONEER [Bot84], while Chess1 through Chess4 were learned to approximate expected-outcome, using the various formulations described above. Chess5, 6, and 7 were introduced to investigate the feature set's inherent strength: Chess5 made consistent moves using random criteria by assigning each piece a random value between one and 500, Chess6 ignored material and moved randomly, and Chess7 valued all pieces equally, and simply counted the number of them on the board. Every pair of functions clashed 500 times, with each evaluator getting white 250 of them. In addition to allowing their material evaluators to play random openings and end-games, the programs

Chess Material Functions										
Function	P	R	N	B	Q	W	L	D	Points	Source
Chess4	1.0	5.6	2.0	3.0	7.5	1150	686	1664	1982	Learned
Chess0	1	5	3	3	9	1152	710	1638	1971	PIONEER
Chess3	1.0	4.4	1.5	2.4	6.8	1116	696	1688	1960	Learned
Chess2	4.0	4.9	3.0	3.7	10.4	1111	705	1684	1953	Learned
Chess1	4.0	4.8	2.5	3.3	9.6	1049	710	1741	1919.5	Learned
Chess7	1	1	1	1	1	1047	749	1704	1899	Random
Chess5	98	79	47	375	45	947	877	1676	1785	Random
Chess6	0	0	0	0	0	79	2518	903	530.5	Random

Table 6: This table describes a chess tournament among eight material evaluators. Five of them were designed intelligently — Chess0 by an expert (Botvinnik), and Chess1 through Chess4 to approximate expected-outcome values under a variety of marginally different assumptions. The other three were introduced as controls: Chess5 makes consistent decisions using random criteria, Chess6 makes random moves, and Chess7 counts all pieces equally.

were removed even further from reality[21] by foregoing lookahead and attempting no quiescence detection.

Despite the unrealistic setting for the tournament, 3500 games should have been enough for distinctions among the evaluators to appear. None did. If the standings were set strictly in terms of number of victories, the expert-designed Chess0 would win — although only by two games. Counting draws as half points (a la professional hockey), on the other hand, would place the trophy in the hands of Chess4, the function learned by removing pawns and ignoring draws. Taken

[21] Used loosely, to refer to the world of truly competitive chess programs

as a whole, the spread separating the five intelligently designed evaluators, (103 wins or 62.5 points), is completely insignificant. Their overall performance was essentially equivalent. Perhaps an even greater surprise, however, is how well two of the random functions fared. Although neither Chess5 nor Chess7 played quite on par with the first five functions, their performance was consistently almost as good. (In fact, the difference between the uniform values of Chess7 and the weakest learned function, Chess1, was only 2 wins or 20 points). Against all competition, however, the completely random Chess6 was soundly trounced. Thus, it seems that in chess, at least under the simplistic implementation used in this tournament, the most significant aspect of the material evaluator family is its feature set — even random coefficients were substantially better than none at all. In what may be a characteristic of weak feature sets, these results seem to indicate that although absurd coefficients sacrifice a slight edge, any set that is even remotely reasonable will play at about the same level. In all likelihood, however, as the inappropriate standalone use of these evaluators is corrected and they become incorporated into highly sophisticated programs, differences in play should begin to appear.

The challenge posed for expected-outcome by chess was rather humble because material evaluators are not expected to lead to strong play. With the realization that nearly any coefficient set would have competed reasonably with

the accepted one[22], the proximity of the relationship learned among the major pieces to the consensus set is noteworthy; the standard values were derived as measures of relative mobility, while the learned coefficients were developed as estimates of the probability of winning a random game. The inflation of pawn values, on the other hand, remains troubling. Although mobility was identified by experts as a significant factor in playing winning chess, adopted as the measure of the relative importance of the major pieces, and generally regarded as an inaccurate assessment of the role of the pawns, the experimentally derived coefficients would have been more convincing if they had approximated the consnesus set on all five pieces. Taken as a whole, then, a comparison of the learned coefficients with the accepted set makes a rather interesting point: the relative mobility of the four major pieces is related to the proportion of winning games in which they appear, while the mobility of the pawn is not. Compounded with the correspondence between the feature set's predictive ability on expected-outcome values and the known caliber of evaluators that use it, the model's performance in the chess domain appears to corroborate the claim that it is not strictly domain-dependent. The full extent of its applicability to chess, however, remains to be seen.

