THE UNIVERSITY OF ALBERTA

# Probabilistic Analysis of Search

BY

Liwu Li

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES AND RESEARCH IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE DEGREE OF *Doctor of Philosophy*

DEPARTMENT OF *Computing Science*

Edmonton, Alberta

SPRING 1989

ISBN 0-315-52867-2

THE UNIVERSITY OF ALBERTA

*RELEASE FORM*

NAME OF AUTHOR: Liwu Li

TITLE OF THESIS: Probabilistic Analysis of Search

DEGREE FOR WHICH THIS THESIS WAS PRESENTED: Doctor of Philosophy

YEAR THIS DEGREE GRANTED: 1989

(Signed) Liwu Li

Permanent Address:

CC/#58, Southern Small Road

The District of the Red Bridge

Tianjin, China

Dated: April [illegible], 1989

I shot an arrow in the air,

It fell to earth I know not where.

*Henry Wadsworth Longfellow*

THE UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research, for acceptance, a thesis entitled **Probabilistic Analysis of Search** submitted by *Liwu Li* in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

T. A. Marsland
..........................................
Supervisor

[illegible]
..........................................

[illegible]
..........................................

[illegible]
..........................................

[illegible]
..........................................

M. Newborn
..........................................
M. Newborn

Date: 27 February 1989

Dedicated to

my Dad and Mom,

Zeng-Xiang Li and Feng-Xien Chen Li

## *Acknowledgements*

The author would like to take this opportunity to thank the people who helped make this work possible. My first acknowledgement goes to my supervisor Professor Tony Marsland for his guidance and encouragement. Second, I would like to thank the other members of my examining committee: Drs. Macki, Nyland, Dobosiewicz and Culberson for their helpful comments, and professor M. Newborn of McGill University for being the external examiner and taking the time to read and comment on my thesis.

This research would not have been possible without financial support from the Department of Computing Science at The University of Alberta, and the Natural Sciences and Engineering Research Council (NSERC) of Canada.

Finally, I would like to thank my brothers and sister for making my education possible. Special thanks are due to my wife, Shiying, for her love and for giving me a reason to finish this work.

## *Abstract*

The performance of decision making in computer game playing programs is determined by the search algorithm in their inner loops. This thesis involves several aspects of the special AI area, computer game playing, which include game t[illegible] modeling, search pathology investigation, pruning efficiency analysis, and position evaluation representation. It focuses on the application of probability.

A new probabilistic model for game trees is proposed. In this model, node value dependence is simulated with conditional probabilities. This model eliminates some curious properties of the existing models, and its flexibility reflects the great variety of real games. Some interesting properties of this model are derived. One of them makes it possible for both the root and terminal nodes to have the same probability to take the same score.

The minimax pathology that searching deeper makes the backed-up values more random is investigated for the new model. It is shown that for a certain class of games, minimax search does reduce the error propagation. Formulas are also derived for computing the probabilities of making a correct decision when an obvious evaluation function is used. The calculation provides strong evidence to support the relation between minimax search benefit and node value dependence.

The commonly used game tree search algorithm, the alpha-beta pruning algorithm, is analyzed for more practical game trees. Recursive equations for the average number of visited terminal nodes are derived. In this way, the effect of node value dependence on the pruning efficiency can be analyzed.

The alpha-beta pruning algorithm can also be generalized for probability-based game tree search when the strength of a node is described by a probability distribution.

The new algorithm inherits some good properties from its point-value version. Several variations of this algorithm are presented, one of which is the "degeneration" of this probability-based algorithm into a range-based one.

**Table of Contents**

## List of Figures

## List of Tables

Chapter 1

# Introduction

## 1. An Overview

Significant problems of decision-making in adversary settings like stock-market investment and military management are ubiquitous in modern society. Aside from being interesting and entertaining problems in their own right, games like chess and GO bear close resemblance to a wide variety of more important problems, including translation, logical deduction, symbolic computation, battlefield decision making, and earthquake prediction. In any of these fields, skillful performance requires thought; and satisfactory solutions, although generally attainable, are rarely trivial. The games provide well defined rules and have afforded a public testing ground for new algorithms and data structures, which aim at speeding up the decision-making process as well as organizing expert knowledge. The advantage of studying games is that both decision-making and adversary settings can be realized without considering the complexity of the particular domains. The progress in computer game playing demonstrates the power of man-machine cooperation for solving these challenging problems, and the techniques developed exhibit their potential use in decision-making for other artificial intelligence problems. Computer game playing has been justified as a valid scientific pursuit. As a result, computer game playing has gained more and more attention from AI researchers.

Along with an introduction to the topic of game tree search, some important problems that have arisen in the area of computer game playing will be presented in the following. These problems have been discussed by many authors in this area, but there are no simple solutions for them. As a matter of fact, some of these problems have been a focus of dispute. In this presentation, an effort is made to emphasize the difference of some views scattered in the literature.

## 1.1. Game Tree Search and Probabilistic Analysis

The two-person zero-sum perfect information games, which will be dealt with in this thesis, can be characterized by a set of "positions" and a set of rules for moving from one position to another, where the players move alternatively. In computer game playing, a game tree is the most often used data structure in organizing positions. For a game, the game tree is a rooted tree that consists of nodes, which represent positions, and branches, which connect positions with possible continuations.[1] The two players, called Max and Min, take strict alternate turns to move. The nodes in the tree are classified as Min nodes and Max nodes, which correspond to the positions that are the turns for player Min and Max, respectively, to move. Shown in Fig. 1.1 is an example game tree, where Max nodes are represented with squares, and Min nodes are represented with circles. The *degree* of a node is the number of its successors. A leaf node has a degree of 0. The *height* of a node is the maximum number of moves from the node to a leaf node. The *depth* of a node is the number of moves from the root node to it. In Fig. 1.1, the degree of each interior node is 2, the depth of each leaf node is 4, and the height of the root node is also 4.

1. In fact, if each position is represented by exactly one node, a directed graph is derived from this tree by identifying the nodes which represent the same position. Following the literature [1], no attempt will be made to distinguish nodes that represent the same position. In other words, it is assumed that all nodes in a game tree represent different positions.

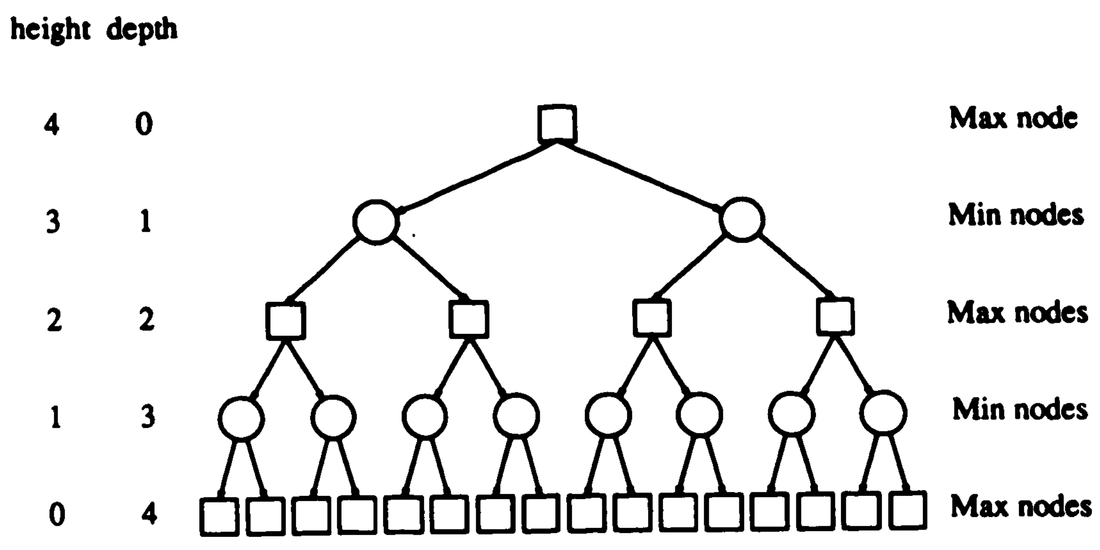


Fig. 1.1 A Game Tree of Height 4

The game tree is an explicit representation of all the possible plays from an initial position of the game. The purpose of game tree search is to find the best move for the players. Because it is rarely possible to exhaust a game tree, the basic method used in computer game playing is bounded look-ahead combined with the use of evaluation functions. The explored part of a game tree is called a *search tree*. The nodes of a search tree are divided into two classes; one consists of the nodes at the current search frontier, which are called terminal nodes,[2] and the other consists of the interior nodes. The value associated with a terminal node is estimated by an evaluation function. The values associated with the interior nodes are calculated in a bottom-up fassion. The process of expanding a game tree, evaluating the nodes at the frontier, and then rolling back the evaluations is known as *game tree search*. As in the literature, when it is not necessary to distinguish the terminal nodes in a search tree from the leaf nodes in the corresponding

2. Note that these terminal nodes do not necessarily represent the terminal positions of the game, only the leaf nodes of the game tree correspond to these positions.

game tree, the term game tree is used to refer to the search tree, and the terms leaf node and terminal node are used synonymously. In this case, we can assume that the computer game playing program searches the complete game tree. This assumption is often used when we analyze game tree search procedures.

Since the performance of computer game playing programs is mainly determined by their search methods, computer game playing needs to develop game tree search techniques. Many different game tree search strategies have been proposed, and some of them have been successfully encoded in game playing programs. At the same time, the comparison and analysis of these strategies has become a significant problem, which helps choose a best suitable search algorithm for a game playing program. Some phenomena also emerge in computer game playing and deserve to be studied. Theoretical research for computer game playing has been recognized as an important topic of artificial intelligence.

Along with the study of computer game playing, probability theory has proven to be useful in analyzing game tree search methods and in studying their related phenomena, although it sometimes gives disappointing and questionable results. For example, to test the efficiency of game tree search procedures, a standard model, which is usually a probabilistic one, is needed to represent games. Deeper searching is widely believed to increase the probability of making a correct decision in computer game playing, but the studies of some probabilistic models contradict this assumption. In efficiency analysis, the expected number of visited terminal nodes has been recognized as a standard for judging different game tree search algorithms. Probabilities can also be used to describe the strengths of game positions in game tree search. These research topics, which demonstrate the use of probability in the area of computer game playing, will be studied in this thesis.

## 1.2. Probabilistic Models for Game Trees

For computer game playing, a standard model for game trees is essential to comparing different game tree search procedures. The model should be simple enough so that it is easy to understand and set up. It should also reflect the character of real games. Most of the analyses of game tree search procedures have been based on probabilistic models. The popular model is the random uniform minimax game tree, which will be called an independent (random) game tree in this thesis. An independent game tree is defined by randomly assigning a value from a fixed set for each leaf node in a uniform tree. This model has been thoroughly studied and extensively used although it has some curious properties. For example, Pearl [2] pointed out that if each of the leaf nodes is assigned a win-loss status with properties $p_0$ and $1-p_0$, respectively, the root node is almost a sure win or a sure loss, depending on whether $p_0$ is higher or lower than some fixed probability. Another related property was described by Nau [3] as biasing: when a minimax search in an independent random game tree is done to an odd depth, all moves tend to look good for one player, and when a minimax search is done to an even depth, all moves tend to look bad for the same player. Note that these odd properties are not so prevalent in the game trees of common games like chess, checkers and GO. It is obvious that some new model for game trees is needed to eliminate, or at least reduce, these strange properties.

Since the independence between terminal node values in game trees seldom occurs in practice, some attempts have been made to introduce dependence between sibling node values into uniform trees. The simplest description for node-value dependence is encoded in the *total dependency model* [4], which has the following property for all interior nodes $g$: for any two successors $g_i$ and $g_j$ of node $g$, all of the terminal nodes following $g_i$ either have greater values than all terminal nodes following $g_j$, or they all have lesser values. But the common method of modeling dependence between node values randomly attaches weights to the branches of a uniform tree [4-6]. For the so called *branch-dependent game tree*, the sum of the weights along a path from the root node to a terminal node is assigned to the terminal node as its static value, which will be estimated by the evaluation function in practice. In some branch-weight assignments, the set of weights varies with the depth of the branches in the tree [4]; in others, this set does not [5, 6]. Here, another view of this dependence modeling approach can be provided. The values are assigned to the nodes of a uniform tree from the root towards the terminal nodes in the following way. First, the root node is assigned the value 0. After an interior node is assigned a value, say $v_i$, each branch following the node will receive a value, say $v_m$, randomly, and the successor node following the branch will be assigned the sum $v_i + v_m$. This top-down manner of assigning values to terminal nodes distinguishes the branch-dependent game trees from the independent random game tree, where the terminal nodes are randomly and independently assigned with values.

## 1.3. Minimax Pathology

Since it is usually not feasible to do a complete search of a large game tree, heuristic strategies are extensively exploited so that the game trees are only explored to some limited depth. Search-deeper strategy is a successful principle for computer game playing, and all existing game playing programs do better by increasing their search depth to improve their quality of decision-making. But the existing investigations do not support it, on the controrary, they produce negative results [6, 7]. The phenomenon that the deeper we search, the worse we play is usually called *minimax pathology*. Some attempts have been made to explain it. For example, to show the reduction in the error probability by look-ahead, Beal assumed a clustering effect in a uniform game tree that a fraction of the nodes at any level were grouped into families, and all members of a family had the same value [8]. As pointed out by Pearl [1, p. 349], the impact of these perfect correlated node clusters could be inferred from the study of "traps." He ascribed the absence of minimax pathology in common game playing to the fact that common games do not possess uniform structure but are riddled with early terminal positions. To verify his conclusion, independent random game trees were modified by assuming each node had a nonzero probability $q$ of being a real "terminal," or *trap*. In fact, no matter how small $q$ is, the percentage of the terminal nodes eliminated from the uniform tree by the trap assumption, which is

$$(qn^d + (1-q)n \times qn^{d-1} + \cdots + (1-q)^d n^d \times qn^0)/n^d$$
$$= q \times \frac{1-(1-q)^{d+1}}{1-(1-q)}$$
$$= 1-(1-q)^{d+1},$$

where $n$ and $d$ are the degree and height of the uniform tree, respectively, tends to be 100% when the height $d$ increases. The above formula is derived as follows. For the root node, which has a probability of $q$ of being a trap, an expected number $qn^d$ of terminal nodes are eliminated. The expected number of nodes at depth one, which are not eliminated by the "trap" root, is $(1-q)n$, and each such node has about $qn^{d-1}$ terminal nodes to be eliminated by the trap assumption. Therefore, the expected number of terminal nodes that follow the traps at depth one is $(1-q)n \times qn^{d-1}$. By induction, the expected number of terminal nodes that are traps at depth $d$ is $(1-q)^d n^d \times qn^0$. Summing up the above derived numbers of the terminal nodes that are eliminated by applying the trap assumption to different levels, we can get the total number of the terminal nodes eliminated from a uniform tree of depth $d$. By the above analysis, in a sufficiently high uniform tree, the trap assumption implies that almost all the terminal nodes can be evaluated exactly. Therefore, the trap assumption implies an unusually strong visibility improvement, which was denied by Pearl himself as an explanation for the search deeper benefit in common game playing. It is obvious that if the percentage of exactly evaluated terminal nodes increases with the increase of the depth of uniform trees, the evaluation function will make fewer errors when searching deeper. Drawing heavily on this kind of evaluation improvement, partial benefit of searching-deeper was derived by Pearl [1].

## 1.4. Analysis of Alpha-Beta Pruning

The alpha-beta algorithm is a game tree search procedure commonly used by computer game playing programs. It is equivalent to a depth-first brute-force game tree search in the sense that each chooses the same move as the best one when searching the same game tree. Brute-force means that all the nodes in the game tree must be visited in a depth-first traversal. But the alpha-beta algorithm prunes subtrees off the game tree by knowing that they cannot lead to a better position. The efficiency of alpha-beta pruning has been extensively studied for different models. These models can be classified into two categories: uniform trees having independent terminal node values and uniform trees having branch-dependent terminal node values. Variations of the first category include the model studied by Knuth and Moore [4], where all the terminal node values are different from each other and independent of each other. The game tree studied by Newborn [9] belongs to the second category, where the $n$ moves available at an interior node are assigned with the values $1, \cdots, n$, respectively. It is interesting to note that for different models, the analysis of the alpha-beta algorithm presents different results. Since the existence of value dependence between related positions in games like chess, checkers and GO is acknowledged by almost every author in this area, the analysis for the models that incorporate branch-value dependence into game trees could lead to more reliable results than for the game trees in the first category. This incorporation would also introduce extra difficulty in the analyses, which was exhibited by the work of Newborn [9].

## 1.5. Using Probability as the Evaluation Result

Another application of probability is in the representation of evaluation results. More appropriate representation methods have been developed to account for the inaccuracy in the evaluation. One such method is the range representation of evaluation proposed by Berliner [10]. The upper and lower bounds of a range for a node is the optimistic and pessimistic estimate of its strength, respectively. In Berliner's B* algorithm, a node is pruned when its optimistic estimate is proved to be lower than the pessimistic estimates of all contending alternatives. Palay [11] proposed using distributions, rather than ranges, to quantify the uncertainty in estimating the strength of game positions. He also improved the B* algorithm with a distribution-based algorithm.

## 2. Terminology

To study the behavior of a computer game playing program, the values that are assigned to the nodes of a game tree by the rules of the game must be differentiated from the values assigned to these nodes by the evaluation function employed by the program. The former values are called *game* values or *merit* values, and are represented by a function

$$\Phi: V \rightarrow D,$$

and the latter, called the *utilities*,[3] and represented by

$$\Psi: V \rightarrow D,$$

where $V$ is the set of nodes in the game tree, and $D$ is the set of possible values for the positions. In fact, these two functions are often the extensions of partial functions

$$\phi: V \rightarrow D,$$

and

3. In the literature of computer game playing, the terms merit value, utility, strength and score often mean the same thing — heuristic evaluation result, all correspond to the notion of utility here.

$$\psi: V \rightarrow D,$$

respectively. The function $\phi$ is defined for the leaf nodes in a game tree determined by the game rules. With chess as an example, the leaf nodes can be in one of three statuses — win, loss and draw. If we specify $D$ as the integers between -1000 and 1000, the status of a terminal position $g$ might be described by $\phi(g) = 999$, -999 or 0. Another choice of $D$, which will often be used in this thesis, is $D = \{1, -1, 0\}$, with $\phi(g) = 1, -1$, and 0 to represent the status of win, loss and draw, respectively. After the partial function $\phi$ is determined for all the leaf nodes in a game tree, under the assumption that both players play their best, it can be extended to the function

$$\Phi: V \rightarrow D.$$

Since usually a complete game tree cannot be exhausted by a computer game playing program, the partial function $\psi$ is defined for the nodes on the search frontier of the game tree rather than for the leaf nodes. After the utilities of the search frontier nodes are evaluated, the function $\psi$ can be extended to the function $\Psi$ for their antecedents,

$$\Psi: V \rightarrow D.$$

Note that the correlation between $\Psi$ and $\Phi$ determines the quality of the computer game playing program. It will be seen that the evaluation error probability is measured with respect to the discrepancy between the two functions.

The extension from function $\phi$ to $\Phi$, or from $\psi$ to $\Psi$, can be described by two different methods, which are called *minimax* and *negamax*, respectively. The two descriptions are equivalent in that they will propose the same move as the best choice for the same game position. But sometimes one method may simplify some concepts more than the other. In Chapters 2 through 4, a minimax description is exploited; while negamax description is used in Chapter 5.

In the minimax description, the game value $\phi(g)$ for a leaf node $g$ is determined by the game rules from the point of view of one player, for example, Max. For a leaf node

$g$, the game value $\Phi(g)$ is determined by

$$\Phi(g) = \phi(g).$$

For an interior node $g$, the game value $\Phi(g)$ is determined according to the successors' values and the player who will make a move at the position. More specifically, if $g$ is a Max node

$$\Phi(g) = \max\{\Phi(g_i) \mid g_i \text{ is a child of } g\};$$

if $g$ is a Min node

$$\Phi(g) = \min\{\Phi(g_i) \mid g_i \text{ is a child of } g\}.$$

In the minimax description, given the evaluation results $\psi(g)$ for terminal nodes $g$, which are their utilities, the function $\Psi(g)$ can be determined similarly: if $g$ is a terminal node,

$$\Psi(g) = \psi(g);$$

if $g$ is an interior Max node, the utility $\Psi(g)$ is defined as the maximum of its successors' values,

$$\Psi(g) = \max\{\Psi(g_i) \mid g_i \text{ is a child of } g\};$$

if $g$ is an interior Min node,

$$\Psi(g) = \min\{\Psi(g_i) \mid g_i \text{ is a child of } g\}.$$

Usually, the determination of the function $\Psi(g)$ from function $\psi(g)$ is a part of the work of the computer game playing program, which is called the ***minimax back-up***.

In the negamax description, the game value $\phi(g)$ for a leaf node $g$ is determined by the game rules from the point of view of the player who makes a move at the corresponding position. For a leaf node $g$, we have

$$\Phi(g) = \phi(g).$$

For an interior node $g$, the game value $\Phi(g)$ is determined according to the successors' values by first negating these values, and then taking the maximum. This process can be described as

$$\Phi(g) = \max\{ -\Phi(g_i) \mid g_i \text{ is a child of } g \}.$$

In negamax description, given the evaluation results, utilities $\psi(g)$, for search frontier nodes $g$, the function $\Psi(g)$ can be determined by the *negamax back-up* process in the following way. If $g$ is a terminal node,

$$\Psi(g) = \psi(g).$$

If $g$ is an interior node, the utility $\Psi(g)$ is defined as the maximum of the negated successors' utilities,

$$\Psi(g) = \max\{-\Psi(g_i) \mid g_i \text{ is a child of } g \}.$$

Bi-valued game trees are often used in the literature for studying game tree search. In a bi-valued game tree, each leaf node $g$ is assigned a game value $\phi(g)$, which is either win or loss, that is $\phi(g) \in \{\text{win, loss}\}$. For a node $g$, we will use the propositions $\Phi(g) = \text{win}$, win($g$) and $\Phi(g) = 1$ to describe the same status of node $g$, the propositions $\Phi(g) = \text{loss}$, loss($g$) and $\Phi(g) = 0$ also have equal meaning.

The notation $g_i$ has been used to represent the $i$ th child of node $g$ in the above paragraphs. Another notation for representing the successors of a node $g$ is the "Dewey decimal system" [4]: The root node corresponds to the empty integer sequence, and the $i$ th child of a node $g$, which is represented by a series of nonnegative integers, are represented by $g.i$. Therefore, if $g$ is a node, both $g_i$ and $g.i$ may be used to indicate its $i$ th successor.

Probability theory concepts will be used to describe the properties of game trees. The notation $\Pr[E]$ is used to represent the probability of event $E$. The notation $\Pr[E \mid C]$ represents the probability for event $E$ given condition $C$. The notation $P_g(v)$ is used as an abbreviation for the probability $\Pr[\Phi(g) = v]$ for a node $g$.

## 3. Outline of the Presentation

The next chapter provides the motivation for introducing node-value dependence into the model of game trees. A new model is described along with some of its special properties. Hence, the chapter also contains a discussion of, and comparison with, previous work in game tree modeling.

The focal point of Chapter 3 is the minimax search pathology. First, the problem of whether or not minimax search benefits decision-making is studied for bi-valued evaluations. By benefit we mean that searching deeper will back up a more reliable value than the static evaluation. A discussion based on different measures of node-value dependence and error probability is presented. Then, mathematical formulas are developed for computing the probabilities of making a correct decision when searching to different depths in a game tree with a "piece-counting" evaluation function. The numerical results are used to illustrate the relation between minimax search pathology and node-value dependence.

Chapter 4 analyzes the performance of the alpha-beta pruning algorithm when node-value dependence is taken into account. The analysis is exemplified with bi-valued binary game trees. Recursive equations for the average number of visited terminal nodes are derived and solved. The effect of node-value dependence on the pruning efficiency is discussed. The method for determining the average number of visited terminal nodes in a multi-valued $n$-ary game tree is also described there.

In Chapter 5, an efficient search scheme, depth-first traversal, is introduced into probability-based game tree search, where the evaluation function returns probabilities for the merit value of a node. It is shown how $\alpha$-$\beta$ bounded windows are used to cut off some subtrees from search. This generalized algorithm inherits some good properties from its point-value version. Several variations of this algorithm are also presented.

Simulations are used there to show that the probability-based alpha-beta pruning algorithm can be exploited to effectively prune the search of some subtrees.

Finally, this thesis concludes with a summary of its contributions and describes some possible future work.

**Chapter 2**

# Node-Dependent Minimax Game Trees

## 1. Game Tree Modeling

The idea of studying simplified examples is characteristic of most basic research in artificial intelligence. Game trees are useful in describing many kinds of decision-making situations. One of the motivations behind studying games, or their associated game trees, is that they simulate the decisions and th[illegible]lversary. For this reason game trees have been a subject of considerable investigation [illegible] in artificial intelligence and in decision analysis.

Making a decision on a game tree involves searching the tree to compute the utility values for the nodes of the tree. Since the number of nodes in the tree usually grows exponentially with the depth of the tree, it is not feasible to do a complete search of a large game tree. Almost all game playing programs use variants of the look-ahead minimax heuristic. Involved in this process are two major computational efforts: generating a reasonable part of the game tree and evaluating the nodes at the frontier. Once the type of static evaluation function to be used for the frontier nodes has been determined, the search effort is directly proportional to the number of frontier nodes that are generated and evaluated. That number has become a standard measure of the complexity of game tree search procedures [1].

There is a variety of real games. Since it is impractical to test every game tree search procedure against a real game or its game tree, standard models for game trees should be used to compare game tree search methods. A few probabilistic models for game trees have been proposed to measure the complexity of different game tree search procedures. In this section, a commonly used model is described. Parallel to this description, a new probabilistic model for game trees will be introduced in the next section.

The simple but commonly used model for game trees is the *independent* (random uniform) game tree, which was called random uniform tree in the literature [1].

*Definition* 2.1. A tree in which

(a) all interior nodes have exactly $n$ successors, and

(b) all bottom positions (or *leaf nodes*) are at depth $d$

is called a *uniform* tree of *degree n* and *depth d*.

A uniform tree which satisfies the additional condition

(c) the merit values assigned to all leaf nodes are independent identically distributed random variables

is called a *node-independent random uniform game tree*, or simply, *independent* (random uniform) game tree. $\square$

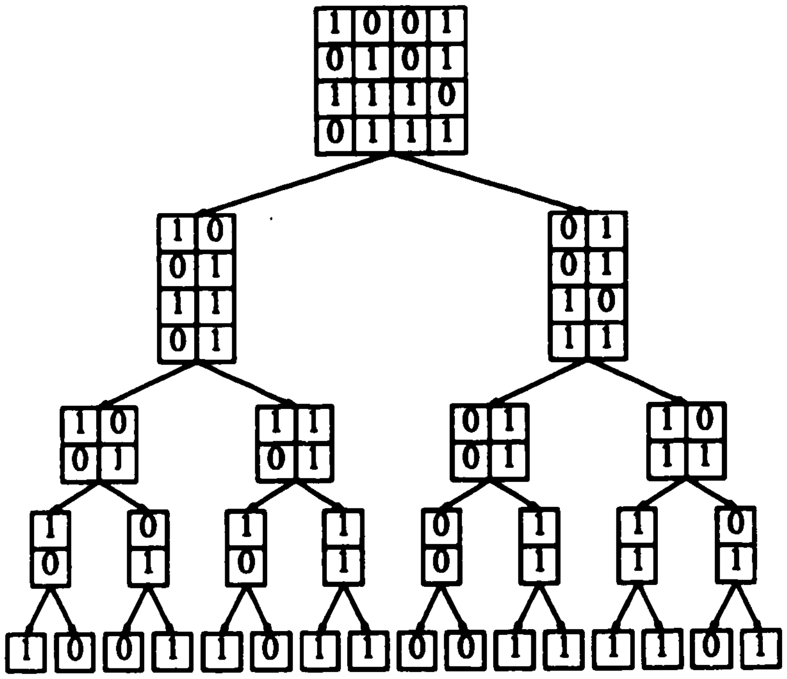
The independent game tree is used to represent a class of board-splitting games, which are two-player zero-sum perfect-information games [6]. The *board-splitting* games are played on a chess board measuring $n^{\lfloor d/2 \rfloor}$ by $n^{\lceil d/2 \rceil}$, rather than 8 by 8, for some integer $d > 0$. The initial configuration of the playing board for a game is constructed by assigning each square of the board a random value from a fixed set, for example, $\{0, 1\}$. Two players move in strict alternation. A move for the first player consists of dividing the board into $n$ equal pieces vertically, and choosing one. A move for the second player consists of dividing what is left of the board into $n$ pieces horizontally, and choosing one piece. The play continues in this manner until only one square is left. The value of this final square is the score for the player who made the first move in the game.[4]

4. In some of the literature [6], the value is interpreted as the score for the player who made the

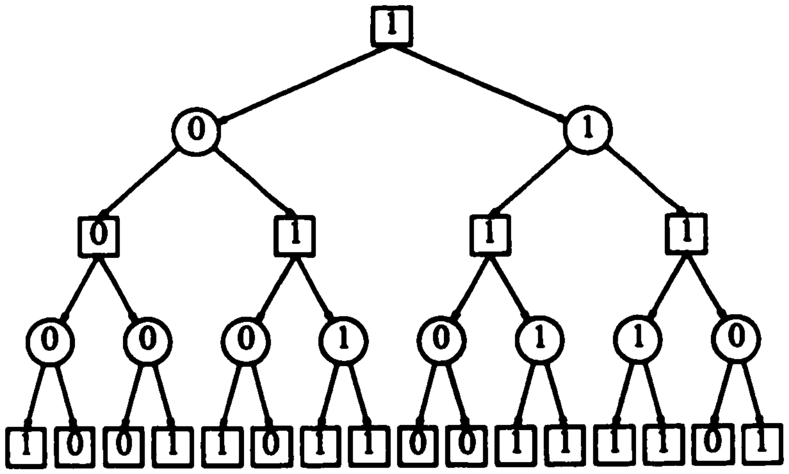
For the node value set {0, 1}, values 0 and 1 mean loss and win for the first player, respectively. Every game in the class takes *d* moves to play no matter what moves the players choose. In the game tree of such a board-splitting game, the root node represents the initial configuration of the complete board, each interior node has *n* successors, which correspond to the *n* smaller pieces, and the leaf nodes correspond to the board squares. Fig. 2.1(a) illustrates a board splitting game with $n = 2$ and $d = 4$. The corresponding game tree is depicted in Fig. 2.1(b). It is easy to set up the one-one correspondence between the squares on the playing board and the leaf nodes in the game tree. In the following, all the board-splitting games are illustrated with their game trees.

Given an instance of an independent game tree, by the correspondence between the squares on the playing board and the leaf nodes in the uniform tree, we can figure out the initial configuration of the playing board for the corresponding board-splitting game.

last move. This interpretation facilitates some recursive calculation, and will be used in Chapter 3.



(a)



(b)

Fig. 2.1 A Board-Splitting Game and Its Game Tree

The player who makes the first move in the above described games is named Max, and his opponent, Min. Assuming the two players play their best, since Max's strategy is to lead the game towards squares with higher values, while Min's strategy is to lead the play towards terminal positions with lower values, each node of the game tree can be assigned a merit value based on the merit values of the leaf nodes by the minimax process. Since the merit value backed-up to a node by the minimax process depends only on the backed-up values of its successors, it is immediately observed that, by condition (c), the backed-up values of all nodes at the *same* depth in an independent game tree are also independent identically distributed random variables. This is the reason that these game trees are called node-independent game trees.[5]

As observed by Nau[6], in games such as chess or checkers, positions are often characterized as "strong" or "weak". Strong positions are likely to be win nodes, and are likely to have higher utility values then. Since board positions change incrementally, the merit values of sibling nodes are likely to be similar. But in independent random game trees, the merit values of sibling nodes are independent of each other. To eliminate this diversity between the independent game trees and the game trees of common games such as chess and checkers, some proposals have been presented in the literature. As indicated in Section 1 of Chapter 1, these methods first randomly decide branch values, then make use of these values to generate the leaf node values. In the next section, a new model for game trees is presented, which introduces node-value dependence in a different way.

## 2. A New Model for Game Trees

To model the dependence among the merit values that sibling positions in a game can take, conditional probabilities for the merit values of sibling nodes will be introduced

5. Since in this thesis only the dependence and independence between node values, rather than branch values, are emphasized, the descriptive word "node" is often omitted.

into uniform trees. These probabilities are conditioned on the parent's values.

*Definition* 2.2. A uniform game tree of degree $n$ and depth $d$ that satisfies the following additional conditions

(d) the root node is randomly assigned a merit value $v$ from some fixed set $S$ with some probability distribution $P_0(v)$ for $v \in S$, and

(e) for an interior Max node $g$, the successors' merit values $\Phi(g_i)$ $(1 \le i \le n)$ are randomly determined by a set of conditional probabilities

$$\Pr[\Phi(g_1)=v_1, \Phi(g_2)=v_2, \cdots, \Phi(g_n)=v_n \mid \Phi(g)=v_0], \tag{2.1}$$

where $v_i \in S$ for $0 \le i \le n$, and

(f) for an interior Min node $g$, the successors' merit values $\Phi(g_i)$ $(1 \le i \le n)$ are randomly determined by a set of conditional probabilities

$$\Pr[\Phi(g_1)=v'_1, \Phi(g_2)=v'_2, ..., \Phi(g_n)=v'_n \mid \Phi(g)=v'_0], \tag{2.2}$$

where $v'_i \in S$ for $0 \le i \le n$

is called a ***node-dependent uniform minimax game tree***, or simply, dependent (minimax) game tree. □

Informally speaking, the definition of dependent game tree simulates the process of game designing. We can imagine that to design a game, first, the initial configuration of the game should be designed, and then, a set of rules should be determined to transform a position into other positions. The condition (d) describes the initial configuration and corresponds to the initial configuration design; the conditional probabilities in (e) and (f) transforms the probabilities for the possible merit values of a position to the probabilities for the merit values of the successors, and "describe" the set of game rules. Therefore, the dependent game tree determines the merit values for the nodes in it in a top-down manner, which was exploited by the branch-dependent game tree. In fact, definition 2.2 can be regarded as an improvement of the branch-dependent game tree. As described in Section 1 of Chapter 1, to determine the leaf merit values in the branch-dependent game tree, the value for a node is the sum of its parent value and the value of the branch leading to it, which is determined randomly. Here, the randomness is replaced by the conditional probabilities for the successors' values.

In a dependent minimax game tree, the merit values of leaf nodes are no longer independent of each other. The probability distributions for these values are determined by the probability distribution of the root value, and the conditional probabilities specified in conditions (e) and (f). In addition to completeness, the sets of conditional probabilities (2.1) and (2.2) must satisfy some consistency requirements, which are specific to game trees and described as follows. By the definition of minimax, a Max node $g$ has a merit value $\Phi(g) = v_0$ if and only if all its successors have merit values less than or equal to $v_0$ and at least one successor's value is $v_0$. Therefore, the conditional probabilities in (2.1) must satisfy the following requirements:

$$v_i \neq v_0 \text{ for all } 0 \leq i \leq n \Rightarrow \Pr[\Phi(g_1)=v_1, \Phi(g_2)=v_2, \ldots, \Phi(g_n)=v_n \mid \Phi(g)=v_0] = 0,$$

$$v_i > v_0 \text{ for some } 0 \leq i \leq n \Rightarrow \Pr[\Phi(g_1)=v_1, \Phi(g_2)=v_2, \ldots, \Phi(g_n)=v_n \mid \Phi(g)=v_0] = 0,$$

where the symbol => means propositional implication. The conditional probabilities in (2.2) must satisfy some similar propositions.

The dependent minimax game trees are also related to a class of two-player zero-sum perfect-information board-splitting games. These games have the same playing rule as the games described in Section 1; but the initial configuration [illegible] in a different way. The values of the board squares are determined by the probability distribution for the root node merit value and the conditional probabilitie [illegible] i) a [illegible] ) in the following way. First, randomly assign the root node of the unifor [illegible] h merit value $v \in S$ with probability distribution $P_0(v)$. As soon as the merit value $\Phi(g)$ of an interior node $g$ is determined, the merit values of its successors are randomly selected according to the merit value $\Phi(g)$ and the conditional probabilities imposed by condition (e) when $g$ is a Max node, or by condition (f) when $g$ is a Min node. Based on the one-to-one correspondence between the leaf nodes of the uniform tree and the squares of the playing board, each square of the game board can be assigned a unique value from $S$.

The dependent game trees have some interesting properties, which are missed by the model of an independent game tree. For the sake of simplicity, the degree $n$ will be fixed to 2, the merit value set $S = \{0, 1\}$, and the final value 0 means loss for the first player and 1 means win. It will be assumed $d = 2k$ for some integer $k \geq 0$. In other words, bi-valued binary uniform game trees of even heights will be used to reveal some properties of the dependent game trees. For these game trees, the probability $P_0(1)$ specified in condition (d) of definition 2.2 will be denoted as $p_0$. For the bi-valued binary dependent game trees, some conditional probabilities are trivial:

for a Max node $g$

$$\Pr[\Phi(g_1)=0, \Phi(g_2)=0 \mid \Phi(g)=0] = 1;$$

for a Min node $g$

$$\Pr[\Phi(g_1)=1, \Phi(g_2)=1 \mid \Phi(g)=1] = 1.$$

In addition to these conditional probabilities, the conditions (e) and (f) for a bi-valued binary dependent game tree will be specified by

(e') For an interior win Max node $g$, the two successors' merit values $\Phi(g_1)$ and $\Phi(g_2)$ are randomly determined by the conditional probabilities

$$\Pr[\Phi(g_1)=1, \Phi(g_2)=1 \mid \Phi(g)=1] = f_1,$$
$$\Pr[\Phi(g_1)=1, \Phi(g_2)=0 \mid \Phi(g)=1] = \frac{1}{2}(1-f_1),$$
$$\Pr[\Phi(g_1)=0, \Phi(g_2)=1 \mid \Phi(g)=1] = \frac{1}{2}(1-f_1)$$

with $0 \le f_1 \le 1$

(f') For an interior loss Min node $g$, the two successors' merit values $\Phi(g_1)$ and $\Phi(g_2)$ are randomly determined by the conditional probabilities

$$\Pr[\Phi(g_1)=0, \Phi(g_2)=0 \mid \Phi(g)=0] = f_2,$$
$$\Pr[\Phi(g_1)=1, \Phi(g_2)=0 \mid \Phi(g)=0] = \frac{1}{2}(1-f_2),$$
$$\Pr[\Phi(g_1)=0, \Phi(g_2)=1 \mid \Phi(g)=0] = \frac{1}{2}(1-f_2)$$

with $0 \le f_2 \le 1$.

The parameters $f_1$ and $f_2$, presented in the above definition, describe the dependences of successor values and are called *dependent factors*. It is obvious that both successors $g_1$ and $g_2$ of a node $g$ take the value 1 (or 0) with the same probability. As a result, all the leaf nodes in a dependent game tree have the same probability to receive merit value 1 (or 0). This fact implies that the merit values assigned to the leaf positions are identically distributed random variables. An instance of a dependent minimax game tree can be generated with three parameters, $p_0$, $f_1$ and $f_2$. Note that $p_0$ is actually the probability that the player Max wins the game. In Fig. 2.2, two instances of a dependent minimax game tree are shown, which were generated with parameters $p_0 = 0.5$, $f_1 = 0.4$ and $f_2 = 0.3$. The root node merit value of the game tree in Fig. 2.2(a) is 0, and the root node of the game tree in Fig. 2.2(b) has a merit value 1.

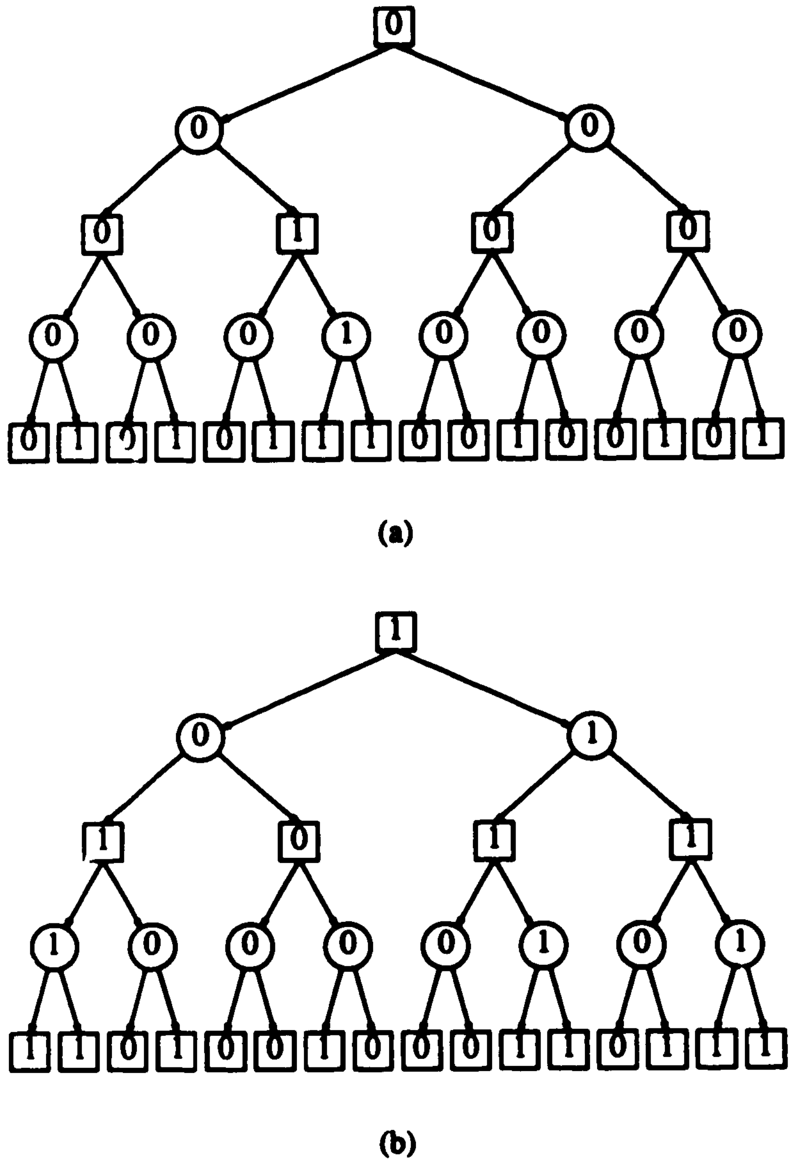


Fig. 2.2 Two Game Trees Generated with Parameters $p_0 = 0.5, f_1 = 0.4, f_2 = 0.3$

A common consensus on the dependence between sibling node merit values in game trees is that if a parent has a high merit value, then all its successors tend to have high merit values. As quoted earlier, Nau [6] indicated that in games such as chess or checkers, the utility value of a node is usually positively correlated with the utility value of its parent. Since the utility value is an approximation to the merit value, the merit values of sibling nodes, the merit values of parent nodes and successor nodes are likely to be similar. For the dependent minimax game trees, suppose a Max node $g$ has merit value 1. Then at least one of its successors, say $g_1$, must have merit value 1. For this case, the statement that all successors tend to have high merit values can be modeled with the requirement that node $g_2$ is more likely to have merit value 1 than 0. Similarly, if a Min node $g$ has a merit value 0, one of its successor must have the merit value 0, and the property that all successors likely have similar merit values is reflected by the requirement that the other successor tends to have merit value 0. These requirements are described by the following concept.

*Definition* 2.3. For a dependent minimax game tree, if the conditional probabilities in (e') satisfy the condition

$$\Pr[\Phi(g_2)=1 \mid \Phi(g)=1, \Phi(g_1)=1] > \Pr[\Phi(g_2)=0 \mid \Phi(g)=1, \Phi(g_1)=1], \qquad (2.3)$$

the Min nodes in this game tree are *positively correlated*. Similarly, if the conditional probabilities in (f') satisfy the condition

$$\Pr[\Phi(g_2)=0 \mid \Phi(g)=0, \Phi(g_1)=0] > \Pr[\Phi(g_2)=1 \mid \Phi(g)=0, \Phi(g_1)=0], \qquad (2.4)$$

the Max nodes in this game tree are *positively correlated*. If both the Max and Min nodes are positively correlated, this game tree is said to be *positively correlated*. □

Since for Max nodes $g$, we have

$$\Pr[\Phi(g_1)=1 \mid \Phi(g)=1, \Phi(g_2)=1] = \Pr[\Phi(g_2)=1 \mid \Phi(g)=1, \Phi(g_1)=1],$$

$$\Pr[\Phi(g_1)=0 \mid \Phi(g)=1, \Phi(g_2)=1] = \Pr[\Phi(g_2)=0 \mid \Phi(g)=1, \Phi(g_1)=1],$$

and for Min nodes $g$,

$$\Pr[\Phi(g_1)=0 \mid \Phi(g)=0, \Phi(g_2)=0] = \Pr[\Phi(g_2)=0 \mid \Phi(g)=0, \Phi(g_1)=0],$$

$$\Pr[\Phi(g_1)=1 \mid \Phi(g)=0, \Phi(g_2)=0] = \Pr[\Phi(g_2)=1 \mid \Phi(g)=0, \Phi(g_1)=0],$$

the conditions (2.3) and (2.4) are equivalent to

$$\Pr[\Phi(g_1)=1 \mid \Phi(g)=1, \Phi(g_2)=1] > \Pr[\Phi(g_1)=0 \mid \Phi(g)=1, \Phi(g_2)=1],$$

and

$$\Pr[\Phi(g_1)=0 \mid \Phi(g)=0, \Phi(g_2)=0] > \Pr[\Phi(g_1)=1 \mid \Phi(g)=0, \Phi(g_2)=0],$$

respectively. As shown by the following lemma, the property of positive correlation is determined by the dependent factors $f_1$ and $f_2$.

*Lemma* 2.1 The Max (Min) nodes in a dependent minimax game tree are positively correlated if and only if the dependent factor $f_2 > \frac{1}{3}$ (or, $f_1 > \frac{1}{3}$, respectively).