[22]To the best of my knowledge, the tournament described above is the first evidence of the essential futility of trying to fine-tune material values. Since competition among material evaluators has frequently been used as a means for comparing their accuracy, this result calls some previous conclusions about material functions into question.

Part III

Conclusions

6 Contributions: What's Been Accomplished?

This thesis focused on the classic artificial intelligence model of decision-making in adversarial settings: two-player games. The stated purpose of the research was to remove a fundamental obstacle from our understanding of decision-making in games, namely the absence of a precise, meaningful, useful, and domain-independent model of static evaluation functions. To fill this void came the expected-outcome model, which defines a node's value as the expected value of the game's outcome, given random play from that point on. As a statistical interpretation of game-trees, rather than games, the model immediately provided the necessary precision and generality. Its accuracy was then demonstrated through a series of proofs and experiments:

- Section 5.1 demonstrated that given any set of trees chosen from a particular simple class, an ideal expected-outcome function would select the best one. This correspondence between the random-play and perfect-play assumptions motivated the model's study in progressively more realistic

settings.

- In section 5.2.1 the model was implemented on some small versions of tic-tac-toe and Othello, where its decision quality was shown to be better than that of some popular functions that have been studied in the past. Expected-outcome was shown to be better than 95% accurate on tic-tac-toe trees of average branching factor four, and about 85% accurate on Othello trees of average branching factor seven. Neither of the standard functions studied came close to this level of decision quality.

- An estimated expected-outcome function was implemented via a random sampler in section 5.2.2, where it was shown that given enough time, the sampler could defeat a standard Othello evaluator. Although the most powerful sampler tested required several orders of magnitude more time to move than its standard static opponent, its superiority made a significant point — it is possible to achieve reasonable play through expected-outcome estimates made by randomly sampling leaves.

- Efficiency considerations were brought back into play in section 5.2.3, where expected-outcome was combined with a bit of domain knowledge and a parameter-learning technique to develop static evaluators in Othello and chess. Once again, the learned functions were able to compete with a more

standard set.

Taken as a whole, these results offer strong support for both the validity of the expected-outcome model and the rationality of its underlying assumptions. The idea of regarding non-terminal nodes in a search space as random variables is new. In the past, it has always been assumed that these nodes have exact, albeit unknown values. Maximizing expected utilities via expected-outocome is only the first application of these ideas. Several other techniques are discussed in the next section. In the long run, it may well turn out that the major contributions of this research were the redefinition of search trees as probabilistic entities and the resultant formalization of heuristic search theory that this new model makes possible.

The extent to which expected-outcome will affect our understanding of game-tree search procedures remains to be seen. At the present time, its significance lies in its clarification of the static evaluator: for at least a large class of games, expected-outcome evaluators were shown to be strong when exact, reasonable when estimated, and efficiently calculable. It is, of course, unreasonable to expect the initial implementation of any new model, regardless of inherent merit, to match the achievements of thirty-five years of progressive research. The functions derived for Othello and chess would not fare well in actual competition against more standard programs, although the scoring functions developed for

common feature sets in both games hint that previous ad hoc evaluators may have been unknowingly estimating expected-outcome values. Many of the popular programming techniques, however, are in no way contradictory to the new model, and could easily be used in conjunction with expected-outcome functions. The major immediate contribution of expected-outcome to the programming of specific games, then, is that it provides a degree of precision to some of its more ambiguous components, most notably the design of mid-game evaluation functions.

7 Implications: Where Might the Model Lead?

The expected-outcome model of game-trees, unlike all previously studied models, is essentially probabilistic in nature. In addition to providing a precise definition for the static evaluator, the interpretation of internal nodes as random variables rather than deterministic values (e.g., as specified by minimax), suggests rethinking virtually every component of game programming. This section lists some interesting questions that should be reconsidered in light of the new model. Although the model's extension into these areas is still primarily speculative, a brief discussion of the issues that they bring up should reveal some potentially radical new approaches that become possible in a probabilistic setting.