*Proof.*

Let $g$ be a Min node. Note that

$$\begin{aligned}
&\Pr[\Phi(g_2)=0 \mid \Phi(g)=0] \\
&\quad = \Pr[\Phi(g_1)=0, \Phi(g_2)=0 \mid \Phi(g)=0] + \Pr[\Phi(g_1)=1, \Phi(g_2)=0 \mid \Phi(g)=0] \\
&\quad = f_2 + \frac{1}{2}(1 - f_2) \\
&\quad = \frac{1}{2}(1 + f_2).
\end{aligned}$$

We have

$$\begin{aligned}
&\Pr[\Phi(g_1)=0 \mid \Phi(g)=0, \Phi(g_2)=0] \\
&\quad = \frac{\Pr[\Phi(g_1)=0, \Phi(g_2)=0 \mid \Phi(g)=0]}{\Pr[\Phi(g_2)=0 \mid \Phi(g)=0]} \\
&\quad = \frac{2f_2}{1+f_2},
\end{aligned}$$

and

$$\Pr[\Phi(g_1)=1 \mid \Phi(g)=0, \Phi(g_2)=0]$$

$$= 1 - \Pr[\Phi(g_1)=0 \mid \Phi(g)=0, \Phi(g_2)=0]$$
$$= \frac{1-f_2}{1+f_2}.$$

Therefore,

$$\Pr[\Phi(g_2)=0 \mid \Phi(g)=0, \Phi(g_1)=0] > \Pr[\Phi(g_2)=1 \mid \Phi(g)=0, \Phi(g_1)=0],$$

if and only if

$$\frac{2f_2}{1+f_2} > \frac{1-f_2}{1+f_2},$$

which is equivalent to

$$f_2 > \frac{1}{3}.$$

Similarly, it can also be proven that the Min nodes are positively correlated in the dependent minimax tree if and only if

$$f_1 > \frac{1}{3}.$$

$\square$

By the proof of Lemma 2.1, it can be seen that the degree of positive correlation in dependent game trees can be manipulated by adjusting their dependent factors $f_1$ and $f_2$.

Before continuing the discussion of the properties of dependent game trees, we point out that the node-value dependences expressed in conditions (e') and (f') can be regarded as a restriction on the following more general dependences:

(e'') For an interior win Max node $g$, the two successors' merit values $\Phi(g_1)$ and $\Phi(g_2)$ are randomly determined by the conditional probabilities

$$\Pr[\Phi(g_1)=1, \Phi(g_2)=1 \mid \Phi(g)=1] = f_1,$$
$$\Pr[\Phi(g_1)=1, \Phi(g_2)=0 \mid \Phi(g)=1] = z_1$$
$$\Pr[\Phi(g_1)=0, \Phi(g_2)=1 \mid \Phi(g)=1] = 1 - f_1 - z_1$$

with $f_1 + z_1 \leq 1$

(f'') For an interior loss Min node $g$, the two successors' merit values $\Phi(g_1)$ and $\Phi(g_2)$ are randomly determined by the conditional probabilities

$$\Pr[\Phi(g_1)=0, \Phi(g_2)=0 \mid \Phi(g)=0] = f_2,$$
$$\Pr[\Phi(g_1)=1, \Phi(g_2)=0 \mid \Phi(g)=0] = z_2,$$
$$\Pr[\Phi(g_1)=0, \Phi(g_2)=1 \mid \Phi(g)=0] = 1 - f_2 - z_2$$

with $f_2 + z_2 \leq 1$.

By specifying $f_1, f_2, z_1$ and $z_2$ that satisfy, for example, $z_1 > 1 - f_1 - z_1$ and $z_2 > 1 - f_2 - z_2$, *ordered* dependent game trees can be constructed. By ordered game tree we mean that in a depth-first traversal, the best move for the player Max is likely to be visited before the worst one. These game trees could be used to verify the relation between the efficiency of a game tree search procedure and the measure of node ordering presented in a game tree. For the sake of simplisity, we will focus on the binary dependent game trees that satisfy conditions (e') and (f') rather than (e'') and (f'').

## 3. Probabilities for Terminal Win Positions

In a dependent game tree, the probability for a node to take the merit value 1 is determined by the three parameters, $p_0$, $f_1$ and $f_2$ of the game tree. Since this probability is determined recursively in a top-down manner, a formula for it can be derived. The quantity to be computed is $P_k$, the probability that player Max can force a win at a node at depth $2k$ in a dependent minimax game tree with parameters $p_0$, $f_1$ and $f_2$. Let $Q_k$ denote the probability that Max can force a win at a node at depth $2k + 1$. $P_k$ and $Q_k$ are calculated prior to constructing any instance of the game tree. By condition (e') presented in Section 2, if a Max node $g$ is assigned merit value 1, the probability that a successor is assigned merit value 1 is

$$f_1 + \frac{1}{2}(1 - f_1) = \frac{1}{2}(1 + f_1);$$

if $g$ is assigned 0, the probability that a successor is assigned merit value 1 is 0. Therefore, we have

$$Q_k = \frac{1}{2}(1+f_1)P_k. \tag{2.5}$$

If a Min node is assigned merit value 1, the probability that a successor is assigned merit value 1 is 1; if the Min node is assigned 0, the probability that a successor is assigned merit value 1 is $\frac{1}{2}(1-f_2)$. Therefore, we have

$$P_{k+1} = Q_k + \frac{1}{2}(1-f_2)(1-Q_k) = \frac{1}{2}(1+f_2)Q_k + \frac{1}{2}(1-f_2) \tag{2.6}$$

By (2.5) and (2.6), the recurrence equation for $P_k$ can be derived:

$$P_{k+1} = \frac{1}{4}(1+f_1)(1+f_2)P_k + \frac{1}{2}(1-f_2) \tag{2.7}$$

for $k \geq 0$, where $P_0 = p_0$ is the probability that the root node is assigned merit value 1.

The equation (2.7) can be solved by cases. If $f_1 = f_2 = 1$, by (2.7), we have $P_k = p_0$ for any $k \geq 0$. Therefore, we will assume $f_1 + f_2 < 2$ in the following. For the sake of notation simplicity, denote

$$x = \frac{1}{4}(1+f_1)(1+f_2) \text{ and } y = \frac{1}{2}(1-f_2).$$

For $k \geq 0$, equation (2.7) can be solved by

$$\begin{aligned} P_{k+1} &= xP_k + y \\ &= x^2P_{k-1} + xy + y \\ &= \cdots \\ &= x^{k+1}P_0 + x^ky + x^{k-1}y + \cdots + y \\ &= p_0x^{k+1} + \frac{y(1-x^{k+1})}{1-x}. \end{aligned}$$

In other words, the probability that player Max can force a win at a node of depth $2k$ is

$$P_k = p_0(\frac{1}{4}(1+f_1)(1+f_2))^k + \frac{1}{2}(1-f_2)\frac{1-(\frac{1}{4}(1+f_1)(1+f_2))^k}{1-\frac{1}{4}(1+f_1)(1+f_2)}, \tag{2.8}$$

if not both $f_1$ and $f_2$ are equal to 1.

Formula (2.8) describes the probability for the leaf nodes in a dependent game tree of height $2k$ to take merit value 1, which is the function of parameters $p_0$, $f_1$ and $f_2$, where $f_1 + f_2 < 2$. A property of the dependent minimax game trees can immediately be derived from this formula.

*Lemma* 2.2. For any $0 < p_0 < 1$ and $k \geq 0$, there are dependent factors $f_1$ and $f_2$, $0 < f_1 < 1$, $0 < f_2 < 1$, such that the probability $P_k$ that a leaf node in a dependent game tree of height $2k$ with parameters $p_0$, $f_1$ and $f_2$ is assigned value 1 is equal to $p_0$.

*Proof.*

In fact, the values of dependent factors $f_1$ and $f_2$ that satisfy this lemma can be obtained by solving the equation

$$p_0 = p_0(\frac{1}{4}(1+f_1)(1+f_2))^k + \frac{1}{2}(1-f_2)\frac{1-(\frac{1}{4}(1+f_1)(1+f_2))^k}{1-\frac{1}{4}(1+f_1)(1+f_2)},$$

which is equivalent to

$$p_0 = \frac{\frac{1}{2}(1-f_2)}{1-\frac{1}{4}(1+f_1)(1+f_2)}.$$

For any value $f_2$ with $0 < f_2 < 1$ and value $p_0$ with

$$\frac{\frac{1}{2}(1-f_2)}{1-\frac{1}{4}(1+0)(1+f_2)} < p_0 < \frac{\frac{1}{2}(1-f_2)}{1-\frac{1}{4}(1+1)(1+f_2)} = 1,$$

since the quantity $\dfrac{\frac{1}{2}(1-f_2)}{1-\frac{1}{4}(1+f_1)(1+f_2)}$ is continuous and increasing with $f_1$, a value $f_1$ with $0 < f_1 < 1$ can be found so that

$$p_0 = \frac{\frac{1}{2}(1-f_2)}{1-\frac{1}{4}(1+f_1)(1+f_2)}.$$

Since $\lim\limits_{f_2 \to 1^-} \dfrac{\frac{1}{2}(1-f_2)}{1-\frac{1}{4}(1+f_2)} = 0$, for any $p_0$ with $0 < p_0 < 1$, values of $f_1$ and $f_2$ with $0 < f_1 < 1$ and $0 < f_2 < 1$ can be found which satisfy

$$p_0 = \frac{\frac{1}{2}(1-f_2)}{1-\frac{1}{4}(1+f_1)(1+f_2)}. \qquad \square$$

As an example with probability $p_0 = 1/3$, $1/2$ and $(\sqrt{5} - 1)/2$, some combinations of $f_1$ and $f_2$ for which a leaf node at depth 2 has merit value 1 with probability $p_0$ are shown in Table 2.1. In Fig. 2.3, the corresponding curves for which both the leaf nodes and the root node have merit value 1 with the same probability $p_0$ are drawn on $(f_1, f_2)$-plane. Each point of the curves corresponds to a combination of $f_1$ and $f_2$.

**Table 2.1** Some Combinations of $f_1$ and $f_2$ for $P_k = p_0$

| $p_0 = 0.333$ $f_1$ | $f_2$ | $p_0 = 0.500$ $f_1$ | $f_2$ | $p_0 = 0.618$ $f_1$ | $f_2$ |
|---|---|---|---|---|---|
| 0.000 | 0.600 | 0.000 | 0.333 | 0.055 | 0.133 |
| 0.102 | 0.633 | 0.143 | 0.400 | 0.176 | 0.200 |
| 0.200 | 0.667 | 0.273 | 0.467 | 0.284 | 0.267 |
| 0.294 | 0.700 | 0.391 | 0.533 | 0.382 | 0.333 |
| 0.385 | 0.733 | 0.500 | 0.600 | 0.624 | 0.533 |
| 0.472 | 0.767 | 0.551 | 0.633 | 0.691 | 0.600 |
| 0.556 | 0.800 | 0.647 | 0.700 | 0.753 | 0.667 |
| 0.636 | 0.833 | 0.736 | 0.767 | 0.810 | 0.733 |
| 0.714 | 0.867 | 0.818 | 0.833 | 0.863 | 0.800 |
| 0.789 | 0.900 | 0.895 | 0.900 | 0.912 | 0.867 |
| 0.862 | 0.933 | 0.931 | 0.933 | 0.957 | 0.933 |
| 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |

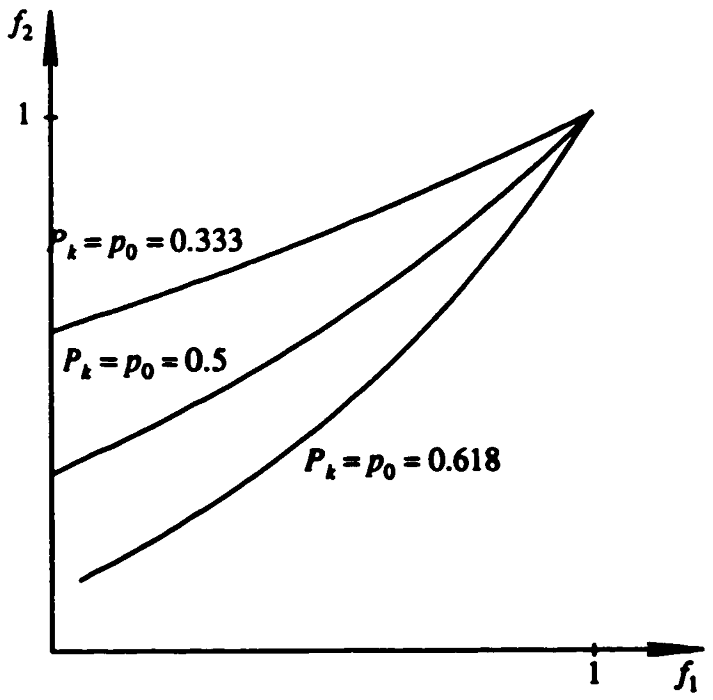


Fig. 2.3 The Curves for $P_k = p_0 = \frac{1}{3}, \frac{1}{2}$ and $\frac{\sqrt{5}-1}{2}$ on $f_1$-$f_2$-Plane

## 4. The Average Number of Terminal Win Positions

The average number of win leaf nodes, in a dependent minimax game tree of height $2k$ for some integer $k > 0$ is determined not only by the probability $p_0$, which is the probability that the player Max wins the game, but also by the dependent factors $f_1$ and $f_2$. Define $R_k$ as the average number of win nodes at depth $2k$, and $T_k$ as the average number of win nodes at depth $2k + 1$. Note that the average number of loss nodes at depth $2k$ is $2^{2k} - R_k$, and the average number of loss nodes at depth $2k + 1$ is $2^{2k+1} - T_k$. Since the nodes at depth $2k$ are Max nodes, each win node has an average number of

$$2(f_1 + \frac{1}{2}(1-f_1)) = 1 + f_1$$

win successors, and each loss node cannot have any win successors, we have

$$T_k = R_k(1 + f_1).$$

For a win node at depth $2k + 1$, which is a Min node, both of its two successors must be win nodes, and a loss node has $2\times\frac{1}{2}(1-f_2) = 1 - f_2$ win successors on average. Therefore, if there are $T_k$ win nodes at depth $2k + 1$, the average number of win nodes at depth $2k + 2$ is

$$R_{k+1} = 2T_k + (2^{2k+1} - T_k)(1 - f_2) = T_k(1 + f_2) + 2^{2k+1}(1 - f_2).$$

The recursive equation for $R_k$ is

$$R_{k+1} = R_k(1 + f_1)(1 + f_2) + 2^{2k+1}(1 - f_2).$$

Since $R_0 = p_0$, the solution for this recursive equation is

$$R_k = p_0((1+f_1)(1+f_2))^k + 2^{2k-1}(1-f_2)\frac{1 - (\frac{1}{4}(1+f_1)(1+f_2))^k}{1 - \frac{1}{4}(1+f_1)(1+f_2)}, \tag{2.9}$$

when $f_1 + f_2 < 2$. If $f_1 = f_2 = 1$, it is easy to deduce that $P_k = 4^k p_0$.

In fact, the solution $R_k$ in (2.9) can also be derived from the solution for $P_k$ in (2.8). Since $P_k$ is the probability that a node at depth $2k$ is a win node in a dependent minimax game tree and there are $2^{2k}$ nodes at depth $2k$ in a binary uniform tree, the average number of win leaf nodes in a dependent minimax game tree of height $2k$ is

$$2^{2k}P_k = p_0((1+f_1)(1+f_2))^k + 2^{2k-1}(1-f_2)\frac{1-(\frac{1}{4}(1+f_1)(1+f_2))^k}{1-\frac{1}{4}(1+f_1)(1+f_2)},$$

when $f_1 + f_2 < 2$.

By formula (2.9), the average number of win leaf nodes can be changed while keeping the probability for the player Max to win the game unchanged. Because of this property, two dependent game trees can be constructed, which correspond to two different combinations of $p_0$, $f_1$ and $f_2$, so that the first game tree has more chances for the player Max to win, but the second is likely to have more win leaf nodes in a game tree. The first game tree can be generated with parameters $p_0 = 0.618$, $f_1 = 0.4$, $f_2 = 0.6$; the second game tree corresponds to $p_0 = 0.5$, $f_1 = 0.6$, $f_2 = 0.4$. The data shown in Table 2.2 can be used to illustrate this phenomenon. Although, for parameters $p_0 = 0.618$, $f_1 = 0.4$, $f_2 = 0.6$, the root node of a game tree has more chances to be assigned merit value 1, it is possible that there are fewer win leaf nodes in this tree than in the game tree of the same height that has parameters $p_0 = 0.5$, $f_1 = 0.6$, $f_2 = 0.4$. This phenomenon can be related to the fact that the initial configuration design and the rule design are indeed two separate tasks for a game design. We can imagine that the game designer has a predefined probability for a player to win the game. The probability may be 0.5 if the game is to be fair, or a little greater than 0.5 if the advantage of making the first move is taken into account. In the rule design, different rules will generate different outcomes. These outcomes should reflect the relation between a position and its successors. This relation corresponds to the values of $f_1$ and $f_2$, which are the "specifications" for the successor merit values when the merit value of the position is given. Note that the number $R_1$ of win leaf nodes in a dependent game tree of height 2 with parameters $p_0 = 0.5$, $f_1 = 0.6$,

$f_2 = 0.4$ is less than that number for the dependent game tree with $p_0 = 0.618$, $f_1 = 0.4$, $f_2 = 0.6$. This observation implies that the probability $p_0$ has more effect on merit values of the shallower nodes, the conditional probabilities $f_1$ and $f_2$ have more effect on that of the deeper nodes.

Table 2.2 The Average Numbers of Terminal Win Nodes

| $p_0$ $f_1$ $f_2$ | $R_1$ | $R_2$ | $R_3$ | $R_4$ | $R_5$ | $R_6$ | $R_7$ | $R_8$ |
|---|---|---|---|---|---|---|---|---|
| 0.500 0.6 0.4 | 1.720 | 6.253 | 23.606 | 91.278 | 358.063 | 1416.461 | 5630.472 | 22442.657 |
| 0.618 0.4 0.6 | 1.879 | 6.186 | 21.767 | 80.399 | 306.661 | 1193.187 | 4697.800 | 18623.322 |

## 5. Comparison with Other Dependence Descriptions

As noticed by Knuth and Moore [4], if the evaluation function was based on the piece count in a chess game, all the positions following a blunder would tend to have low scores for the player who lost more pieces. They proposed the *total dependency model* to account for such dependences. In this model, for an interior node $g$ and any two of its successors $g_i$ and $g_j$, all of the leaf nodes in the subtree rooted at $g_i$ either have greater merit values than all leaf nodes in the subtree rooted at $g_j$, or they all have lesser merit values. The situation that one move decides all of the relative strengths of the terminal positions is also rarely seen. A more accurate description should be that if a position $g$ is stronger than another position $g'$, the positions following $g$ are *probably* stronger than those following $g'$. In terms of binary game trees, for Max nodes $g$ and $g'$, if the probability $p_0$ for the first node to have merit value 1 is greater than the probability $p'_0$ for $g'$, then the successors of the former are more likely to have merit value 1 than the successors of the latter. In fact, the probability for a successor $g_i$ of $g$ to have merit value 1 is

$\frac{1}{2}(1+f_1)p_0$, which is greater than the probability for a successor $g'_j$ of $g'$ to have merit value 1, which is $\frac{1}{2}(1+f_1)p'_0$. A similar conclusion can be made for Min nodes. By this observation, it can be seen that dependent game tree can be regarded as a less restricted version of the total dependency model.

Another method of modeling dependence between node merit values randomly attaches values to the branches, sums up the branch values along a path from the root node to a leaf node, and assigns the leaf node with this sum. For example, in an $n$-ary uniform tree, Knuth and Moore [4] proposed to assign to each of the $n$ branches directed from an interior node at ply $k$ a distinct value from the set $\{1/n^k, 2/n^k, ..., n/n^k\}$, and Fuller *et al.* [5] proposed to assign $\{1, 2, ..., n\}$ to the branches directed from a node. This method simulates game playing tactics, but it is very difficult to exploit this model in analyzing game tree search procedures. Only a few papers have addressed these branch-dependent game trees. The difficulty is related to the fact that the leaf node merit values are the algebraic summations of the branch values along the paths from the root node to the leaf nodes, while the merit values $\Phi(g)$ for nodes $g$ in the tree are backed up with minimax process. This incompatibility makes recursive analysis of branch-dependent game trees very difficult, if not impossible. As shown by the results of Newborn [9], the performance of alpha-beta pruning is difficult to be derived for the trees of depth four.

## 6. Comparison with Independent Game Trees

Most of the analyses of game tree search procedures have been based on independent game trees or their variations. Some of them have distinct leaf node merit values [4], some allow ties between terminal positions [12]. They also have been extensively exploited in theoretical research on computer game playing [6,7]. These trees have attracted a lot of attention because of their regularity and simplicity. But these characteristics also cause some problems. First, it is impossible to create a game tree of arbitrary height where the root node has reasonable probabilities for the different merit values. Let $P^*$ be the solution of equation

$$x^2 + x - 1 = 0$$

in the range $0 < x < 1$. In fact, this number dictates the behavior of binary independent random uniform trees [2, 3]. For the bi-valued binary independent game trees, if the probability $p'_0$ that each leaf node receives merit value 1 is slightly greater than $P^*$, Max is almost assured a win if the tree is long enough; whereas the player faces an almost sure loss when the probability is less than $P^*$. For example, for a tree of height 10, if $p'_0 = 0.5$, the chance that player Max wins a game, which is the probability that the root node in the independent game tree takes merit value 1, is only 1.95%; when $p'_0 = 0.7$, the chance increases to 98.5% [2]. Therefore, the analysis of game tree search procedures has to be tested against the independent game trees that have $p'_0 = P^*$. This probability implies that the root node is assigned merit value 1 with the probability $P^*$. Thus, prior to playing the game, each player is assured a predefined probability, and this probability is biased against one player. In real games like chess and checkers, no player is very concerned with being the second player. The chance for the first player, who starts the game, to win the game is generally conceived as about 0.5, or slightly larger. Therefore, to generate more convincing test data for game tree search procedures, the probability that the root node is assigned merit value 1 should be something around 0.5. As shown above, this is very difficult to implement in an independent game tree. But this is not a problem for the dependent game trees at all, since the probability for the root node to take merit value 1 does not depend on the successors.

As shown in Lemma 2.2, for each probability $p_0$, an infinite number of combinations of values for the dependent factors $f_1$ and $f_2$ can be found so that in the dependent game trees with these dependent factors, both the root node and the leaf nodes have the same probability, $p_0$, to be assigned with merit value 1. Based on the observation that the probability for a player to win a mid-game should not be much different from the probability that the player wins the initial game, the probability for the root node to take merit value 1 should be equal to that for an interior node to take that merit value in an ideal game tree model. As shown by Lemma 2.2, the node-dependent game trees can attain this property by setting the dependent factors $f_1$ and $f_2$, which do not depend on their heights. This property can not be simulated by independent random uniform trees, unless $p_0 = (\sqrt{5} - 1)/2$. Therefore, dependent game trees provide a flexible model for different games.

As pointed out in several papers [4, 6], lack of dependence between the sibling node merit values is rarely seen in real games. Therefore, if a game tree search procedure is tested against independent random uniform trees, the reliability of the testing result is doubtful for common games. As to be shown in Chapter 4, the dependence between sibling node merit values and between parent and successor node merit values, which is reflected by different settings of $f_1$ and $f_2$, does affect the pruning efficiency. Since the existence of this sort of dependence in common games is almost universally accepted, the relation between the dependence and the pruning efficiency deserves to be studied, and the model of dependent game tree provides a new basis for testing game tree search procedures.

It should also be pointed out that the flexibility of setting dependent factors $f_1$ and $f_2$ might reflect the great variety of games. In addition to their different initial configurations, the degrees of position dependence are important characters of these games. No one believes that making a wrong move in some games is usually as serious as loosing the queen in chess. In chess, usually the strengths of all the positions following the loss of queen is severely reduced. Therefore, different games would have different degrees of dependence, and different setting of dependent factors could be used to model them.

## 7. Generality of the New Model

Although only bi-valued binary trees have been used so far to illustrate the properties of this new model of game trees, the results obtained in this chapter can be generalized for multi-valued $n$-ary trees. For example, given the probability distribution $P_0(v)$ and the two sets of conditional probabilities which are defined in the conditions (d), (e) and (f) for a node-dependent game tree (cf. Definition 2.2 in Section 2), a set of recursive equations for the probabilities $P_k(v)$ that are the probabilities for a node at depth $2k$ to take merit values $v \in S$ can be established. To make all the nodes at the same depth have the same probability to take a merit value $v \in S$, we assume that if a vector $(v_1, v_2, \cdots, v_n)$ is a permutation of another vector $(w_1, w_2, \cdots, w_n)$, then the conditional probabilities specified by (2.1) satisfy

$$\begin{aligned}&\Pr[\Phi(g_1){=}v_1, \Phi(g_2){=}v_2, \cdots, \Phi(g_n){=}v_n \mid \Phi(g){=}v_0],\\ =\ &\Pr[\Phi(g_1){=}w_1, \Phi(g_2){=}w_2, \cdots, \Phi(g_n){=}w_n \mid \Phi(g){=}v_0],\end{aligned}$$

for any node $g$ and domain value $v_0 \in S$, and the conditional probabilities specified by (2.2) satisfy a similar requirement. As in Section 3, we can derive a set of equations

$$P_k(v) = \sum_{i \in S} a_{vi} P_{k-1}(i) + b_v, \text{ for } v \in S,$$

where $a_{vi}$ and $b_v$ are constants that are determined by the domain value $v$ and the conditional probabilities defined in conditions (e) and (f). Let matrix $\mathbf{A} = (a_{vi})$, and column vector $\mathbf{B} = (b_v)$. The above equations can be represented by

$$\mathbf{P}_{k+1} = \mathbf{A}\mathbf{P}_k + \mathbf{B},$$

where column vector $\mathbf{P}_k = (P_k(v))$. Since this linear recursive equation can be solved if the matrix $(\mathbf{I} - \mathbf{A})$ is non-singular, the formulas for $P_k(v)$ can be derived, which is a function of integer $k$, probability distribution function $P_0(v)$ and the conditional probabilities defined in (e) and (f). By these solutions, the properties for the given multi-valued $n$-ary node-dependent minimax game tree, which are similar to those presented in Sections 3 and 4, can be derived. Therefore, the node-dependent minimax game tree is a general model for game trees.

Chapter 3

# An Investigation of Minimax Pathology

## 1. Minimax Pathology and Node-Value Dependence

Heuristic look-ahead search is a successful technique for computer game playing, and there is a universal agreement that increasing the depth of the search improves the quality of the decision [13, 14]. However, the investigation presented in the literature [6, 7, 15] showed that, given a certain theoretical model of the errors made by an evaluation function, there exists an infinite class of game trees which are *pathological* in the sense that as long as the search does not reach the end of the game (in which case, a correct decision is guaranteed), searching deeper will not increase the probability that a correct decision is made, but will instead cause the decision to become increasingly random. By assuming a uniform game tree model, Beal [15] demonstrated that the probability of error would not be reduced with minimax backing-up, and fixed-depth backed-up values were less trustworthy than the static values. A class of board-splitting games which are called Pearl's games were used by Nau [6] to demonstrate the existence of minimax search pathology. By computation, it is shown that Pearl's games are pathological when the obvious evaluation function is used. When minimax backing-up is exploited in independent uniform game trees, Pearl [7] showed that the minimax process introduces a spurious noise, which may cause the pathology phenomenon: the deeper search degrades the quality of the decision. The conclusion in the literature is that minimax backing up cannot reduce the errors presented in static evaluation, and several explanations have been proposed for the successful minimax search in chess or other game playing.

It is notable that almost all the existing results on minimax pathology have been derived under the condition that sibling nodes in a game tree are independent of each other.[6] The probabilistic model of dependent minimax game tree provides a new basis for observing the minimax process. In this chapter, the problem of whether or not minimax search benefits decision making is studied for this model. First, the minimax search error is investigated for bi-valued evaluation functions with dependent game trees. The cases with extremal values of dependent factors are studied with respect to decision making. Mathematical formulas are developed for computing the probabilities of making a correct decision when searching to various depths in the games with a piece-counting evaluation function. By a piece we mean a 1-square in the corresponding position. The numerical results obtained from these formulas are used to show the effect of node-value dependence on minimax pathology.

In this investigation, it is essential to distinguish a search tree, the explored part of a game tree, from the game tree itself. In fact, we are going to study the relationship between the evaluation results for the terminal nodes of the search tree, which are defined as utilities, and the merit values for the leaf nodes in the game tree, which represent the terminal positions of the game and can be evaluated accurately. Therefore, the terms terminal and leaf nodes, search tree and game tree have different connctations in this chapter.

We will use a class of board-splitting games, which are called node-dependent games, or D-games. These D-games have the same playing rules as Pearl's games and Nau's games, which were described in Section 1 of Chapter 2, but their game trees are node-dependent, independent and branch-dependent game trees, respectively. To make

6. An exception is Nau's game [6], which has a branch-dependent game tree. But no mathematics formula has been established to describe the behavior of the minimax search with respect to making correct decision. Only Monte Carlo simulation is presented there.

the recursive calculation of some probabilities easy and independent of the height of game trees, we will assume that the player who makes the last move is the player Max.

## 2. Benefits of Minimax Search

The problem of whether or not a deeper search increases the probability of making a correct decision is studied with a bi-valued evaluation function here.

### 2.1. Error-Propagation Patterns

Assume $G$ is a bi-valued binary dependent minimax game tree with dependent factors $f_1$ and $f_2$, and imagine $G$ as a search tree. Each terminal node $g$ in the search tree is assigned a utility by the bi-valued evaluation function, with $\psi(g) = 1$ or $\psi(g) = 0$ to indicate a strong or weak position. Based on the fact that when determining the utility of a position, most evaluation functions employed by computer game playing programs do not examine the features of other positions, we assume that the utility of a node $g$ is independent of the merit value of the sibling node of $g$ when the merit value of $g$ is given.

Evaluation errors are made when value $\psi(g) = 1$ is assigned to a terminal node $g$ which is in reality a loss, $\Phi(g) = 0$, and vice versa, a value of $\psi(g') = 0$ to a win node $g'$. The utilities of nodes at higher levels in the search tree $G$ are calculated by minimax process. In this way, the evaluation function actually assigns a unique value, which is either $\Psi(g) = 1$ or $\Psi(g) = 0$, to each node $g$ in the search tree. This value would constitute somewhat unreliable prediction on the merit value of the position.

Shown in Fig. 3.1 is a subtree of $G$, where the root node $g$ is a Max node at depth $2(k-1)$. For convenience, the nodes $g$, $g.1$, $g.2$, $g.1.1$, $g.1.2$, $g.2.1$ and $g.2.2$ are called node 1 through node 7, respectively, the symbol $\Psi_i$ is used to denote the utility of node $i$ for $1 \le i \le 7$, and $\Phi_i$ denote the merit value of node $i$ for $1 \le i \le 7$. Using these notations, the propagations for both $\Phi$ and $\Psi$ follow the minimax back-up. More specifically, we have

$$\Phi_1 = \begin{cases} \text{win} & \text{if } \Phi_2 = \text{win or } \Phi_3 = \text{win} \\ \text{loss} & \text{if } \Phi_2 = \text{loss and } \Phi_3 = \text{loss}; \end{cases}$$

$$\Phi_2 = \begin{cases} \text{win} & \text{if } \Phi_4 = \text{win and } \Phi_5 = \text{win} \\ \text{loss} & \text{if } \Phi_4 = \text{loss or } \Phi_5 = \text{loss}; \end{cases}$$

$$\Phi_3 = \begin{cases} \text{win} & \text{if } \Phi_6 = \text{win and } \Phi_7 = \text{win} \\ \text{loss} & \text{if } \Phi_6 = \text{loss or } \Phi_7 = \text{loss}; \end{cases}$$

$$\Psi_1 = \begin{cases} 1 & \text{if } \Psi_2 = 1 \text{ or } \Psi_3 = 1 \\ 0 & \text{if } \Psi_2 = 0 \text{ and } \Psi_3 = 0; \end{cases}$$

$$\Psi_2 = \begin{cases} 1 & \text{if } \Psi_4 = 1 \text{ and } \Psi_5 = 1 \\ 0 & \text{if } \Psi_4 = 0 \text{ or } \Psi_5 = 0; \end{cases}$$

$$\Psi_3 = \begin{cases} 1 & \text{if } \Psi_6 = 1 \text{ and } \Psi_7 = 1 \\ 0 & \text{if } \Psi_6 = 0 \text{ or } \Psi_7 = 0. \end{cases}$$

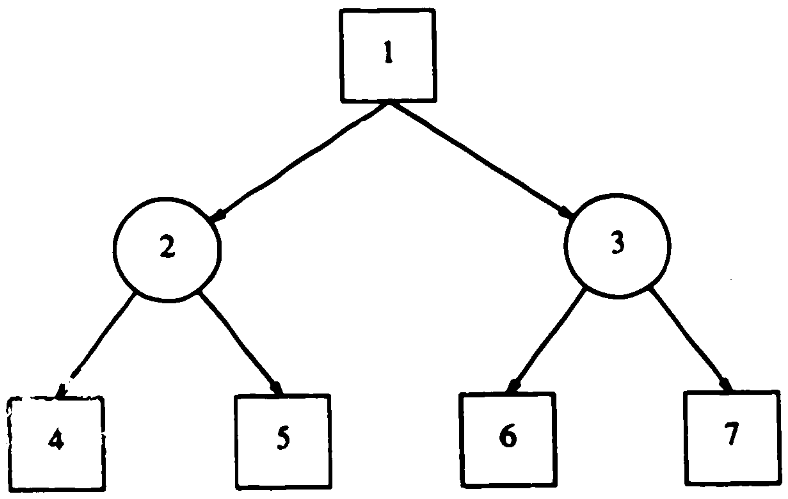


**Fig.** 3.1 A Subtree of Height 2

Here, we assume that the search frontier is not higher than node 4. The informedness of the evaluation function $\Psi$ at a node $i$ is quantified with two parame :

$$\alpha_i = \Pr[\Psi_i = 1 \mid \Phi_i = \text{loss}]$$

$$\beta_i = \Pr[\Psi_i = 0 \mid \Phi_i = \text{win}]$$

for $i = 1, \cdots, 7$. Since $\Phi_2$ and $\Phi_3$ ($\Phi_4$, $\Phi_5$, $\Phi_6$, and $\Phi_7$) are identically distributed random variables, it is reasonable to assume $\alpha_2 = \alpha_3$, $\beta_2 = \beta_3$ ($\alpha_4 = \alpha_5 = \alpha_6 = \alpha_7$, $\beta_4 = \beta_5 = \beta_6 = \beta_7$). The dependent factors $f_1$ and $f_2$ are involved in calculating error propagation. Here, error propagation means the calculation of $\alpha_1$ and $\beta_1$ given $\alpha_4$ and $\beta_4$. But first, $\alpha_2$ and $\beta_2$ will be calculated. In the calculation of error propagation, we make use of the equation

$$\Pr[\Psi_i = v_1 \mid \Phi_i = v_2, \Phi_j = v_3] = \Pr[\Psi_i = v_1 \mid \Phi_i = v_2],$$

where nodes $i$ and $j$ are sibling nodes like nodes 4 and 5, and the values $v_1$, $v_2$, and $v_3 \in \{0, 1\}$. The above equation is derived by

$$\Pr[\Psi_i = v_1 \mid \Phi_i = v_2, \Phi_j = v_3]$$

$$= \frac{\Pr[\Psi_i = v_1, \Phi_i = v_2, \Phi_j = v_3]}{\Pr[\Phi_i = v_2, \Phi_j = v_3]}$$

$$= \frac{\Pr[\Psi_i = v_1, \Phi_j = v_3 \mid \Phi_i = v_2]\Pr[\Phi_i = v_2]}{\Pr[\Phi_i = v_2, \Phi_j = v_3]}$$

$$= \frac{\Pr[\Psi_i = v_1 \mid \Phi_i = v_2]\Pr[\Phi_j = v_3 \mid \Phi_i = v_2]\Pr[\Phi_i = v_2]}{\Pr[\Phi_i = v_2, \Phi_j = v_3]}$$

$$= \Pr[\Psi_i = v_1 \mid \Phi_i = v_2].$$

Since node 2 is a Min node, by the propagation rules for $\Phi$ and $\Psi$ and the definition of dependent factor $f_2$,

$$\alpha_2 = \Pr[\Psi_2 = 1 \mid \Phi_2 = \text{loss}]$$

$$= \Pr[\Psi_4 = \Psi_5 = 1 \mid \Phi_2 = \text{loss}]$$

$$= \Pr[\Psi_4 = \Psi_5 = 1 \mid \Phi_4 = \Phi_5 = \text{loss}]\Pr[\Phi_4 = \Phi_5 = \text{loss} \mid \Phi_2 = \text{loss}]$$

$$+ \Pr[\Psi_4 = \Psi_5 = 1 \mid \Phi_4 = \text{win}, \Phi_5 = \text{loss}]\Pr[\Phi_4 = \text{win}, \Phi_5 = \text{loss} \mid \Phi_2 = \text{loss}]$$

$$+ \Pr[\Psi_4 = \Psi_5 = 1 \mid \Phi_4 = \text{loss}, \Phi_5 = \text{win}]\Pr[\Phi_4 = \text{loss}, \Phi_5 = \text{win} \mid \Phi_2 = \text{loss}]$$

$$= \Pr[\Psi_4 = 1 \mid \Phi_4 = \text{loss}]\Pr[\Psi_5 = 1 \mid \Phi_5 = \text{loss}]f_2$$

$$+ \Pr[\Psi_4 = 1 \mid \Phi_4 = \text{win}]\Pr[\Psi_5 = 1 \mid \Phi_5 = \text{loss}]\frac{1}{2}(1 - f_2)$$

$$+ \Pr[\Psi_4 = 1 \mid \Phi_4 = \text{loss}]\Pr[\Psi_5 = 1 \mid \Phi_5 = \text{win}]\frac{1}{2}(1 - f_2)$$

$$= \alpha_4^2 f_2 + \alpha_4(1 - \beta_4)(1 - f_2),$$

$$\beta_2 = \Pr[\Psi_2 = 0 \mid \Phi_2 = \text{win}]$$

$$= \Pr[\Psi_4 = \Psi_5 = 0 \mid \Phi_2 = \text{win}]$$

$$+ \Pr[\Psi_4 = 1, \Psi_5 = 0 \mid \Phi_2 = \text{win}]$$

$$+ \Pr[\Psi_4 = 0, \Psi_5 = 1 \mid \Phi_2 = \text{win}]$$

$$= \Pr[\Psi_4 = \Psi_5 = 0 \mid \Phi_4 = \Phi_5 = \text{win}]$$

$$+ \Pr[\Psi_4 = 1, \Psi_5 = 0 \mid \Phi_4 = \Phi_5 = \text{win}]$$

$$+ \Pr[\Psi_4 = 0, \Psi_5 = 1 \mid \Phi_4 = \Phi_5 = \text{win}]$$

$$= \Pr[\Psi_4 = 0 \mid \Phi_4 = \text{win}]\Pr[\Psi_5 = 0 \mid \Phi_5 = \text{win}]$$
$$+ \Pr[\Psi_4 = 1 \mid \Phi_4 = \text{win}]\Pr[\Psi_5 = 0 \mid \Phi_5 = \text{win}]$$
$$+ \Pr[\Psi_4 = 0 \mid \Phi_4 = \text{win}]\Pr[\Psi_5 = 1 \mid \Phi_5 = \text{win}]$$
$$= \beta_4^2 + 2\beta_4(1-\beta_4).$$

Since node 1 is a Max node,

$$\alpha_1 = \Pr[\Psi_1 = 1 \mid \Phi_1 = \text{loss}]$$
$$= \Pr[\Psi_2 = \Psi_3 = 1 \mid \Phi_1 = \text{loss}]$$
$$+ \Pr[\Psi_2 = 1, \Psi_3 = 0 \mid \Phi_1 = \text{loss}]$$
$$+ \Pr[\Psi_2 = 0, \Psi_3 = 1 \mid \Phi_1 = \text{loss}]$$
$$= \Pr[\Psi_2 = \Psi_3 = 1 \mid \Phi_2 = \Phi_3 = \text{loss}]$$
$$+ \Pr[\Psi_2 = 1, \Psi_3 = 0 \mid \Phi_2 = \Phi_3 = \text{loss}]$$
$$+ \Pr[\Psi_2 = 0, \Psi_3 = 1 \mid \Phi_2 = \Phi_3 = \text{loss}]$$
$$= \Pr[\Psi_2 = 1 \mid \Phi_2 = \text{loss}]\Pr[\Psi_3 = 1 \mid \Phi_3 = \text{loss}]$$
$$+ \Pr[\Psi_2 = 1 \mid \Phi_2 = \text{loss}]\Pr[\Psi_3 = 0 \mid \Phi_3 = \text{loss}]$$
$$+ \Pr[\Psi_2 = 0 \mid \Phi_2 = \text{loss}]\Pr[\Psi_3 = 1 \mid \Phi_3 = \text{loss}]$$
$$= \alpha_2^2 + 2\alpha_2(1-\alpha_2)$$
$$= \left[\alpha_4^2 f_2 + \alpha_4(1-\beta_4)(1-f_2)\right]^2$$
$$+ 2\left[\alpha_4^2 f_2 + \alpha_4(1-\beta_4)(1-f_2)\right]\left[1 - \alpha_4^2 f_2 - \alpha_4(1-\beta_4)(1-f_2)\right],$$

and

$$\beta_1 = \Pr[\Psi_1 = 0 \mid \Phi_1 = \text{win}]$$
$$= \Pr[\Psi_2 = \Psi_3 = 0 \mid \Phi_1 = \text{win}]$$
$$= \Pr[\Psi_2 = \Psi_3 = 0 \mid \Phi_2 = \Phi_3 = \text{win}]\Pr[\Phi_2 = \Phi_3 = \text{win} \mid \Phi_1 = \text{win}]$$
$$+ \Pr[\Psi_2 = \Psi_3 = 0 \mid \Phi_2 = \text{win}, \Phi_3 = \text{loss}]\Pr[\Phi_2 = \text{win}, \Phi_3 = \text{loss} \mid \Phi_1 = \text{win}]$$
$$+ \Pr[\Psi_2 = \Psi_3 = 0 \mid \Phi_2 = \text{loss}, \Phi_3 = \text{win}]\Pr[\Phi_2 = \text{loss}, \Phi_3 = \text{win} \mid \Phi_1 = \text{win}]$$
$$= \Pr[\Psi_2 = 0 \mid \Phi_2 = \text{win}]\Pr[\Psi_3 = 0 \mid \Phi_3 = \text{win}]f_1$$

$$+ \Pr[\Psi_2 = 0 \mid \Phi_2 = \text{win}]\Pr[\Psi_3 = 0 \mid \Phi_3 = \text{loss}]\frac{1}{2}(1 - f_1)$$

$$+ \Pr[\Psi_2 = 0 \mid \Phi_2 = \text{loss}]\Pr[\Psi_3 = 0 \mid \Phi_3 = \text{win}]\frac{1}{2}(1 - f_1)$$

$$= \beta_2^2 f_1 + \beta_2(1 - \alpha_2)(1 - f_1)$$

$$= \left[\beta_4^2 + 2\beta_4(1 - \beta_4)\right]^2 f_1$$

$$+ \left[\beta_4^2 + 2\beta_4(1 - \beta_4)\right]\left[1 - \alpha_4^2 f_2 - \alpha_4(1 - \beta_4)(1 - f_2)\right](1 - f_1).$$

Denoting $\alpha = \alpha_4$, $\beta = \beta_4$, $\alpha' = \alpha_1$ and $\beta' = \beta_1$, the derived equations are

$$\alpha' = \left[\alpha^2 f_2 + \alpha(1 - \beta)(1 - f_2)\right]^2$$
$$+ 2\left[\alpha^2 f_2 + \alpha(1 - \beta)(1 - f_2)\right]\left[1 - \alpha^2 f_2 - \alpha(1 - \beta)(1 - f_2)\right] \tag{3.1}$$

$$\beta' = \left[\beta^2 + 2\beta(1 - \beta)\right]^2 f_1$$
$$+ \left[\beta^2 + 2\beta(1 - \beta)\right]\left[1 - \alpha^2 f_2 - \alpha(1 - \beta)(1 - f_2)\right](1 - f_1). \tag{3.2}$$

The values of $\alpha$ and $\beta$ indicate the probabilities that an error is made at depth $2k$, while $\alpha'$ and $\beta'$ reflect the error probabilities after the evaluation results are rolled back to depth $2(k-1)$. Therefore, formulas (3.1) and (3.2) indicate the evaluation error propagation pattern for a bi-valued binary dependent game tree with dependent factors $f_1$ and $f_2$.

## 2.2. Benefit Analysis of Minimax

The problem of benefit of minimax search can be described as whether the following relation holds:

$$\alpha' < \alpha, \ \beta' < \beta. \tag{3.3}$$

In fact, the answer depends on the specific values of $\alpha$, $\beta$, $f_1$ and $f_2$. For some combinations of these values, the answer is positive, for some others, the answer is negative. To establish (3.3), we need to find the values of $\alpha$, $\beta$, $f_1$ and $f_2$ so that

$$\begin{aligned}&\left[\alpha^2 f_2+\alpha(1-\beta)(1-f_2)\right]^2\\+2&\left[\alpha^2 f_2+\alpha(1-\beta)(1-f_2)\right]\left[1-\alpha^2 f_2-\alpha(1-\beta)(1-f_2)\right]<\alpha,\end{aligned} \tag{3.4}$$

$$\begin{aligned}&\left[\beta^2+2\beta(1-\beta)\right]^2 f_1\\+&\left[\beta^2+2\beta(1-\beta)\right]\left[1-\alpha^2 f_2-\alpha(1-\beta)(1-f_2)\right](1-f_1)<\beta.\end{aligned} \tag{3.5}$$

If $f_2$ is sufficiently close to 1, then $1-f_2$ can be regarded as 0, and inequality (3.4) can be reduced to

$$\alpha^4+2\alpha^2(1-\alpha^2)<\alpha,$$

or

$$\alpha^3+2\alpha(1-\alpha^2)<1.$$

It is obvious that when $\alpha$ is small, for example, $\alpha=\frac{1}{2}$, the above inequality holds. When $f_1$ is sufficiently close to 1, then $1-f_1$ can be regarded as 0, inequality (3.5) can be reduced to

$$\left[\beta^2+2\beta(1-\beta)\right]^2<\beta,$$

i.e.,

$$\beta(\beta+2(1-\beta))^2<1.$$

Since

$$\lim_{\beta\to 0}\beta(\beta+2(1-\beta))^2=0,$$

there is a $\beta>0$, for example, $\beta=\frac{1}{8}$, such that the above inequality holds. These arguments imply that for some values of $\alpha$, $\beta$, $f_1$ and $f_2$, the inequalities (3.4) and (3.5) do hold.