Full-depth Search One of the model's most distinctive features is its use of full-depth, rather than full-width, searches. The random sampling experiments (section 5.2.2), for example, searched entire paths and evaluated only nodes whose exact values were calculable (i.e., leaves). This involved searching only a relatively small, randomly chosen subset of the paths beneath each node. A standard Shannon Type-A program, on the other hand, explores all paths, but only a relatively small portion of each one, via a full-width search to a fixed depth and an estimation of frontier node values [Sha50]. In this strategy, uncertainty comes from the estimates of the positions at the search horizon, whereas in the new model uncertainty is due to sampling error. The strong performance of full-depth search programs suggests an interesting line of future inquiry: what is the most effective use of computational resources? The standard answer to this question has long been that if additional computation time becomes available, the search frontier should be extended. Full-depth search algorithms, on the other hand, suggest traversing additional paths. In the absence of evaluation functions, full-width searches are meaningless while full-depth samplers are strong. The presence of powerful static evaluators, on the other hand, has an as-of-yet undetermined effect on full depth strategies, but allows full-width strategies to make good decisions. Thus, both answers are correct at times. Future analyses may be able to determine the relationship between the two procedures, character-

ize their relative propriety as functions of evaluator strength (now well-defined), and recommend optimal resource allocation to system designers.

Learning When people learn to play games, their performance is dependent on their ability to recognize both significant game features and reasonable moves. Despite the recent success of fast brute force programs [Ber81] [BE86], it has long been acknowledged that true mastery of a game requires an understanding of both factors; Shannon's discussion of computer chess incorporated domain features into static evaluators and unreasonable move pruning into type-B strategies [Sha50]. The main contribution of expected-outcome to the field of machine learning is that it suggests a potential reinterpretation of "reasonable" play. In a deterministic game-tree, moves are either right or wrong. Any definition given for reasonable play, then, is necessarily ad hoc, and dependent on vague notions such as degree of optimality, or player strength [RB83]. Any node in a probabilistic game-tree, on the other hand, should be selected occasionally, with probability corresponding to its relative evaluated standing among its siblings. In other words, a minimax analysis of a game position is that the best node is assigned probability one of being selected, while all others will be chosen with probability zero. When node values are viewed as distribution means, on the other hand, it is clear that their "quality" overlaps, and thus it becomes reasonable to occasionally choose nodes other than the one with the best evaluation.

Although this is still somewhat vague, it suggest a question that may lead to some interesting results: given a well defined notion of evaluator strength and a probabilistic interpretation of node values, is it possible to disambiguate the notion of reasonable play, as well?

Furthermore, although not strictly dependent on expected-outcome, some of the experimental procedures discussed in section 5.2 suggest potentially powerful algorithms for learning stronger evaluators. Implicit in the technique of learning by regression analysis, (originally proposed in [CK86]), are related procedures that may be able to recognize significant game features and gradually improve scoring polynomials. First, although an initial set of features must be identified by an expert, it is possible to automatically convert a large set of potential features into a smaller set of relevant ones by a technique known as *factor analysis.* Factor analysis, which has been widely studied in econometrics and the behavioral and social sciences, is a statistical procedure that discovers relations among a set of variables [Sch77]. Regression analysis, by way of contrast, explains a relationship that is already assumed to exist. Second, iterating the Monte Carlo studies described in section 5.2.3 may lead to stronger evaluators. Consider the following selective sampling procedure: on the first run, initial configurations are generated, values of the dependent variables are calculated, and random paths are traversed to develop a 0^{th}-order evaluator. On the second run, this

0^{th}-order function is used to evaluate and scale a move's successors. Paths are then traversed with appropriate probabilities to yield a 1^{st}-order evaluator. This procedure leads to a selective sampling of the leaves, can be iterated as often as desired, and might produce successively stronger evaluators. The expected-outcome model provides these techniques with a well-defined value to use as the dependent variable, and may be central to their implementations.

Quiescence Analysis The full-depth search suggested by the expected-outcome model effectively resolves the problem of quiescence. Since search frontiers are not set and only leaves are evaluated, the issue is rendered moot. With the introduction of efficiently calculable static estimators, however, the problem of quiescence resurfaces. Will statically estimated expected-outcome functions be as susceptible to the horizon effect as other evaluators? Since the learning procedure that was used to design the efficient estimators included searches all the way to the leaves, future extensions and analyses of the procedure may be able to resolve this question in a positive light.

Backup Strategies In competitive game implementations, moves are almost never based directly on static information — a backup strategy is generally used to combine tip node values and (hopefully) improve decision quality. Partial minimax is both the most popular and the most successful of these strategies;

pruning algorithms, clever record keeping techniques, and special purpose architectures have helped elevate partial minimax based machines to master level in chess [CT82] [BE86]. Despite its widespread use, the procedure of minimaxing estimated values on partial trees has no theoretical basis. In fact, the attempt to equate minimaxing estimates with estimating minimax violates one of the cardinal sins of statistics: a function of estimates is not the same as an estimate of a function. Error-propagation analyses have shown that rather than filtering out errors, minimax tends to compound them [Pea83], and studies performed on (d, b, f)-trees have diagnosed partial minimax as a potentially pathological algorithm [Nau83a]. In addition, a variety of proposed backup strategies have been shown to recommend stronger decisions under certain circumstances (see [Abr86a] for a survey of these strategies). The prevalent justification for partial minimax, then, has been that it seems to work well in most game settings. Whether this is true or not, it is rather unsatisfying.