As a conservative assumption, let the probabilities that the evaluation function misestimates a node at depth $2(k-1)$ and a node at depth $2k$ be the same.[7] The above analysis implies that if the game tree is strongly correlated ($f_1$ and $f_2$ are sufficiently close to 1), and the estimation error measures $\alpha$ and $\beta$ are small enough, the values backed-up from depth $2k$ are more reliable than the values obtained by estimating at depth $2(k-1)$. In other words, for these values of $\alpha$, $\beta$, $f_1$ and $f_2$, pathology does not present itself in minimax search. This kind of benefit from minimax search is illustrated in Fig. 3.2, where a series of points are drawn on the $\alpha$-$\beta$-plane. The first point at (0.3, 0.2) represents the evaluation error probabilities, $\alpha = 0.3$ and $\beta = 0.2$, at depth $2k$ in a dependent minimax game tree with $f_1 = f_2 = 0.7$. The following eight points represent the backed-up errors at depths $2(k-1), \cdots, 2(k-8)$, respectively. Finally, the error disappears at depth $2(k-j)$ with $j > 14$. Therefore, if the search tree $G$ has a height $2k$ with $k > 14$, the static evaluation error incurred at the search frontier almost disappears when the evaluation results are backed up to the root node. These data were obtained by setting $f_1 = f_2 = 0.7$ and iteratively calculating $\alpha'$ and $\beta'$ with the formulas (3.1) and (3.2).

7. In common games, generally speaking, since the visibility will be increased by searching deeper, the probability that an evaluation function misestimates a node at deeper levels will be reduced.

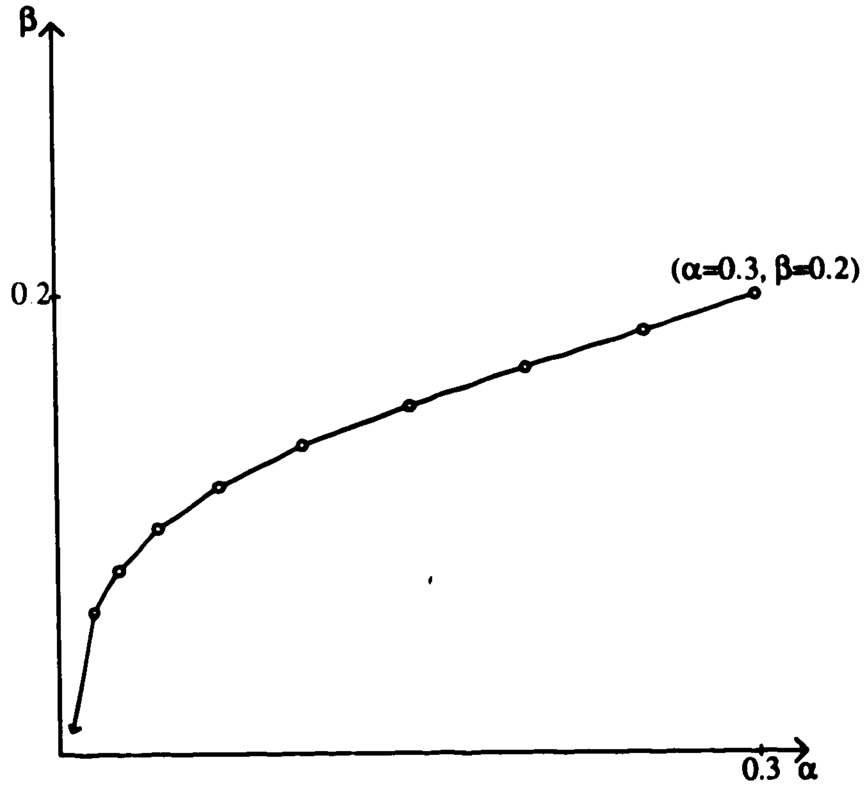


Fig. 3.2 The Benefit of Minimax Back-Up

On the other hand, some values of α, β, $f_1$ and $f_2$ can be found so that

$$\begin{aligned}&\left[\alpha^2 f_2 + \alpha(1-\beta)(1-f_2)\right]^2\\ &+2\left[\alpha^2 f_2 + \alpha(1-\beta)(1-f_2)\right]\left[1-\alpha^2 f_2-\alpha(1-\beta)(1-f_2)\right] > \alpha,\end{aligned} \tag{3.6}$$

$$\begin{aligned}&\left[\beta^2+2\beta(1-\beta)\right]^2 f_1\\ &+\left[\beta^2+2\beta(1-\beta)\right]\left[1-\alpha^2 f_2-\alpha(1-\beta)(1-f_2)\right](1-f_1) > \beta.\end{aligned} \tag{3.7}$$

In fact if $f_2$ is sufficiently close to 0, then $1-f_2$ can be regarded as 1, inequality (3.6) can be reduced to

$$\alpha^2(1-\beta)^2+2\alpha(1-\beta)(1-\alpha(1-\beta)) > \alpha,$$

which is equivalent to

$$\alpha(1-\beta)^2 + 2(1-\beta)(1-\alpha(1-\beta)) > 1.$$

Since $\lim_{\alpha \to 0} \left[\alpha(1-\beta)^2 + 2(1-\beta)(1-\alpha(1-\beta))\right] = 2(1-\beta)$, if $\beta > \frac{1}{2}$, some $\alpha$ and $f_2$ can be found so that the above inequality and the inequality (3.6) holds. If $f_1$ is sufficiently close to 0, then $1-f_1$ can be regarded as 1, along with the assumption that $f_2$ is sufficiently close to 0, inequality (3.7) can be reduced to

$$\left[\beta^2 + 2\beta(1-\beta)\right](1-\alpha(1-\beta)) > \beta,$$

which is equivalent to

$$\left[\beta + 2(1-\beta)\right](1-\alpha(1-\beta)) > 1,$$

Since when $\beta = \frac{2}{3}$, $\beta + 2(1-\beta) > 1$, a value of $\beta > \frac{1}{2}$ and a sufficiently small $\alpha$ can be found so that the above inequality holds. Therefore, there are values of $\alpha$, $\beta$, $f_1$ and $f_2$ for which both inequalities (3.6) and (3.7) hold. This conclusion implies that if the probabilities for an evaluation function to misestimate a node at depth $2(k-1)$ and a node at depth $2k$ are the same, deeper search with minimax back-up will deteriorate the evaluation quality for some D-games and evaluation functions.

In summary, it is shown that both pathology and "health" can appear in minimax searches. Whether or not pathology occurs in a game depends on both the node value dependence and the accuracy of the evaluation function. Notice that this conclusion is different from the points presented in the literature to date [6, 7, 15].

## 3. Special Cases for Multi-Valued Evaluation

Let $g$ be a node in the game tree of a board-splitting game like Pearl's game, Nau's game or D-game, and $g_1$ and $g_2$ be the successors of $g$. If $g_1$ has more 1-squares in its board configuration than $g_2$, it would seem more likely that $g_1$ is a win position[8] than

8. A win position (node) $g$ is a position (node) with $\Phi(g) = 1$.

that $g_2$ is a win position. As pointed out by Nau [6], the number of win-terminal positions, or 1-squares, contained in a game position is a reasonable estimate of the strength for the corresponding node in board-splitting games. In the remainder of this chapter, the number of 1-squares in the position of a node $g$ will be denoted by $\psi(g)$, which is the utility of the position corresponding to $g$ if $g$ is at the search frontier.

In this section, the D-games with the dependent factors $f_1$ and $f_2$ equal to 0 or 1 are studied. If $f_1 = 1$, the successors' values for a win Max node $g$ are totally correlated so that

$$\Phi(g_1) = 1 <=> \Phi(g_2) = 1,$$

where the symbol <=> is the bidirectional propositional implication; similarly, if $f_2 = 1$, the successors' values for a loss Min node $g$ are totally correlated so that

$$\Phi(g_1) = 0 <=> \Phi(g_2) = 0.$$

The extremal value $f_1 = 0$ implies that for any Max node $g$,

$$\Phi(g_1) = 1 => \Phi(g_2) = 0,$$

$$\Phi(g_2) = 1 => \Phi(g_1) = 0;$$

and $f_2 = 0$ implies that for any Min node $g$,

$$\Phi(g_1) = 0 => \Phi(g_2) = 1,$$

$$\Phi(g_2) = 0 => \Phi(g_1) = 1.$$

It will be shown that some of these extremal values do not cause minimax pathology.

When $f_1 = 1$, each interior win Max node has two win successors. Based on this fact, we prove that a win node will have more 1-squares in its configuration than a loss node at the same height $h$, where $h$ can be any natural numbers. This property implies that searching deeper in a dependent game tree with $f_1 = 1$ will not decrease the probability of making a correct decision.

*Lemma.* 3.1. For a D-game that is generated with $f_1 = 1$, the evaluation function $\psi$ can be used to choose the correct move with a search to depth $d$ for any $d \geq 0$.

*Proof.* By induction on the height $h$ of nodes in the dependent game tree, it can be proved that the utility $\psi(g)$ for win node $g$ is greater than $\psi(g')$ for a loss node $g'$ at the same height $h$, and all the win nodes at the same height have the same number of 1-squares in their corresponding configurations. When $h = 0$ or 1, this conclusion is trivially true. Suppose the conclusion is true for $h < H$, and $g$ and $g'$ are win and loss nodes at height $H$, respectively. If $g$ and $g'$ are Max nodes, since $f_1 = 1$, both successors of $g$ are win nodes; both successors of $g'$ are loss nodes. By the induction hypothesis, we have

$$\psi(g) = \psi(g_1) + \psi(g_2) > \psi(g'_1) + \psi(g'_2) = \psi(g').$$

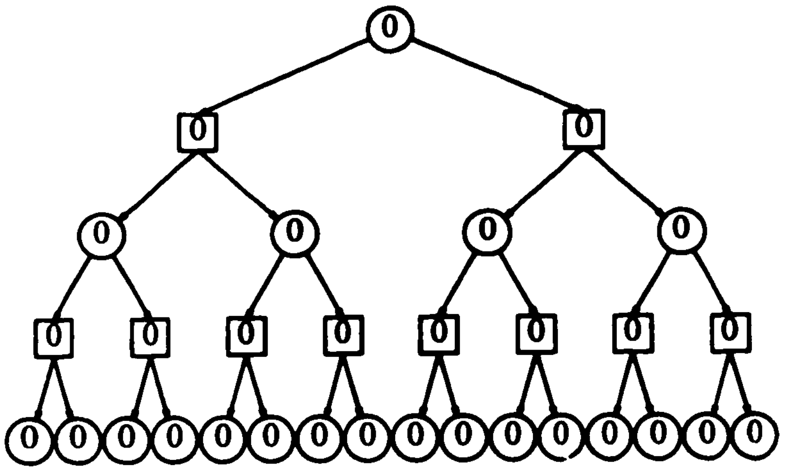
If $g$ and $g'$ are Min nodes, since $g$ is a win node and $g'$ is a loss node, both successors of $g$ are win nodes, and at least one of successors of $g'$ is a loss node. Therefore,

$$\psi(g) = \psi(g_1) + \psi(g_2) > \psi(g'_1) + \psi(g'_2) = \psi(g').$$

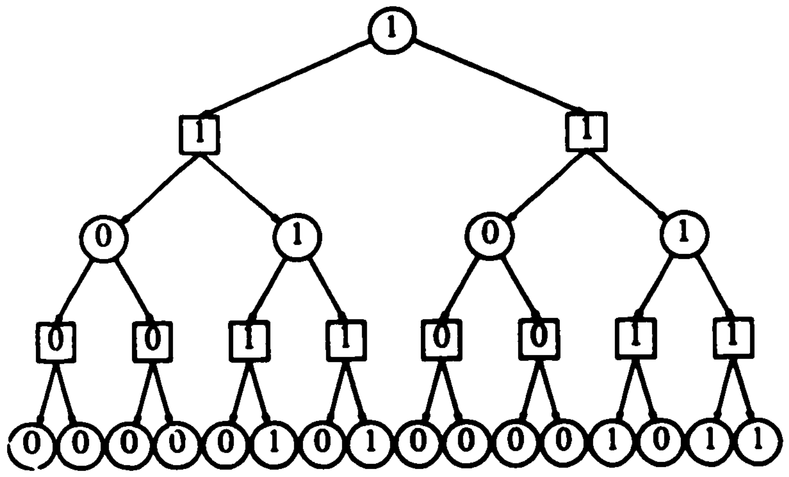
Note that, by induction hypothesis and $f_1 = 1$, all win nodes $g$ at height $H$ have the same number of 1-squares in their configurations, which is $2\psi(g_1)$.

Let the terminal nodes have height $h$, and a win node at height $h$ have $N_h$ 1-squares in its configuration. Since a loss node on the search frontier has less than $N_h$ 1-squares in its configuration, it can be seen that a move leads to a win node if and only if the backed-up value is equal to $N_h$, and a move leads to a loss node if and only if the backed-up value is less than $N_h$. Therefore, the player at the root node can correctly choose a move by evaluating nodes at the search frontier with function $\psi$ and minimax backing up the evaluations. Since $h$ can take any value between 0 and $H_0 - 1$, where $H_0$ is the height of dependent game tree $G$, the lemma follows. $\square$

It can also prove that when $f_2 = 1$, an observation similar to that presented in the proof of Lemma 3.1 holds. In fact, if $f_2 = 1$, all the leaf nodes in the minimax game tree following a loss node must have value 0. Fig. 3.3 shows two subtrees the roots of which have a height 4, which can be generated by dependent factor $f_2 = 1$. Note that since the root node of the first subtree has a merit value 0 and $f_2 = 1$, all of its leaf nodes are assigned value 0. Therefore, the evaluation function $\psi$ can always lead to best moves in minimax search of dependent game trees with $f_2 = 1$.



(a)



(b)

Fig. 3.3 Two Dependent Game Trees Generated with Parameters $f_1 \neq 1, f_2 = 1$

When $f_1 = 0$, each interior win Max node has exactly one win successor; when $f_2 = 0$, each interior loss Min node has exactly one loss successor. Based on this fact, it can be proved that in a dependent game tree that is generated with $f_1 = f_2 = 0$, a win node will have more 1-squares than a loss node at the same height, and all win nodes at the same height have the same number of 1-squares in their configurations, and so are all loss nodes at the same height. These properties imply that searching deeper in a dependent game tree which is generated with $f_1 = f_2 = 0$ will not decrease the probability of making a correct decision at the root n.d·

*Lemma*. 3.2. For a D-game that is generated with $f_1 = f_2 = 0$, the evaluation function $\psi$ can be used to choose the correct move with a search to depth $d$ for any integer $d \geq 1$.

*Proof.* By induction on the height $h$ of nodes in the dependent minimax game tree, we prove that the utility $\psi(g)$ of a win node $g$ is greater than $\psi(g')$ of a loss node $g'$ at the same height, and all the win (loss) nodes at the same height have the same number of 1-squares in their corresponding configurations. In fact, when $h$ 1, this conclusion is trivially true. Suppose the conclusion is true for $h < H$, and $g$ and $g'$ are win and loss nodes at height $H$, respectively. If $g$ and $g'$ are Max nodes, exactly one of the two successors of $g$ is a win node, and both successors of $g'$ are loss nodes. By induction,

$$\psi(g) = \psi(g_1) + \psi(g_2) > \psi(g'_1) + \psi(g'_2) = \psi(g').$$

If $g$ and $g'$ are Min nodes, since $g$ is a win node and $g'$ is a loss node, both successors of $g$ are win nodes, and exactly one of successors of $g'$ is a loss node. Therefore,

$$\psi(g) = \psi(g_1) + \psi(g_2) > \psi(g'_1) + \psi(g'_2) = \psi(g').$$

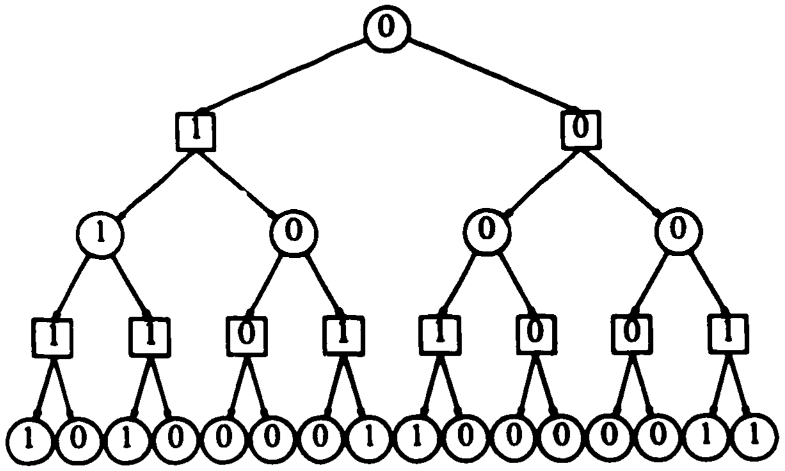
Let $e_0$ ($e_1$) denote the number of 1-squares in the configuration of a loss (win) node at height $h-1$. By induction, we can show that if nodes at height $h$ are Max nodes, all win nodes $g$ at height $h$ have the same number of 1-squares in their configurations, which is $e_0 + e_1$, and all loss nodes at height $h$ have the same number of 1-squares, which is $2e_0$; if nodes at height $h$ are Min nodes, all win nodes $g$ at height $h$ have the same number of 1-squares in their configurations, which is $2e_1$, and all loss nodes at height $h$ have the same number of 1-squares, which is $e_0 + e_1$.

Let the terminal nodes have height $h$, and a win (loss) node at height $h$ have $N_h$ ($O_h$) 1-squares in its configuration. By the above observation, it is easy to see that a move leads to a win node if and only if the backed-up utility value is equal to $N_h$, and a move leads to a loss node if and only if the backed-up utility value is $O_h$. Therefore, the player at the root node can correctly choose the move by evaluating nodes with function $\psi$ at the search frontier and minimax backing up the evaluations. Since $h$ can take any value between 0 and $H_0 - 1$, where $H_0$ is the height of dependent game tree, the lemma follows. □

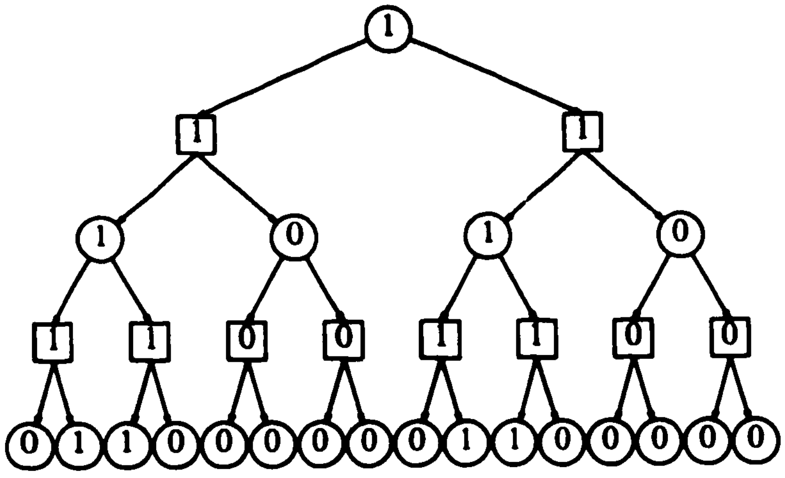
Thc examples shown in Fig. 3.4 (Fig. 3.5) can be used to illustrated that for $f_1 = 0$ and $f_2 \neq 0$ ($f_1 \neq 0$ and $f_2 = 0$), some dependent game trees can be generated which have win node $g$ and loss node $g'$ at the same height with

$$\psi(g) < \psi(g').$$

In other words, the property presented in Lemma 3.2 does not hold for $f_1 = 0$ and $f_2 \neq 0$ ($f_1 \neq 0$ and $f_2 = 0$), and minimax search with evaluation function $\psi$ may generate wrong decisions. This phenomenon will be observed in Section 5.

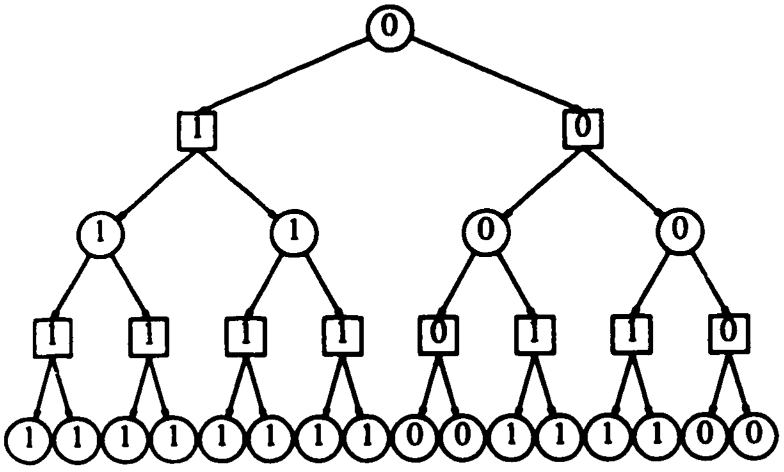


(a)

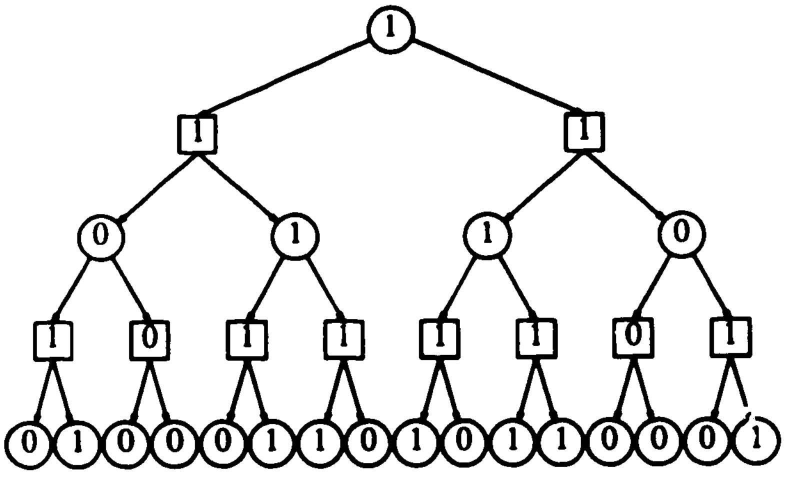


(b)

**Fig. 3.4 Two Dependent Game Trees Generated with Parameters $f_1 = 0, f_2 \neq 0$**



(a)



(b)

Fig. 3.5 Two Dependent Game Trees Generated with Parameters $f_1 \neq 0, f_2 = 0$

## 4. Mathematical Calculation for Decision Making

Let $G$ be a dependent game tree, and suppose node $g$ has height $h(g)$. The *depth d utility value* of $g$ for integer $d > 0$ is defined by

$$e_d(g) = \begin{cases} \psi(g) & \text{if } d=0 \\ \min(e_{d-1}(g_1), e_{d-1}(g_2)) & \text{if } d>0 \text{ and } g \text{ is a Min node} \\ \max(e_{d-1}(g_1), e_{d-1}(g_2)) & \text{if } d>0 \text{ and } g \text{ is a Max node.} \end{cases}$$

Choosing a move at $g$ using a depth $d$ search means choosing the child of $g$ that has the best depth $d-1$ utility value (so that the nodes at depth $d$ relative to $g$ determines the decision that is to be made at $g$). By the "best" value we mean the highest value if Max is to move at node $g$, or the least value if Min is to move. If both children receive the same value, then the player must choose one of them randomly. It is obvious that the probability of making a correct decision depends on the accuracy of the depth $d-1$ utility values of $g_1$ and $g_2$. In the following, we discuss how to compute the probability of making a correct decision in D-games.