A probabilistic interpretation of reasonable play should suggest an interesting backup strategy: calculate the probability with which each move will be chosen, and pass up the corresponding weighted sum of their values. The intuitive appeal of this strategy is that it may account for reasonable play, as opposed to the simpler, but obviously incorrect, assumptions of perfect or random play. Even if this algorithm is provably more accurate than partial minimax, however, it may

still be less desirable; if all nodes above the frontier represent potential moves, pruning is impossible. Since one of the most salient features of partial minimax is that its companion algorithm, α-β-pruning, greatly deepens search and effectively increases the amount of useful information available to the decision maker, this is a significant loss. For the sake of efficiency, then, it may be desirable to weaken the reasonable-play assumption somewhat. Rather than considering all moves as viable options, perhaps only moves within a certain threshold of the best warrant consideration. Looked at in this way, then, partial minimax belongs at one end of a spectrum of backup algorithms; its threshold is zero, and it always assigns a node's best child probability one of being chosen. A similar idea has recently been discussed in [Riv86]. Future analyses may discover some other interesting strategies along the spectrum, and develop a notion of accuracy vs. efficiency tradeoffs.

8 Reprise: Why Study Games?

The opening section of this thesis discussed some early motivation underlying the mathematical study of games. The original idea was that since games are simple models of decision-making, understanding games should help us understand decisions. From that point on the general problem was essentially ignored, and only game playing was discussed. Nevertheless, the results in this thesis were all

done with an eye towards eventual extensibility; additional domain knowledge was brought into play only as a means of implementing the model efficiently. The expected-outcome model focused on the least satisfactory aspect of the current theory of two-player games, namely the static evaluator. The resultant model immediately lifted the design of static evaluators from a domain-dependent, game-by-game task, to one which is well defined for a large class of game-trees. The ease with which the implications drawn on (d, b, f)-trees (section 5.1) transferred to actual game-trees in tic-tac-toe, Othello, and chess (section 5.2), makes it reasonable to expect them to be applicable to more complex domains as well.

The redefinition of game-trees as probabilistic data structures suggests rethinking many of the basic assumptions that have governed the fields of game programming and heuristic search theory. In game-trees, a distinction was noticed between leaves, which had definite set values, and internal nodes, whose values were characterized by random variables instantiated at one of the leaf values beneath them. Looked at in a more general context, the leaves can be viewed as members of a small class of "stable" states with easily assessable values. All other states are "unstable," and will take on actual values only when a stable state is reached. Given a rigid set of rules describing the passage through the state-space from that point on, (e.g. perfect play), unstable states can be assigned exact values as well. In a probabilistic setting, however, there is no reason

to do so. Weaker assumptions governing the behavior of agents in the domain make the values of these internal nodes indeterminate, yet predictable. In this way, the concepts developed on game-trees should extend to more complex search spaces.

In the long run, then, these techniques may be applicable to a broad class of real world problems: those that can be represented usefully in a state-space formalism, contain a small class of states whose values are easily recognizable and can be accurately assessed, and have a (relatively small) finite set of "reasonable" operators (although most real-world problems have infinite, or at least very large, branching factors, many potential operators can frequently be ruled out). Hopefully, future analyses will build on the results presented here, and help expand our understanding of decisions (and our ability to mechanize them) from strictly simple domains into more complex ones.

References

[Abr86a] Bruce Abramson. *Control Strategies for Two-Player Games.* Technical Report, Columbia University, May 1986.

[Abr86b] Bruce Abramson. An explanation of and cure for minimax pathology. In Laveen Kanal and John Lemmer, editors, *Uncertainty in Artificial Intelligence*, North Holland, 1986.

[BB86] Subir Bhattacharya and Amitava Bagchi. Making best use of available memory when searching game-trees. In *Proceedings of the fifth National Conference on Artificial Intelligence*, 1986.

[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.

[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.

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

[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.

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

[Bra80] M.A. Bramer. Correct and optimal strategies in game playing programs. *The Computer Journal*, 23:347–352, 1980.

[CK86] Jens Christensen and Richard Korf. A unified theory of heuristic evaluation functions and its application to learning. In *Proceedings of the fifth National Conference on Artificial Intelligence*, 1986.

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

[CN86] Ping-Chung Chi and Dana S. Nau. Predicting the performance of minimax and product in game-tree searching. In *Proceedings of the 2nd Workshop of Uncertainty in Artificial Intelligence*, pages 49–55, 1986.

[CN87] Ping-Chung Chi and Dana S. Nau. Comparing minimax and product in a variety of games. In *Proceedings of the sixth National Conference on Artificial Intelligence*, 1987.

[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.

[Fel57] William Feller. *An Introduction to Probability Theory and Its Applications.* John Wiley and Sons, 1957.

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

[Fre80] Peter W. Frey. Machine othello. *Personal Computing*, 89–90, 1980.

[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.

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

[Lev76] David Levy. *Chess and Computers.* Computer Science Press, Inc., 1976.

[Mag79] Peter B. Maggs. Programming strategies in the game of reversi. *BYTE*, 4:66–79, 1979.

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

[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.

[Nil80] Nils J. Nilsson. *Principles of Artificial Intelligence.* Tioga Publishing Company, 1980.

[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.

[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.

[Rab76] M. O. Rabin. Probabilistic algorithms. In J.F. Traub, editor, *Algorithms and Complexity: New Directions and Recent Results*, pages 21–39, Academic Press, 1976.

[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.

[Ric81] R. Richards. The revised usoa rating system. *Othello Quarterly*, 3(1):18–23, 1981.

[Ric83] Elaine Rich. *Artificial Intelligence.* McGraw Hill, 1983.

[Riv86] Ronald L. Rivest. *Game Tree Searching by Min/Max Approximation.* Technical Report, MIT Laboratory for Computer Science, July 1986.

[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.

[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.

[Sch77] J.H.F. Schilderinck. *Regression and Factor Analysis in Econometrics.* Martinus Nijhoff Social Disease division, 1977.

[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.

[Ste83] Stephen L. Stepoway. Reversi: an experiment in game-playing programs. In M.A. Bramer, editor, *Computer Game Playing: Theory and Practice*, Ellis Horwood Limited, 1983.

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

[Win77] P.H. Winston. *Artificial Intelligence.* Addison Wesley, 1977.

A Standard Evaluation Functions

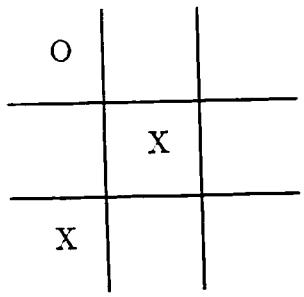
The evaluation function used for tic-tac-toe is rather simple. Each line (row, column, and diagonal) on the board represents a potential win. A line is open to a player if his opponent has no marks in it. The open-lines-advantage evaluation function for position T, $OLA(T)$, is given by:

$$OLA(T) = \begin{cases} \infty & \text{if } T \text{ is a win for X} \\ -\infty & \text{if } T \text{ is a win for O} \\ OPEN_T(X) - OPEN_T(O) & \text{otherwise} \end{cases}$$

where $OPEN_T(Player)$ = the number of lines open to $Player$ in T.

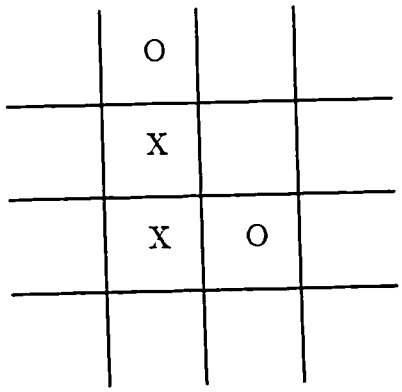
This function was used in [Nil80] to demonstrate issues related to minimax search and static evaluation. It has an intuitive appeal, and works rather well when combined with lookahead. On 3-by-3 tic-tac-toe, a one ply lookahead is sufficient to force a DRAW, which is, in fact, the outcome of the game when both players are perfect. Examples of this function on 3-by-3 and 4-by-4 boards can be found in figure 8.