### 4.1. Computation of Probability Distributions for Utilities

By the construction of D-games, the number of squares in the configuration for node $g$ is $2^{h(g)}$, which is the maximum possible value of $\psi(g)$. From the definition of the dependent factor $f_1$, if $g$ is a Max node and $h(g) = h > 0$,

$$\begin{aligned}
&\Pr[\psi(g)=i \mid h(g)=h, \text{win}(g)] \\
&= \sum_{j=0}^{i} \Pr[\psi(g_1)=j, \psi(g_2)=i-j \mid h(g)=h, \text{win}(g)] \\
&= \sum_{j=0}^{i} (\Pr[\psi(g_1)=j, \psi(g_2)=i-j \mid h(g_1)=h-1, h(g_2)=h-1, \text{win}(g_1), \text{win}(g_2)] \\
&\qquad \times \Pr[\text{win}(g_1), \text{win}(g_2) \mid \text{win}(g)] \\
&+ \Pr[\psi(g_1)=j, \psi(g_2)=i-j \mid h(g_1)=h-1, h(g_2)=h-1, \text{win}(g_1), \text{loss}(g_2)]
\end{aligned}$$

$$\times \Pr[\text{win}(g_1), \text{loss}(g_2) \mid \text{win}(g)]$$

$$+ \Pr[\psi(g_1)=j, \psi(g_2)=i-j \mid h(g_1)=h-1, h(g_2)=h-1, \text{loss}(g_1), \text{win}(g_2)]$$

$$\times \Pr[\text{loss}(g_1), \text{win}(g_2) \mid \text{win}(g)])$$

$$= \sum_{j=0}^{i} (\Pr[\psi(g_1)=j \mid h(g_1)=h-1, \text{win}(g_1)] \Pr[\psi(g_2)=i-j \mid h(g_2)=h-1, \text{win}(g_2)] f_1$$

$$+ \Pr[\psi(g_1)=j \mid h(g_1)=h-1, \text{win}(g_1)] \Pr[\psi(g_2)=i-j \mid h(g_2)=h-1, \text{loss}(g_2)](1-f_1))$$

and

$$\Pr[\psi(g)=i \mid h(g)=h, \text{loss}(g)]$$

$$= \sum_{j=0}^{i} \Pr[\psi(g_1)=j, \psi(g_2)=i-j \mid h(g)=h, \text{loss}(g)]$$

$$= \sum_{j=0}^{i} \Pr[\psi(g_1)=j, \psi(g_2)=i-j \mid h(g_1)=h-1, h(g_2)=h-1, \text{loss}(g_1), \text{loss}(g_2)]$$

$$= \sum_{j=0}^{i} \Pr[\psi(g_1)=j \mid h(g_1)=h-1, \text{loss}(g_1)] \Pr[\psi(g_2)=i-j \mid h(g_2)=h-1, \text{loss}(g_2)].$$

Similarly, when $g$ is a Min node and $h(g) = h > 0$,

$$\Pr[\psi(g)=i \mid h(g)=h, \text{loss}(g)]$$

$$= \sum_{j=0}^{i} (\Pr[\psi(g_1)=j, \psi(g_2)=i-j \mid h(g)=h, \text{loss}(g)]$$

$$= \sum_{j=0}^{i} (\Pr[\psi(g_1)=j, \psi(g_2)=i-j \mid h(g_1)=h-1, h(g_2)=h-1, \text{loss}(g_1), \text{loss}(g_2)]$$

$$\times \Pr[\text{loss}(g_1), \text{loss}(g_2) \mid \text{loss}(g)]$$

$$+ \Pr[\psi(g_1)=j, \psi(g_2)=i-j \mid h(g_1)=h-1, h(g_2)=h-1, \text{win}(g_1), \text{loss}(g_2)]$$

$$\times \Pr[\text{win}(g_1), \text{loss}(g_2) \mid \text{loss}(g)]$$

$$+ \Pr[\psi(g_1)=j, \psi(g_2)=i-j \mid h(g_1)=h-1, h(g_2)=h-1, \text{loss}(g_1), \text{win}(g_2)]$$

$$\times \Pr[\text{loss}(g_1), \text{win}(g_2) \mid \text{loss}(g)])$$

$$= \sum_{j=0}^{i} (\Pr[\psi(g_1)=j \mid h(g_1)=h-1, \text{loss}(g_1)] \Pr[\psi(g_2)=i-j \mid h(g_2)=h-1, \text{loss}(g_2)] f_2$$

$$+ \Pr[\psi(g_1)=j \mid h(g_1)=h-1, \text{win}(g_1)] \Pr[\psi(g_2)=i-j \mid h(g_2)=h-\text{[illegible]})](1-f_2))$$

and

$$\Pr[\psi(g)=i \mid h(g)=h, \mathrm{win}(g)]$$
$$= \sum_{j=0}^{i} \Pr[\psi(g_1)=j, \psi(g_2)=i-j \mid h(g_1)=h-1, h(g_2)=h-1, \mathrm{win}(g_1), \mathrm{win}(g_2)]$$
$$= \sum_{j=0}^{i} \Pr[\psi(g_1)=j \mid h(g_1)=h-1, \mathrm{win}(g_1)]\Pr[\psi(g_2)=i-j \mid h(g_2)=h-1, \mathrm{win}(g_2)].$$

The initial conditions for the above probabilities are

$$\Pr[\psi(g)=0 \mid h(g)=0, \mathrm{win}(g)] = 0, \Pr[\psi(g)=1 \mid h(g)=0, \mathrm{win}(g)] = 1,$$
$$\Pr[\psi(g)=0 \mid h(g)=0, \mathrm{loss}(g)] = 1, \Pr[\psi(g)=1 \mid h(g)=0, \mathrm{loss}(g)] = 0.$$

For any given dependent minimax game tree, the probability distributions for the utilities can be recursively calculated by the above formulas.

## 4.2. The Probability of Making Correct Decision

Suppose a player is to choose a move at some node $g$ of height $h$ in a D-game by searching to depth $d$ with $h \geq d$. If one of $g$'s children (say, $g_1$) is a win node and the other ($g_2$) is a loss node, then we can define a correct move to be the move to $g_1$ if the player is Max, or to $g_2$ if the player is Min. Since we are only interested in the probability of making a correct decision in the case where it makes a difference what move is made, a correct move is not defined if both children are win or loss nodes.

Since player Max moves to the node of highest utility and player Min moves to the node of lowest utility, a correct move will be made if $e_{d-1}(g_1) > e_{d-1}(g_2)$, and an incorrect decision will be made if $e_{d-1}(g_1) < e_{d-1}(g_2)$. If $e_{d-1}(g_1) = e_{d-1}(g_2)$, the player must choose among $g_1$ and $g_2$ at random, whence the probability of correct decision will be 1/2. Hence the correct decision at $g$ is

$$D(d,h) = \Pr[e_{d-1}(g_1) > g_{d-1}(g_2)] + \frac{1}{2}\Pr[e_{d-1}(g_1) = g_{d-1}(g_2)]$$
$$= \sum_{j=0}^{2^{h-1}} (\Pr[e_{d-1}(g_1) \geq j+1, g_{d-1}(g_2 \quad j] + \frac{1}{2}\Pr[e_{d-1}(g_1) = g_{d-1}(g_2) = j]).$$

As in Nau [6], if we define

$$mw(i,d,h) = \Pr[e_d(g) \geq i \mid h(g)=h, \text{win}(g)]$$

and

$$ml(i,d,h) = \Pr[e_d(g) \geq i \mid h(g)=h, \text{loss}(g)],$$

then

$$\begin{aligned} &D(d,h) \\ &= \sum_{j=0}^{2^h-1} (mw(j+1,d-1,h-1)(ml(j,d-1,h-1) - ml(j+1,d-1,h-1)) \\ &\qquad + \frac{1}{2}(mw(j,d-1,h-1) - mw(j+1,d-1,h-1))(ml(j,d-1,h-1) - ml(j+1,d-1,h-1))) \\ &= \sum_{j=0}^{2^h-1} \frac{1}{2}(mw(j,d-1,h-1) + mw(j+1,d-1,h-1))(ml(j,d-1,h-1) - ml(j+1,d-1,h-1)). \end{aligned}$$

We now discuss how to compute $mw$ and $ml$. Since $\psi(g)$ is never greater than $2^h$,

$$mw(2^h,0,h) = \Pr[\psi(g)=2^h \mid h(g)=h, \text{win}(g)]$$

and

$$ml(2^h,0,h) = \Pr[\psi(g)=2^h \mid h(g)=h, \text{loss}(g)]$$

for $h = 0, 1, 2, \ldots$. These values can be computed by the formulas presented in Section 5.1. And, then

$$\begin{aligned} mw(i,0,h) &= mw(i+1,0,h) + \Pr[\psi(g)=i \mid h(g)=h, \text{win}(g)], \\ ml(i,0,h) &= ml(i+1,0,h) + \Pr[\psi(g)=i \mid h(g)=h, \text{loss}(g)], \end{aligned}$$

for $i = 2^h-1$ down to 0.

Suppose $h$ is even, $g$ is a node of height $h$, and positive integer $d \leq h$. Since $g$ is a Min node, we have

$$\begin{aligned} &mw(i,d,h) \\ &= \Pr[e_d(g) \geq i \mid \text{win}(g)] \\ &= \Pr[e_{d-1}(g_1) \geq i, e_{d-1}(g_2) \geq i \mid \text{win}(g_1), \text{win}(g_2)] \end{aligned}$$

$$= \Pr[e_{d-1}(g_1) \ge i \mid \text{win}(g_1)]^2$$

$$= mw(i, d-1, h-1)^2.$$

Since node $g$ is a loss node if and only if at least one of its successors $g_1$ and $g_2$ is a loss node, we have

$$ml(i, d, h)$$

$$= \Pr[e_d(g) \ge i \mid \text{loss}(g)]$$

$$= \Pr[e_{d-1}(g_1) \ge i, e_{d-1}(g_2) \ge i \mid \text{loss}(g_1), \text{loss}(g_2)]$$

$$\times \Pr[\text{loss}(g_1), \text{loss}(g_2) \mid \text{loss}(g)]$$

$$+ \Pr[e_{d-1}(g_1) \ge i, e_{d-1}(g_2) \ge i \mid \text{win}(g_1), \text{loss}(g_2)]$$

$$\times \Pr[\text{win}(g_1), \text{loss}(g_2) \mid \text{loss}(g)]$$

$$+ \Pr[e_{d-1}(g_1) \ge i, e_{d-1}(g_2) \ge i \mid \text{loss}(g_1), \text{win}(g_2)]$$

$$\times \Pr[\text{loss}(g_1), \text{win}(g_2) \mid \text{loss}(g)]$$

$$= \Pr[e_{d-1}(g_1) \ge i \mid \text{loss}(g_1)]^2 f_2$$

$$+ \Pr[e_{d-1}(g_1) \ge i \mid \text{win}(g_1)] \Pr[e_{d-1}(g_2) \ge i \mid \text{loss}(g_2)] \frac{1}{2}(1 - f_2)$$

$$+ \Pr[e_{d-1}(g_1) \ge i \mid \text{loss}(g_1)] \Pr[e_{d-1}(g_2) \ge i \mid \text{win}(g_2)] \frac{1}{2}(1 - f_2)$$

$$= ml(i, d-1, h-1)^2 f_2 + ml(i, d-1, h-1) mw(i, d-1, h-1)(1 - f_2).$$

Suppose $h$ is odd instead of even, then $g$ is a Max node, so $e_d(g) < i$ if and only $e_{d-1}(g_1) < i$ and $e_{d-1}(g_2) < i$. Now $g$ is a win position if and only if either $g_1$ or $g_2$ is a win. Therefore,

$$mw(i, d, h)$$

$$= 1 - \Pr[e_d(g) < i \mid \text{win}(g)]$$

$$= 1 - (\Pr[e_{d-1}(g_1) < i, e_{d-1}(g_2) < i \mid \text{win}(g_1), \text{win}(g_2)]$$

$$\times \Pr[\text{win}(g_1), \text{win}(g_2) \mid \text{win}(g)]$$

$$+ \Pr[e_{d-1}(g_1) < i, e_{d-1}(g_2) < i \mid \text{win}(g_1), \text{loss}(g_2)]$$

$$\times \Pr[\text{win}(g_1), \text{loss}(g_2) \mid \text{win}(g)]$$
$$+ \Pr[e_{d-1}(g_1)<i, e_{d-1}(g_2)<i \mid \text{loss}(g_1), \text{win}(g_2)]$$
$$\times \Pr[\text{loss}(g_1), \text{win}(g_2) \mid \text{win}(g)])$$
$$= 1 - (\Pr[e_{d-1}(g_1)<i \mid \text{win}(g_1)]^2 f_1$$
$$+ \Pr[e_{d-1}(g_1)<i \mid \text{win}(g_1)]\Pr[e_{d-1}(g_2)<i \mid \text{loss}(g_2)]\frac{1}{2}(1-f_1)$$
$$+ \Pr[e_{d-1}(g_1)<i \mid \text{loss}(g_1)]\Pr[e_{d-1}(g_2)<i \mid \text{win}(g_2)]\frac{1}{2}(1-f_1))$$
$$= 1 - ((1 - mw(i, d-1, h-1))^2 f_1$$
$$+ (1 - ml(i, d-1, h-1))(1 - mw(i, d-1, h-1))(1 - f_1)).$$

Now $g$ is a loss position if and only if both $g_1$ or $g_2$ are loss nodes.

$$ml(i, d, h)$$
$$= 1 - \Pr[e_d(g)<i \mid \text{loss}(g)]$$
$$= 1 - \Pr[e_{d-1}(g_1)<i, e_{d-1}(g_2)<i \mid \text{loss}(g_1), \text{loss}(g_2)]$$
$$= 1 - \Pr[e_{d-1}(g_1)<i \mid \text{loss}(g_1)]^2$$
$$= 1 - (1 - ml(i, d-1, h-1))^2.$$

Thus for $h = 0, 1, \ldots, d = 0, \ldots, h$ and $i = 0, \ldots, 2^h$, $mw(i, d, h)$ and $ml(i, d, h)$ can be recursively computed.

## 4.3. Numerical Results

A program was written to implement the above formulas. For some different values of dependent factors $f_1$ and $f_2$, the probabilities of making a correct decision at the root node with search to various depths $d$ were calculated. The probabilities for a dependent game tree of height 9 with dependent factors $f_1 = f_2 = i/10$ for $0 \leq i \leq 10$ are listed in Table 3.1. As proved by Lemmas 3.1 and 3.2, when $f_1 = f_2 = 0$ or $f_1 = f_2 = 1$, the evaluation function can determine the correct move for any search depth $d$. But when $f_1 = f_2 = 0.1$, pathology is present, since since searching to depth 2 is worse than to depth 1, i.e., one move search is worse than the static evaluation. As the dependent factors increase, the pathology gradually disappears. The last row of Table 3.1 lists for each column in the table a sum, called ***pathology sum*** or PS, which is calculated in the following way that if $D(d-1, 9) > D(d, 9)$ for $0 < d \leq 8$, then this pathological depth $d$ search contributes $D(d-1, 9) - D(d, 9)$ to the sum. Pathology sum is used here as an approximate indication of the degree of pathology present in a game. When $f_1 = f_2 = 0.1$, the pathology sum rises sharply to a peak and, then, drops equally quickly. After rising to the second peak, the pathological sums decrease strictly along with the increase of $f_1$ and $f_2$. When $f_1 = f_2 > 0.5$, pathology totally disappears.

Data were also collected for the dependent factors $f_1 = i/10$, $f_2 = (10-i)/10$ for $0 \leq i \leq 10$. The data are presented in Table 3.2. When $f_1 = 1$ or $f_2 = 1$, the pathological sum is zero and there is no pathology in minimax search. This phenomenon is also described by Lemma 3.1. Note the extremely large pathological sums for dependent factors $f_1 = 0.8, f_2 = 0.2$ and $f_1 = 0.9, f_2 = 0.1$, which are 0.122 and 0.206 respectively. In real games, since both players apply the same set of rules, the node dependence with $f_1 >> f_2$ or vice versa that $f_1 << f_2$ should not appear. In other words, the difference $|f_1 - f_2|$ between the dependent factors for a real game should be relatively small. Since a player does not include the obvious bad move into a game tree, the game trees actually established in the computer have a strong positive correlation. The corresponding node-dependent game tree should have large $f_1$ and $f_2$. This explains why computer game playing programs can avoid minimax pathology and do better by searching deeper.

More data are shown in Table 3.3, which contains the probabilities of making a correct decision in dependent minimax game trees of height 10 with various depth search. For this experiment, the dependent factors $f_1$ and $f_2$ force both the root and the leaf nodes to have the same probability, 0.5, to take value 1. Generally speaking, the pathology sums also decrease along with the increase of dependent factors. Twenty-one different values for $f_2$ have been used in this calculation, the difference between an adjacent pair of $f_2$ is 0.033 or 0.034; the corresponding values of $f_1$ range from 0 to 1. Note that there are only nine of the twenty-one columns where minimax search exhibits pathology. In these nine "pathological" columns, only 17 pairs of adjacent data contribute non-zero values to the pathological sums. Since there are 189 pairs of adjacent data in Table 3.3, we can say that minimax search pathology is an uncommon event, which harasses only games with less positively correlated sibling node values and disappears when searching deeper.

Table 3.1 The Probabilities for Correct Decision with $h = 9$ and $f_1 = f_2$

| $d$ | $f_1=f_2$ 0.0 | $f_1=f_2$ 0.1 | $f_1=f_2$ 0.2 | $f_1=f_2$ 0.3 | $f_1=f_2$ 0.4 | $f_1=f_2$ 0.5 | $f_1=f_2$ 0.6 | $f_1=f_2$ 0.7 | $f_1=f_2$ 0.8 | $f_1=f_2$ 0.9 | $f_1=f_2$ 1.0 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1.000 | 0.637 | 0.679 | 0.743 | 0.816 | 0.883 | 0.937 | 0.973 | 0.992 | 0.999 | 1.000 |
| 2 | 1.000 | 0.634 | 0.674 | 0.738 | 0.813 | 0.884 | 0.942 | 0.979 | 0.996 | 1.000 | 1.000 |
| 3 | 1.000 | 0.641 | 0.684 | 0.756 | 0.840 | 0.915 | 0.966 | 0.991 | 0.999 | 1.000 | 1.000 |
| 4 | 1.000 | 0.644 | 0.684 | 0.753 | 0.836 | 0.914 | 0.968 | 0.993 | 0.999 | 1.000 | 1.000 |
| 5 | 1.000 | 0.680 | 0.744 | 0.829 | 0.910 | 0.966 | 0.991 | 0.998 | 1.000 | 1.000 | 1.000 |
| 6 | 1.000 | 0.655 | 0.770 | 0.874 | 0.940 | 0.977 | 0.994 | 0.999 | 1.000 | 1.000 | 1.000 |
| 7 | 1.000 | 0.975 | 0.976 | 0.984 | 0.991 | 0.997 | 0.999 | 1.000 | 1.000 | 1.000 | 1.000 |
| 8 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| 9 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| PS | 0.000 | 0.028 | 0.005 | 0.008 | 0.007 | 0.001 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |

Table 3.2 The Probabilities for Correct Decision with $h = 9$ and $f_1 + f_2 = 1$

| $d$ | $f_1\ f_2$ 0.0 1.0 | $f_1\ f_2$ 0.1 0.9 | $f_1\ f_2$ 0.2 0.8 | $f_1\ f_2$ 0.3 0.7 | $f_1\ f_2$ 0.4 0.6 | $f_1\ f_2$ 0.5 0.5 | $f_1\ f_2$ 0.6 0.4 | $f_1\ f_2$ 0.7 0.3 | $f_1\ f_2$ 0.8 0.2 | $f_1\ f_2$ 0.9 0.1 | $f_1\ f_2$ 1.0 0.0 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1.000 | 0.982 | 0.945 | 0.915 | 0.895 | 0.883 | 0.879 | 0.884 | 0.904 | 0.950 | 1.000 |
| 2 | 1.000 | 0.989 | 0.948 | 0.915 | 0.894 | 0.884 | 0.885 | 0.896 | 0.923 | 0.971 | 1.000 |
| 3 | 1.000 | 1.000 | 0.994 | 0.974 | 0.945 | 0.915 | 0.887 | 0.867 | 0.864 | 0.910 | 1.000 |
| 4 | 1.000 | 1.000 | 0.996 | 0.978 | 0.947 | 0.914 | 0.887 | 0.874 | 0.883 | 0.943 | 1.000 |
| 5 | 1.000 | 1.000 | 1.000 | 0.998 | 0.989 | 0.966 | 0.927 | 0.881 | 0.844 | 0.858 | 1.000 |
| 6 | 1.000 | 1.000 | 1.000 | 0.999 | 0.995 | 0.977 | 0.938 | 0.899 | 0.862 | 0.893 | 1.000 |
| 7 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 0.997 | 0.984 | 0.939 | 0.838 | 0.833 | 1.000 |
| 8 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| 9 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| PS | 0.000 | 0.000 | 0.000 | 0.000 | 0.001 | 0.001 | 0.000 | 0.029 | 0.122 | 0.206 | 0.000 |

Table 3.3 The Probabilities for Correct Decision with $k = 10$ and $p_0 = 0.5$

| $d$ | $f_1\ f_2$ | $f_1\ f_2$ | $f_1\ f_2$ | $f_1\ f_2$ | $f_1\ f_2$ | $f_1\ f_2$ | $f_1\ f_2$ |
|---|---|---|---|---|---|---|---|
| | 0.000 0.333 | 0.073 0.367 | 0.143 0.400 | 0.209 0.433 | 0.273 0.467 | 0.333 0.500 | 0.391 0.533 |
| 1 | 0.643 | 0.665 | 0.697 | 0.733 | 0.772 | 0.809 | 0.845 |
| 2 | 0.684 | 0.703 | 0.732 | 0.767 | 0.805 | 0.841 | 0.874 |
| 3 | 0.673 | 0.693 | 0.722 | 0.756 | 0.793 | 0.830 | 0.865 |
| 4 | 0.751 | 0.770 | 0.797 | 0.830 | 0.865 | 0.898 | 0.927 |
| 5 | 0.737 | 0.763 | 0.793 | 0.826 | 0.861 | 0.894 | 0.924 |
| 6 | 0.911 | 0.922 | 0.935 | 0.949 | 0.963 | 0.974 | 0.983 |
| 7 | 0.959 | 0.970 | 0.977 | 0.983 | 0.988 | 0.991 | 0.994 |
| 8 | 1.000 | 0.999 | 0.999 | 0.999 | 0.999 | 0.999 | 0.999 |
| 9 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| 10 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| PS | 0.025 | 0.017 | 0.014 | 0.015 | 0.016 | 0.015 | 0.012 |

| $d$ | $f_1\ f_2$ | $f_1\ f_2$ | $f_1\ f_2$ | $f_1\ f_2$ | $f_1\ f_2$ | $f_1\ f_2$ | $f_1\ f_2$ |
|---|---|---|---|---|---|---|---|
| | 0.447 0.567 | 0.500 0.600 | 0.551 0.633 | 0.600 0.667 | 0.647 0.700 | 0.692 0.733 | 0.736 0.767 |
| 1 | 0.877 | 0.905 | 0.928 | 0.948 | 0.963 | 0.975 | 0.984 |
| 2 | 0.904 | 0.929 | 0.949 | 0.965 | 0.977 | 0.986 | 0.992 |
| 3 | 0.898 | 0.926 | 0.949 | 0.967 | 0.980 | 0.989 | 0.995 |
| 4 | 0.951 | 0.968 | 0.981 | 0.989 | 0.994 | 0.997 | 0.999 |
| 5 | 0.949 | 0.968 | 0.982 | 0.990 | 0.995 | 0.998 | 0.999 |
| 6 | 0.990 | 0.994 | 0.997 | 0.999 | 0.999 | 1.000 | 1.000 |
| 7 | 0.996 | 0.997 | 0.998 | 0.999 | 1.000 | 1.000 | 1.000 |
| 8 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| 9 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| 10 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| PS | 0.008 | 0.003 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |

| $d$ | $f_1 f_2$ 0.778 0.800 | $f_1 f_2$ 0.818 0.833 | $f_1 f_2$ 0.857 0.867 | $f_1 f_2$ 0.895 0.900 | $f_1 f_2$ 0.931 0.933 | $f_1 f_2$ 0.966 0.967 | $f_1 f_2$ 1.000 1.000 |
|---|---|---|---|---|---|---|---|
| 1 | 0.991 | 0.995 | 0.998 | 0.999 | 1.000 | 1.000 | 1.000 |
| 2 | 0.996 | 0.998 | 0.999 | 1.000 | 1.000 | 1.000 | 1.000 |
| 3 | 0.998 | 0.999 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| 4 | 0.999 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| 5 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| 6 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| 7 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| 8 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| 9 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| 10 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| PS | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |

Chapter 4

# Pruning Efficiency for Node-Dependent Game Trees

## 1. Alpha-Beta Pruning

In computer game playing, alpha-beta pruning is a commonly used technique for speeding up search processes. Knuth and Moore [4] credit the idea of pruning some nodes from the search process to McCarthy and his group at MIT, and trace back this search reduction method to 1958. The first formal treatment of this topic appears to be Brudno's 1963 paper [16]. This algorithm determines the utility of the root node of a game tree by traversing the tree in depth-first order and skipping all those nodes that can no longer influence the backing-up of utility $\Psi(g)$ for the root node $g$.