The function used for Othello is a bit more complex. The basis of a weighted-squares strategy is the realization that not all squares on the board are of equal value; edge squares are less likely to be flipped than squares in the center, and corner squares will never be flipped at all. Squares immediately adjacent to the corner, on the other hand, are frequently detrimental to the player moving there



$OPEN_T(X) = 5$
$OPEN_T(O) = 2$
$OLA(T) = (5 - 2) = 3$

(a)



$OPEN_T(X) = 5$
$OPEN_T(O) = 5$
$OLA(T) = (5 - 5) = 0$

(b)

Figure 8: This was the standard expert-designed evaluator used in the tic-tac-toe tests. An example is shown on both 3-by-3 (a) and 4-by-4 (b) boards.

first, because they allow the other player to take the corner. Once the corner has been taken, however, they lose that special status. Weighted-squares strategies have been discussed in the literature and implemented in many programs [Mag79] [Ste83]. The function used for 8-by-8 Othello is a weighted-squares strategy, based on the one presented in [Mag79], and modified to account for my personal experience with the game. Although it is not immediately clear how the 8-by-8 function should be applied to smaller boards, the convention chosen was to consider squares based on their locations with respect to the corners. The values for each square are shown in figures 9 and 10. Note that the large numbers were retained because they were not derived scientifically in the first place. Thus, although the 64 in the corners came from the number of squares on an 8-by-8 board, there is no reason to believe that it is not equally reasonable for corners on a smaller board as well. Leaves were recognized when neither player was able to move, (this occurs trivially when the board is full), and classified as WINs, LOSSes, or DRAWs as dictated by the final score.

64	5,-30	10	5	5	10	5,-30	64
5,-30	-2,-40	-3	-3	-3	-3	-2,-40	5,-30
10	-3	2	1	1	2	-3	10
5	-3	1	1	1	1	-3	5
5	-3	1	1	1	1	-3	5
10	-3	2	1	1	2	-3	10
5,-30	-2,-40	-3	-3	-3	-3	-2,-40	5,-30
64	5,-30	10	5	5	10	5,-30	64

Figure 9: The expert-designed weighted squares function for 8-by-8 Othello.

64	5,-30	10	10	5,-30	64
5,-30	-2,-40	-3	-3	-2,-40	5,-30
10	-3	2	2	-3	10
10	-3	2	2	-3	10
5,-30	-2,-40	-3	-3	-2,-40	5,-30
64	5,-30	10	10	5,-30	64

(a)

64	5,-30	5,-30	64
5,-30	-2,-40	-2,-40	5,-30
5,-30	-2,-40	-2,-40	5,-30
64	5,-30	5,-30	64

(b)

Figure 10: The expert-designed weighted squares function given in figure 9, reduced to (a) 6-by-6, and (b) 4-by-4 Othello.

B The Random Sampler

The experiments described in sections 5.2.2 and 5.2.3 used a random sampler to estimate expected-outcome values. This section outlines the sampler's implementation in C-like pseudocode.

```
/*  This function generates random paths, determines leaf
    values, and estimates the expected-outcome value of
    an input board */

int Estimate(Board)

    /* First, initialize variables.  Set the WIN, LOSS, DRAW,
       and LEAF tallies to 0, indicate that no estimate has
       been converged to, and start with an initial guess
       of -1. Set the initial sample size, and search the
       appropriate number of paths. */

  WINS = LOSSES = DRAWS = LEAVES = 0;
  estimate = -1;  convergent = 'N';
  samplesize = 16;  /* Essentially arbitrary.  I used 16. */
  while (the estimate has not converged) {
    while (samplesize has not been reached) { /* Keep going */
      while (not at a leaf) {
          /* Find all legal moves, and randomly choose one. */
        options = the number of legal moves;
        if (options != 0) randomly choose a move;
        }   /* End of the path */
      LEAVES++;                  if (Black won) WINS++;
      if (White won) LOSSES++;   if (Draw) DRAWS++;
    } while (LEAVES < samplesize);

     /* Either project the expected-outcome value over a fixed
        number of games (to avoid floating-point math), or
        represent it as a real number.  Any of the formulations
        mentioned in section 5.2.3's discussion of chess
```

```
        can be used. */

     previous = estimate;
     estimate = projected expected-outcome value over 1024 games;

     /* If the difference between the previous and current
        estimates is within the tolerable error bounds, an
        estimate has been converged to. Otherwise, double the
        sample size, and continue. */

     if (|estimate-previous| <= Tolerable) convergent = 'Y';
     else {pass *= 2; tries++;}
    }   /* End of pass.  Start over */
   return(estimate);
} /* End Function Estimate */
```