In this chapter, we assume $\psi(g) = \phi(g)$ for each terminal node $g$. Therefore, $\Psi(g) = \Phi(g)$ for every node $g$ in a game tree. In fact, this assumption implies that the alpha-beta algorithm searches the game trees to their bottoms. In this way, we can concentrate on the efficiency of the algorithm without being concerned by the accuracy of the evaluation functions. Another assumption for this chapter lets the root node of each game tree be a Max node. This assumption is customarily adopted in the literature.

The alpha-beta algorithm is called with three parameters: $g$, which is the node to be visited, and *alpha* and *beta*, which are integers with *alpha* < *beta*. The range [*alpha*, *beta*] is usually called a ***search window***, or simply, a ***window*** [17]. In alpha-beta pruning, the reduction of the game tree search is achieved by a node passing down to its successors the current value backed-up so far and a successor using these values as pruning bounds. For a minimax game tree, the alpha-beta pruning algorithm is defined by the following recursive procedures, taken from Knuth and Moore [4]: if node $g$ is a Max node,

```
integer procedure F1($g$ : Max node; alpha, beta: integer):
integer $m$ , $i$ , $t$ , $n$ ;
begin determine the successor nodes $g_1, \ldots, g_n$ ;
    if $n = 0$ then F1 := $\psi(g)$ else
    begin $m$ := alpha;
        for $i$ := 1 to $n$ do
            begin $t$ := F2($g_i$ , $m$ , beta);
                if $t > m$ then $m := t$ ;
                if $m \geq beta$ then return F1 := $m$ ;
            end;
        F1 := $m$ ;
    end.
end;
```

If $g$ is a Min node,

```
integer procedure F2($g$ : Min node; alpha, beta: integer):
integer $m$ , $i$ , $t$ , $n$ ;
begin determine the successor nodes $g_1, \ldots, g_n$ ;
    if $n = 0$ then F2 := $\psi(g)$ else
    begin $m$ := beta;
        for $i$ := 1 to $n$ do
            begin $t$ := F1($g_i$ , alpha, $m$ );
                if $t < m$ then $m := t$ ;
                if $m \leq alpha$ then return F2 := $m$ ;
            end;
        F2 := $m$ ;
    end;
end.
```

In the above procedures, the evaluation function $\psi(g)$ provides the utility for terminal nodes $g$. These procedures are proved to be correct in the sense that the call F1($g$, $-\infty$, $+\infty$) for a Max node $g$ or F2($g$, $-\infty$, $+\infty$) for a Min node $g$ always returns the utility $\Psi(g)$ for the node $g$, which is the value assigned by the minimax process [4]. More generally, it can be shown that:

$$\begin{aligned} &Fi(g, alpha, beta) \le alpha, \text{ if } \Psi(g) \le alpha, \\ &Fi(g, alpha, beta) = \Psi(g), \text{ if } alpha \le \Psi(g) \le beta, \qquad (4.1) \\ &Fi(g, alpha, beta) \ge beta, \text{ if } \Psi(g) \ge beta, \end{aligned}$$

where $i = 1$ if $g$ is a Max node, otherwise, $i = 2$. These conditions imply that

$$Fi(g, -\infty, +\infty) = \Psi(g).$$

The efficiency of alpha-beta pruning has been studied in several papers. The papers written by Fuller *et al.* [5] and Knuth and Moore [4] address the problem of searching a uniform game tree of degree $n$ and depth $d$ with alpha beta pruning under the assumption that the $n^d$ merit values assigned to the terminal nodes are all distinct and random. A general formula for the average number of terminal nodes examined by the alpha-beta pruning algorithm was developed by Fuller *et al.* [5]. This formula, however, is computationally intractable and leads to undesirable errors for large trees since it involves, in particular, a $2d - 2$ nested summation of terms with alternating signs and requires on the order of $n^d$ steps for its evaluation. A simpler version of the full alpha-beta pruning not considering the possibility of deep cut-offs was studied by Knuth and Moore [4]. Baudet [12] took into account both shallow and deep cut-offs, and developed a formula to compute the average number of terminal nodes examined by the alpha-beta algorithm in an independent uniform tree of degree $n$ and depth $d$, where ties are allowed among the terminal nodes. An analysis of the efficiency of alpha-beta pruning was conducted by Newborn [9] for a branch-dependent game tree. The analysis for this model is very difficult, did not even obtain the exact formula for game trees of depth four [9]. In summary, four schemes have been proposed for the analysis of alpha-beta pruning in the literature to date. All four schemes consider a uniform tree of degree $n$ and depth $d$. Scheme 1 simply assigns a different number to each of the $n^d$ terminal nodes at random. The second scheme, which is exploited by Baudet [12], randomly assigns the values from a finite set to the terminal nodes of the uniform game tree, and allows ties among these nodes. The third scheme, proposed by Knuth and Moore [4], assumes that the values are not independent, but dependent on the [illegible] as follows: randomly assign a weight value from [illegible] $/n^k, 2/n^k, \ldots, n/n^k$ } to each of the $n$ branches directed from a node $g$ at depth [illegible] let the score of a terminal node be equal to the sum of the weights of the br[illegible] along the path from the root node to that terminal node. The branch-dependent game tree model proposed by Fuller *et al.* [5] is identical with the third scheme with the exception that the set $\{1, 2, \ldots, n\}$ replaces the set $\{1/n^k, 2/n^k, \ldots, n/n^k\}$. Newborn made use of branch-dependent game trees in analyzing alpha-beta pruning [9].

Here, a different game tree model, the node-dependent minimax game tree, is used to analyze the alpha-beta algorithm. First, the effect of the initial search window on the efficiency of alpha-beta pruning is discussed. It is pointed out that the so called best case analysis presented in the literature to date applies only to the largest search window with alpha $= -\infty$ and beta $= +\infty$. Under the assumption of alpha-beta pruning with an appropriate search window, we derive the recursive equations for the average number of terminal nodes visited in a bi-valued binary node-dependent game tree. The effect of node-dependence on the pruning efficiency is analyzed. Finally, the analysis method is generalized for multi-valued $n$-ary dependent minimax game trees with $n \geq 2$.

## 2. Tight Window Pruning

The search window for the root node is usually set to $[-\infty, +\infty]$ in the literature when analyzing the efficiency of alpha-beta pruning [4,9], even if the range of the evaluation function is a finite set [12]. Under the assumption that the root node value is not equal to $\pm\infty$, the lower bound for the number of terminal nodes visited by the alpha-beta algorithm in a uniform game tree of degree $n$ and depth $d$ is

$$n^{\lfloor d/2 \rfloor} + n^{\lceil d/2 \rceil} - 1. \tag{4.2}$$

It will be shown that using a search window other than $[-\infty, +\infty]$ for the root node, a different lower bound for the number of visited terminal nodes can be derived. A bi-valued binary game tree of depth 1 is shown in Fig. 4.1. Let the root node be a Max node and its left child have a value 1. If the root node is visited with window $[-\infty, +\infty]$, both the left and right successors must be visited, and formula (4.2) gives the number $2^{\lfloor 1/2 \rfloor} + 2^{\lceil 1/2 \rceil} - 1 = 2$. But for this bi-valued tree, if the root node is visited with window $[0, 1]$, the right child is pruned from the search. Now only one terminal node is examined by the alpha-beta pruning algorithm, this observation stands in contrast to formula (4.2) and improves the pruning efficiency. Therefore, if the range of the evaluation function is a finite set $\{i_1, i_2, ..., i_q\}$, where $i_1 < i_2 < ... < i_q$, the window $[i_1, i_q]$, rather than $[-\infty, +\infty]$, should be used for the root node in an alpha-beta search. In this case, the window $[i_1, i_q]$ is called the *tight* window.

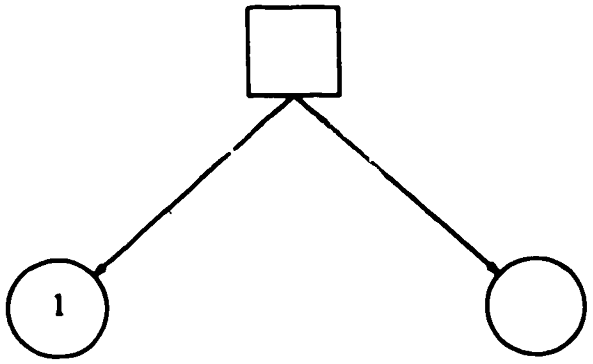


**Fig.** 4.1 A Binary Uniform Tree of Depth 1

The fact that executing the alpha-beta algorithm with a tight window $[i_1, i_q]$ can return the minimax value of the root node is simply derived from the conditions in (4.1). In fact, when the minimax value $\Psi(g)$ of the root node $g$ satisfies

$$i_1 \le \Psi(g) \le i_q,$$

by (4.1), it can be derived that

$$Fi(g, i_1, i_q) = \Psi(g),$$

i.e., the alpha-beta pruning algorithm with tight search window can correctly back up the minimax value of the root node.

A tight window for an alpha-beta search is best in that it always visits fewer nodes than other windows, this observation is summarized by the following lemma. In the proof of this lemma, we use the fact that if a window contains another one, an alpha-beta search with the first window either returns the same result as with the second window, or returns a value that is not contained in the second window. This conclusion about the alpha-beta algorithm is simply implied by the condition (4.1).

*Lemma* 4.1 For a game tree, if window $[alpha_1, beta_1]$ contains window $[alpha_2, beta_2]$ as a subset, the execution of the alpha-beta algorithm with the first window will visit all the nodes visited with the second window.

*Proof.* Assume two executions of the alpha-beta pruning algorithm, one with window $[alpha_1, beta_1]$ and the other with window $[alpha_2, beta_2]$ for the root node of a uniform tree. By induction, it can be proved that if a node $g$ is visited by both executions, the window used to visit $g$ by the first execution contains the window used by the second execution. Assume $g$ is the first node that does not satisfy this proposition, and $g'$ is its elder brother, the backed-up values for which in the two executions determine the relation between the visiting windows for $g$. Since the visiting window for $g'$ in the first execution contains that in the second execution, the backed-up value $v'_1$ for $g'$ in the first execution either is equal to the backed-up value $v'_2$, or is not in the window in the second execution. By the conditions of (4.1), we can deduce that if $g'$ is a Min node, $v'_1 \leq v'_2$, otherwise, $v'_1 \geq v'_2$. Therefore, the two backed-up values for $g'$ cannot change the relationship between the two windows used to visit $g$, a contradiction.

The above argument implies that, if a node is visited by the second execution, it must be visited by a non-empty window by the first execution, i.e., it must be visited by the first execution. $\square$

By the above discussion, the best window for the alpha-beta algorithm is determined by the smallest and the largest possible utility values. For bi-valued dependent minimax game trees, the search window for the root node will be set to $[0, 1]$ rather than $[-\infty, +\infty]$. The alpha-beta pruning algorithm will be analyzed with respect to this window. This analysis is different from the traditional methods [1, 4, 9, 12].

## 3. Some Properties of Alpha-Beta Pruning

In this section, some notations are introduced, and a necessary and sufficient condition for a node to be visited by the alpha-beta algorithm, which was presented by Baudet in the context of negamax [12], is described in terms of minimax and bi-valued binary game trees.

While the utility values $\Psi(g)$ deal with the static aspect of a game tree, the quantities that will be introduced deal more with the dynamic aspect of the tree when being searched by the alpha-beta algorithm. For a Max node $g_j$ at depth $d \geq 1$, we define

$$c(g_j) = \begin{cases} \Psi(g_{j-1}) & \text{if } j = 2 \\ 1 & \text{otherwise}; \end{cases}$$

for a Min node $g_j$ at depth $d \geq 1$, we define

$$c(g_j) = \begin{cases} \Psi(g_{j-1}) & \text{if } j = 2 \\ 0 & \text{otherwise}. \end{cases}$$

The quantity $c(g)$ accounts for the information provided to node $g$ by its elder brother, if any. For any node $g = j_1 \cdots j_d$ at depth $d \geq 1$ in a game tree, two quantities are directly assigned to the node by the alpha-beta algorithm, if the node is visited. For $i = 1, \ldots, d$, let $g(i) = j_1 \cdots j_i$. We define

$$\alpha(g) = \max\,\{c(g(i)) \mid i \text{ is odd}, 0 \leq i \leq d\},$$

$$\beta(g) = \min\,\{c(g(i)) \mid i \text{ is even}, 0 \leq i \leq d\},$$

where the node $g(0)$ represents the root, and $c(g(0)) = 1$ by convention. It is convenient to define $\alpha(g(0)) = 0$ and $\beta(g(0)) = 1$.

The following lemma justifies the notations just introduced. It is adapted from Theorem 2.1 of Baudet [12], where the root node is visited with window $[-\infty, +\infty]$.

*Lemma* 4.2 [12, Theorem 2.1]. Assume that, initially, the root node, denoted as *root*, of a bi-valued game tree is explored by the alpha-beta algorithm through the call

$$\mathrm{F1}(root, 0, 1).$$

A node $g$ in the game tree is visited by the alpha-beta algorithm if and only if $\alpha(g) = 0$ and $\beta(g) = 1$. $\square$

The above lemma implies the following lemma about bi-valued trees.

*Lemma* 4.3 If a Max (Min) node $g$ is visited by the alpha-beta algorithm with window [0, 1], the first successor $g_1$ must be visited by the alpha-beta algorithm with the same window, and the second successor $g_2$ is visited by the alpha-beta algorithm if and only if $\Psi(g_1) = 0$ (or $\Psi(g_1) = 1$).

*Proof*

Just note that

$$c(g_1) = 0,\ \alpha(g_1) = \max(\alpha(g), c(g_1)),\ \beta(g_1) = \beta(g)$$

if $g$ is a Max node, and

$$c(g_1) = 1,\ \alpha(g_1) = \alpha(g),\ \beta(g_1) = \min(\beta(g), c(g_1))$$

if $g$ is a Min node. If the visiting window for $g$ is [0, 1], i.e., $\alpha(g) = 0$, $\beta(g) = 1$, the visiting window must be [0, 1] for $g_1$. and $g_1$ must be visited by the alpha-beta algorithm. If node $g$ is a Max node and $\Psi(g_1) = 1$,

$$\alpha(g_2) = \max(\alpha(g), c(g_2))) : \qquad = \beta(g) = 1,$$

and node $g_2$ is pruned by the alpha-beta algorithm. ιr node $g$ is a Max node and $\Psi(g_1) =$ 0,

$$\alpha(g_2) = \max(\alpha(g), c(g_2))) = 0,\ \beta(g_2) = \beta(g) = 1,$$

and node $g_2$ will be visited by the alpha-beta algorithm. A similar argument holds for Min node $g$. □

## 4. Pruning Efficiency

Here, the complexity of the alpha-beta pruning algorithm is measured by the number of terminal nodes visited in a game tree, and the game tree used to analyze this algorithm is a bi-valued binary dependent minimax game tree with parameters $p_0$, $f_1$ and $f_2$. The quantity to be computed is $W(h, f_1, f_2)$ ($L(h, f_1, f_2)$) which is the average number of terminal nodes examined by the algorithm in a deoendent minimax game tree of height $h = 2k$ with a win (loss, respectively) root node and dependent factors $f_1$ and

$f_2$. The average number of terminal nodes visited by the alpha-beta pruning algorithm in a dependent minimax game tree with parameters $p_0$, $f_1$ and $f_2$ will be

$$p_0 W(h, f_1, f_2) + (1 - p_0) L(h, f_1, f_2).$$

The analysis is recursive from the root node towards the terminal nodes. In fact, only three adjacent levels of the dependent minimax game tree are involved in the discussion. Let the three levels be at heights $2k$, $2k-1$ and $2k-2$, respectively. If a node $g$ at height $2k$ is a win node, the possible merit value assignments for its children $g.1$ and $g.2$ are 1 and 1, 0 and 1, and 1 and 0, which are shown in Fig. 4.2(a), (b), and (c), respectively. For the assignment of 1 and 1 for $g.1$ and $g.2$, the possible merit value assignment for the nodes $g.1.1$, $g.1.2$, $g.2.1$ and $g.2.2$, which are at height $2k-2$, is 1, 1, 1 and 1. This merit value assignment is denoted as $w1$ in Fig. 4.2(a). Since the probability for the merit value assignment 1 and 1 of $g.1$ and $g.2$ is $f_1$, the probability for the merit value assignment $w1$ of $g.1.1$, $g.1.2$, $g.2.1$ and $g.2.2$ is $f_1$, which is also shown in Fig. 4.2(a). Similarly, all the other possible merit value assignments, which are denoted as $w2$ through $w7$, for the nodes $g.1.1$, $g.1.2$, $g.2.1$ and $g.2.2$ are listed in Fig. 4.2(b) and (c). The probabilities for these assignments are derived from the fact that $g$ is a win Max node and $f_1$ and $f_2$ are dependent factors. If the root node $g$ is visited with window [0, 1], according to Lemma 4.3, for the assignment $w1$, nodes $g.1.1$ and $g.1.2$ are visited, for the assignment $w2$, nodes $g.1.1$, $g.2.1$ and $g.2.2$ are visited, for the assignment $w3$, nodes $g.1.1$, $g.1.2$, $g.2.1$ and $g.2.2$ are visited, for the assignment $w4$, nodes $g.1.1$, $g.2.1$ and $g.2.2$ are visited, for the assignments $w5$, $w6$, and $w7$, nodes $g.1.1$ and $g.1.2$ are visited. According to the probabilities for the merit value assignments $w1$, ..., $w7$, the quantity

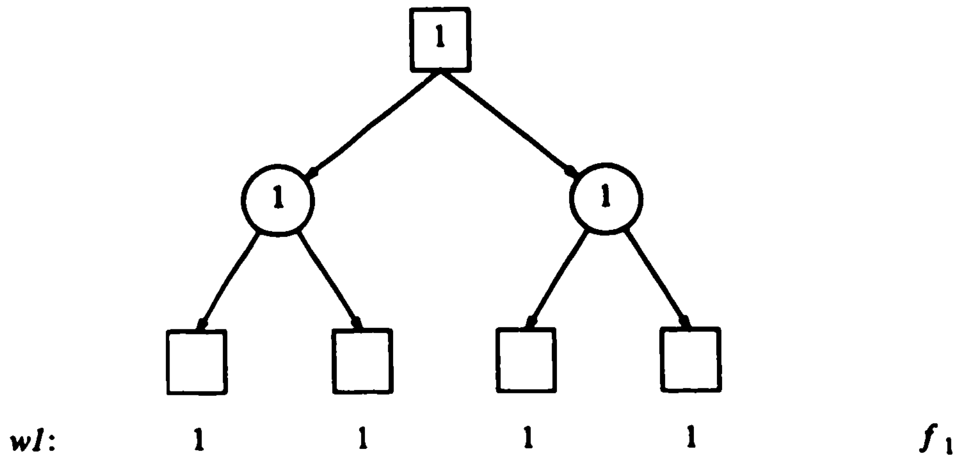
$$\begin{aligned}
&W(h, f_1, f_2) \\
&= f_1 2W(h-2, f_1, f_2) \\
&\qquad + \frac{1}{2}(1 - f_1) f_2 (2W(h-2, f_1, f_2) + L(h-2, f_1, f_2))
\end{aligned}$$

$$
\begin{aligned}
&+ \frac{1}{2}(1-f_1)\frac{1}{2}(1-f_2)(3W(h-2,f_1,f_2)+L(h-2,f_1,f_2)) \\
&+ \frac{1}{2}(1-f_1)\frac{1}{2}(1-f_2)(2W(h-2,f_1,f_2)+L(h-2,f_1,f_2)) \\
&+ \frac{1}{2}(1-f_1)f_2 2W(h-2,f_1,f_2) \\
&+ \frac{1}{2}(1-f_1)\frac{1}{2}(1-f_2)2W(h-2,f_1,f_2) \\
&+ \frac{1}{2}(1-f_1)\frac{1}{2}(1-f_2)2W(h-2,f_1,f_2) \\
= {} & (2+\frac{1}{4}(1-f_1)(1-f_2))W(h-2,f_1,f_2)+\frac{1}{2}(1-f_1)L(h-2,f_1,f_2).
\end{aligned}
$$

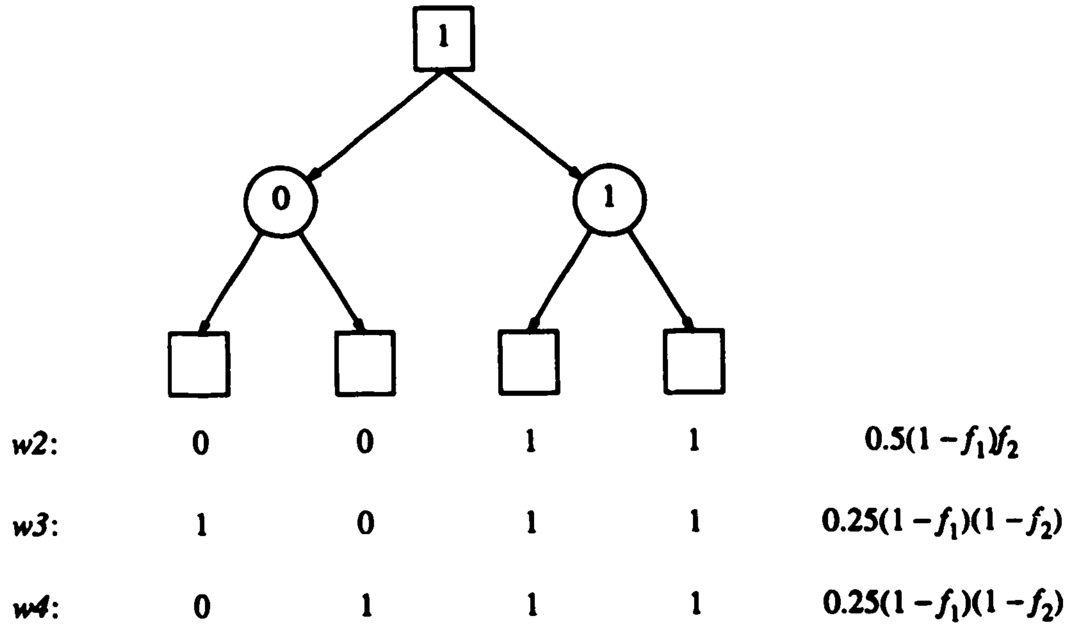
If the root node $g$ is a loss node, both of its children, $g.1$ and $g.2$, must be loss nodes. The possible merit value assignments for the successors of its children, $g.1.1$, $g.1.2$, $g.2.1$ and $g.2.2$, are listed in Fig. 4.3. The probability for each assignment can be derived from the fact that $g$ is a loss Max node and $f_1$ and $f_2$ are the dependent factors, and is listed at the right side in Fig. 4.3. If the root node $g$ is visited with window [0, 1], by Lemma 4.3, for assignment *l1*, nodes $g.1.1$ and $g.2.1$ are visited, for assignment *l2*, nodes $g.1.1$, $g.1.2$ and $g.2.1$ are visited, for assignment *l3*, nodes $g.1.1$ and $g.2.1$ are visited, for assignment *l4*, nodes $g.1.1$, $g.2.1$ and $g.2.2$ are visited, for assignment *l5*, nodes $g.1.1$ and $g.2.1$ are visited, for assignment *l6*, nodes $g.1.1$, $g.1.2$, $g.2.1$ and $g.2.2$ are visited, for assignment *l7*, nodes $g.1.1$, $g.1.2$, and $g.2.1$ are visited, for assignment *l8*, nodes $g.1.1$, $g.2.1$ and $g.2.2$ are visited, for assignment *l9*, nodes $g.1.1$ and $g.2.1$ are visited. Therefore, the average number of terminal nodes visited for assignment *l1* is $2L(h-2,f_1,f_2)$, the average number for *l2* is $W(h-2,f_1,f_2) + 2L(h-2,f_1,f_2)$, the average number for *l3* is $2L(h-2.f_1,f_2)$, the average number for *l4* is $W(h-2,f_1,f_2) + 2L(h-2,f_1,f_2)$, the average number for *l5* is $2L(h-2,f_1,f_2)$, the average number for *l6* is $2W(h-2,f_1,f_2) + 2L(h-2,f_1,f_2)$, the average number for *l7* is $W(h-2,f_1,f_2) + 2L(h-2,f_1,f_2)$, the average number for *l8* is $W(h-2,f_1,f_2) +$

$2L(h-2, f_1, f_2)$, the average number for *l9* is $2L(h-2, f_1, f_2)$. According to the probabilities for the assignments, the quantity

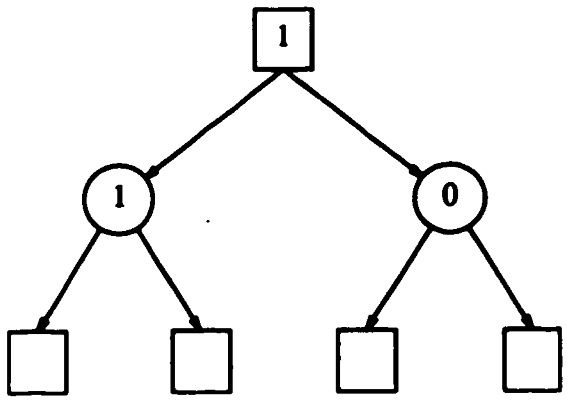
$$
\begin{aligned}
&L(h, f_1, f_2) \\
&= f_2^2 \, 2L(h-2, f_1, f_2) \\
&\quad + \frac{1}{2}(1-f_2)f_2(W(h-2, f_1, f_2) + 2L(h-2, f_1, f_2)) \\
&\quad + \frac{1}{2}(1-f_2)f_2 2L(h-2, f_1, f_2) \\
&\quad + \frac{1}{2}(1-f_2)f_2(W(h-2, f_1, f_2) + 2L(h-2, f_1, f_2)) \\
&\quad + \frac{1}{2}(1-f_2)f_2 2L(h-2, f_1, f_2) \\
&\quad + \frac{1}{4}(1-f_2)^2(2W(h-2, f_1, f_2) + 2L(h-2, f_1, f_2)) \\
&\quad + \frac{1}{4}(1-f_2)^2(W(h-2, f_1, f_2) + 2L(h-2, f_1, f_2)) \\
&\quad + \frac{1}{4}(1-f_2)^2(W(h-2, f_1, f_2) + 2L(h-2, f_1, f_2)) \\
&\quad + \frac{1}{4}(1-f_2)^2 2L(h-2, f_1, f_2) \\
&= 2L(h-2, f_1, f_2)) + (1-f_2)W(h-2, f_1, f_2).
\end{aligned}
$$


1
1
1
w1: 1 1 1 1 $f_1$

(a)


1
0
1
w2: 0 0 1 1 $0.5(1-f_1)f_2$
w3: 1 0 1 1 $0.25(1-f_1)(1-f_2)$
w4: 0 1 1 1 $0.25(1-f_1)(1-f_2)$

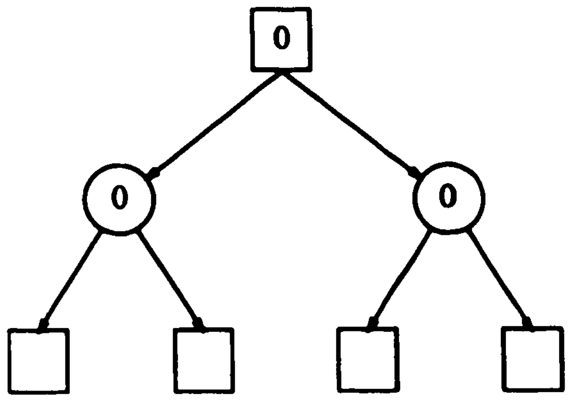
(b)



| | | | | | |
|---|---|---|---|---|---|
| *w5*: | 1 | 1 | 0 | 0 | $0.5(1-f_1)f_2$ |
| *w6*: | 1 | 1 | 1 | 0 | $0.25(1-f_1)(1-f_2)$ |
| *w7*: | 1 | 1 | 0 | 1 | $0.25(1-f_1)(1-f_2)$ |

(c)

**Fig 4.2 The Computation for $W(h, f_1, f_2)$**



| | | | | | |
|---|---|---|---|---|---|
| *l1*: | 0 | 0 | 0 | 0 | $f_2^2$ |
| *l2*: | 1 | 0 | 0 | 0 | $0.5(1-f_2)f_2$ |
| *l3*: | 0 | 1 | 0 | 0 | $0.5(1-f_2)f_2$ |
| *l4*: | 0 | 0 | 1 | 0 | $0.5(1-f_2)f_2$ |
| *l5*: | 0 | 0 | 0 | 1 | $0.5(1-f_2)f_2$ |
| *l6*: | 1 | 0 | 1 | 0 | $0.25(1-f_2)^2$ |
| *l7*: | 1 | 0 | 0 | 1 | $0.25(1-f_2)^2$ |
| *l8*: | 0 | 1 | 1 | 0 | $0.25(1-f_2)^2$ |
| *l9*: | 0 | 1 | 0 | 1 | $0.25(1-f_2)^2$ |

Fig. 4.3 The Computation for $L(h, f_1, f_2)$

For the sake of notation simplicity, let $W_k = W(h, f_1, f_2)$ and $L_k \quad (h, f_1, f_2)$, where $h = 2k$. Then, the recursive equations for $W_k$ and $L_k$ are

$$\begin{cases} W_k = (2 + \frac{1}{4}(1-f_1)(1-f_2))W_{k-1} + \frac{1}{2}(1-f_1)L_{k-1} \\ L_k = 2L_{k-1} + (1-f_2)W_{k-1}. \end{cases} \tag{4.3}$$

The equations of (4.3) can be solved by cases. If $f_1 = 1, f_2 \neq 1$, the first equation of (4.3) implies

$$W_k = 2W_{k-1}.$$

Since $W_0 = 1$, we have $W_k = 2^k$. Replacing $W_k$ in the second equation, we have

$$\begin{aligned} L_k &= 2L_{k-1} + (1-f_2)2^{k-1} \\ &= 2^2 L_{k-2} + (1-f_2)2^{k-1} + (1-f_2)2^{k-1} \\ &\dots \\ &= 2^k L_0 + k(1-f_2)2^{k-1} \\ &= 2^k(1 + \frac{k}{2}(1-f_2)). \end{aligned}$$

If $f_1 \neq 1, f_2 = 1$, the second equation of (4.3) implies

$$L_k = 2L_{k-1}.$$

Since $L_0 = 1$, we have $L_k = 2^k$. Replacing $L_k$ in the first equation, we have

$$\begin{aligned} W_k &= 2W_{k-1} + (1-f_1)2^{k-2} \\ &= 2^2 W_{k-2} + (1-f_1)2^{k-2} + (1-f_1)2^{k-2} \\ &\dots \\ &= 2^k W_0 + k(1-f_2)2^{k-2} \\ &= 2^k(1 + \frac{k}{4}(1-f_2)). \end{aligned}$$

If $f_1 = f_2 = 1$, we have

$$W_k = 2W_{k-1}, L_k = 2L_{k-1}.$$

Since $W_0 = L_0 = 1$, the solution for (4.3) is

$$W_k = 2^k, L_k = 2^k.$$

In the following, it is assumed that $f_1 \neq 1$ and $f_2 \neq 1$. By the first equation of (4.3), the following equalities can be derived

$$L_{k-1} = (\frac{1}{2}(1-f_1))^{-1}(W_k - (2 + \frac{1}{4}(1-f_1)(1-f_2))W_{k-1}),$$

and

$$L_k = (\frac{1}{2}(1-f_1))^{-1}(W_{k+1} - (2 + \frac{1}{4}(1-f_1)(1-f_2))W_k).$$

Replacing $L_{k-1}$ and $L_k$ in the second equation of (4 3), the recursive equation for $W_k$ is

$$W_{k+1} - (2 + \frac{1}{4}(1-f_1)(1-f_2))W_k$$
$$= 2W_k - 2(2 + \frac{1}{4}(1-f_1)(1-f_2))W_{k-1} + \frac{1}{2}(1-f_1)(1-f_2)W_{k-1},$$

which is

$$W_{k+1} - (4 + \frac{1}{4}(1-f_1)(1-f_2))W_k + 4W_{k-1} = 0. \tag{4.4}$$

By the second equation of (4.3), the following equalities can be derived

$$W_{k-1} = (1-f_2)^{-1}(L_k - 2L_{k-1}),$$

and

$$W_k = (1-f_2)^{-1}(L_{k+1} - 2L_k).$$

Replacing $W_{k-1}$ and $W_k$ in the first equation of (4.3), the recursive equation for $L_k$ is

$$L_{k+1} - 2L_k$$
$$= (2 + \frac{1}{4}(1-f_1)(1-f_2))(L_k - 2L_{k-1}) + \frac{1}{2}(1-f_1)(1-f_2)L_{k-1},$$

which is

$$L_{k+1} - (4 + \frac{1}{4}(1-f_1)(1-f_2))L_k + 4L_{k-1} = 0. \tag{4.5}$$

Since the boundary conditions for recursive equations (4.4) and (4.5) are

$$W_0 = 1, W_1 = 2 + \frac{1}{4}(1-f_1)(1-f_2) + \frac{1}{2}(1-f_1)$$

and

$$L_0 = 1, L_1 = 2 + (1-f_2),$$

respectively, these equations can be solved and the solutions are as follows. When

$f_1 \neq 1$ and $f_2 \neq 1$, the solution for $W_k$ is

$$W_k = Ax_1^k + Bx_2^k,$$

where

$$x_1 = \frac{1}{8}\left[16 + (1-f_1)(1-f_2) + (((1-f_1)(1-f_2))^2 + 32(1-f_1)(1-f_2))^{1/2}\right]$$

$$x_2 = \frac{1}{8}\left[16 + (1-f_1)(1-f_2) - (((1-f_1)(1-f_2))^2 + 32(1-f_1)(1-f_2))^{1/2}\right]$$

$$A = \frac{1}{2} + \frac{(1-f_1)(5-f_2)}{2((1-f_1)(1-f_2))^2 + 32(1-f_1)(1-f_2))^{1/2}},$$

$$B = \frac{1}{2} - \frac{(1-f_1)(5-f_2)}{2((1-f_1)(1-f_2))^2 + 32(1-f_1)(1-f_2))^{1/2}};$$

the solution for $L_k$ is

$$L_k = A_1x_1^k + B_1x_2^k,$$

where

$$A_1 = \frac{1}{2} + \frac{(1-f_1)(7+f_2)}{2((1-f_1)(1-f_2))^2 + 32(1-f_1)(1-f_2))^{1/2}},$$

$$B_1 = \frac{1}{2} - \frac{(1-f_1)(7+f_2)}{2((1-f_1)(1-f_2))^2 + 32(1-f_1)(1-f_2))^{1/2}}.$$

The details of solving these recursive equations are described in our Appendix to this chapter.

## 5. Best Case Analysis

Let $w_k$ ($l_k$) be the lower bound for the number of terminal nodes in a bi-valued uniform tree of degree $n$ and height $h$ that are visited by the alpha-beta pruning algorithm when the root node has value 1 (0, respectively).

*Lemma* 4.4 For a bi-valued uniform tree of degree $n$ and height $h$, the alpha-beta algorithm visits at least $n^{\lfloor h/2 \rfloor}$ terminal nodes, i.e.,

$$\min (w_h, l_h) = n^{\lfloor h/2 \rfloor}.$$

*Proof.* When height $h = 0$, the root node is the only terminal node, therefore,

$$w_0 = l_0 = 1.$$

If $h > 0$ and the root node $g$ is a Max node, one successor with utility 1 can establish $\Psi(g) = 1$. It is obvious that if $g$ has a utility 1, the minimum number of terminal nodes will be visited by the alpha-beta algorithm only when $g.1$ has utility 1. This observation implies

$$w_h = w_{h-1}.$$

To establish $\Psi(g) = 0$, all successor nodes must have utility 0. By Lemma 4.3, all successor nodes must be visited by the alpha-beta pruning algorithm, and this fact implies

$$l_h = nl_{h-1}.$$

If $h > 0$ and the root node $g$ is a Min node, then one successor with utility 0 can establish $\Psi(g) = 0$. This fact implies that if and only if $g.1$ has utility 0, is it possible that the minimum number of terminal nodes be visited by the alpha-beta algorithm. This observation implies

$$l_h = l_{h-1}.$$

To establish $\Psi(g) = 1$, all successor nodes must have utility 1. By an observation similar to Lemma 4.3, all successor nodes must be visited by the alpha-beta pruning algorithm, and this fact implies

$$w_h = nw_{h-1}.$$

The above recursive equations for $w_h$ and $l_h$ imply that $w_h = n^{h/2}$, $l_h = n^{h/2}$ when $h$ is even, and $w_h = n^{(h+1)/2}$, $l_h = n^{(h-1)/2}$ when $h$ is odd, and the lemma follows. □

By the proof, it can be seen that the lower bound is tight since the alpha-beta algorithm visits this number of terminal nodes in some bi-valued uniform tree of degree $n$ and height $h$.

By the solution of (4.3), which is presented in Section 4, it can be seen that only when $f_1 = f_2 = 1$, the average number of terminal nodes visited by the alpha-beta algorithm in a bi-valued binary node-dependent minimax game tree attains this smallest one.

*Theorem* 4.1 For a bi-valued binary dependent game tree of even height with $0 < p_0 < 1$, the alpha-beta algorithm visits the minimum average number of terminal nodes in even height if and only if $f_1 = f_2 = 1$.

□

## 6. Numerical Results

We now turn back to the bi-valued binary game trees. As shown in Section 3 of Chapter 2, for a probability $p_0$, there is an infinite number of value pairs for $f_1$ and $f_2$ which force both the root and the terminal nodes in a dependent minimax game tree with parameters $p_0$, $f_1$ and $f_2$ have the probability $p_0$ to take value 1. Also shown there is the fact that these values of $f_1$ and $f_2$ are positively correlated, i.e., for the value pairs, a larger value of $f_1$ corresponds to a larger value of $f_2$.

The first column in the following tables lists some of these value pairs for $f_1$ and $f_2$ with respect to $p_0 = 1/3$, $1/2$, $(\sqrt{5} - 1)/2$, respectively, in increasing order. The solution of recursive equations (4.3) relates the pruning efficiency of the alpha-beta algorithm to the node-value dependence in a game tree. The remaining columns list the expected number $N_k$ of terminal nodes in a dependent game tree of height $h = 2k$ with parameters $p_0$, $f_1$ and $f_2$ that are visited by the alpha-beta algorithm. These expected numbers are calculated by the solutions of $W_k$ and $L_k$, presented in Section 4, and formula

$$N_k = p_0 W_k + (1-p_0)L_k.$$

In Table 4.1, there are thirteen rows, which correspond to thirteen pairs of values of dependent factors $f_1$ and $f_2$, such that both the root node and the terminal nodes have the same probability 1/3 of being assigned value 1. It can be observed that as the dependent factors $f_1$ and $f_2$ increase in value, the average number $N_k$ of terminal nodes visited by the alpha-beta pruning algorithm in a dependent game tree of height $2k$ decreases. When $f_1 = f_2 = 1$, for a win (loss) root node, all the terminal nodes have the status of win (loss, respectively), the alpha-beta algorithm visits the least number of terminal nodes. In Table 4.2, the average numbers $N_k$ of terminal nodes visited by the alpha-beta pruning algorithm correspond to the dependent factors so that both the terminal nodes and the root node take the status win with a probability of 1/2; in Table 4.3, the numbers $N_k$ correspond to the dependent factors for both the root and terminal nodes to have a probability of $(\sqrt{5} - 1)/2$ to take the merit value 1. It is readily seen that in each column in these tables the average number of visited terminal nodes $N_k$ is in decreasing order. Based on these tables, we can draw the conclusion that the more positively correlated a dependent minimax game tree is, the fewer terminal nodes will be visited by the alpha-beta algorithm. This observation implies that the alpha-beta pruning efficiency is positively related to the node-value dependence presented in the game trees. As shown by Theorem 4.1, when $f_1 = f_2 = 1$, the alpha-beta algorithm visits the smallest number of terminal nodes.

Based on the data presented here, we can propose a conjecture: Given two dependent game trees of the same height that are described by parameters $p_0$, $f_1$, and $f_2$ and $p_0$, $f'_1$ and $f'_2$, respectively, if $f_1 \geq f'_1$ and $f_2 \geq f'_2$, the alpha-beta algorithm will visit fewer terminal nodes of the first tree than of the second tree.

Table 4.1 The Numbers of Visited Terminal Nodes ($P_k = p_0 = 0.333$)

| $f_1$ | $f_2$ | $N_6$ | $N_7$ | $N_8$ | $N_9$ | $N_{10}$ |
|---|---|---|---|---|---|---|
| 0.000 | 0.600 | 336.804 | 848.163 | 2130.255 | 5341.391 | 13378.685 |
| 0.102 | 0.633 | 294.676 | 727.925 | 1792.969 | 4407.904 | 10822.912 |
| 0.200 | 0.667 | 257.897 | 625.099 | 1510.439 | 3641.954 | 8768.616 |
| 0.294 | 0.700 | 226.134 | 538.195 | 1276.744 | 3021.797 | 7140.220 |
| 0.385 | 0.733 | 198.209 | 463.350 | 1079.583 | 2509.253 | 5821.685 |
| 0.472 | 0.767 | 173.711 | 398.933 | 913.157 | 2084.982 | 4751.422 |
| 0.556 | 0.800 | 152.185 | 343.456 | 772.711 | 1734.173 | 3884.346 |
| 0.636 | 0.833 | 133.305 | 295.679 | 653.990 | 1443.183 | 3178.702 |
| 0.714 | 0.867 | 116.289 | 253.358 | 550.688 | 1194.555 | 2586.827 |
| 0.789 | 0.900 | 101.179 | 216.417 | 462.096 | 985.154 | 2097.426 |
| 0.862 | 0.933 | 87.495 | 183.475 | 384.343 | 804.363 | 1681.937 |
| 0.932 | 0.967 | 75.205 | 154.287 | 316.414 | 648.686 | 1329.449 |
| 1.000 | 1.000 | 64.000 | 128.000 | 256.000 | 512.000 | 1024.000 |

Table 4.2 The Numbers of Visited Terminal Nodes ($P_k = p_0 = 0.5$)

| $f_1$ | $f_2$ | $N_6$ | $N_7$ | $N_8$ | $N_9$ | $N_{10}$ |
|---|---|---|---|---|---|---|
| 0.000 | 0.333 | 409.457 | 1093.859 | 2920.010 | 7791.515 | 20785.256 |
| 0.073 | 0.367 | 365.669 | 959.918 | 2517.813 | 6600.938 | 17300.845 |
| 0.143 | 0.400 | 327.544 | 845.400 | 2180.099 | 5619.049 | 14478.126 |
| 0.209 | 0.433 | 294.410 | 747.522 | 1896.264 | 4807.583 | 12184.323 |
| 0.273 | 0.467 | 264.730 | 661.266 | 1650.203 | 4115.608 | 10260.308 |
| 0.333 | 0.500 | 239.133 | 588.083 | 1444.834 | 3547.465 | 8706.293 |
| 0.391 | 0.533 | 216.305 | 523.841 | 1267.392 | 3064.316 | 7405.569 |
| 0.447 | 0.567 | 195.803 | 467.000 | 1112.744 | 2649.588 | 6305.986 |
| 0.500 | 0.600 | 177.883 | 418.075 | 981.669 | 2303.461 | 5402.339 |
| 0.551 | 0.633 | 161.870 | 374.983 | 867.898 | 2007.415 | 4640.765 |
| 0.600 | 0.667 | 147.474 | 336.763 | 768.371 | 1752.019 | 3992.932 |
| 0.647 | 0.700 | 134.688 | 303.302 | 682.488 | 1534.813 | 3449.933 |
| 0.692 | 0.733 | 123.256 | 273.780 | 607.722 | 1348.263 | 2989.882 |
| 0.736 | 0.767 | 112.800 | 247.118 | 541.073 | 1184.139 | 2590.472 |
| 0.778 | 0.800 | 103.510 | 223.740 | 483.403 | 1044.020 | 2254.055 |
| 0.818 | 0.833 | 95.218 | 203.123 | 433.165 | 923.458 | 1968.190 |
| 0.857 | 0.867 | 87.636 | 184.493 | 388.305 | 817.092 | 1719.034 |
| 0.895 | 0.900 | 80.754 | 167.789 | 348.582 | 724.086 | 1503.918 |
| 0.931 | 0.933 | 74.636 | 153.108 | 314.062 | 644.182 | 1321.222 |
| 0.966 | 0.967 | 69.062 | 139.877 | 283.300 | 573.773 | 1162.051 |
| 1.000 | 1.000 | 64.000 | 128.000 | 256.000 | 512.000 | 1024.000 |

Table 4.3 The Numbers of Visited Terminal Nodes ($P_k = p_0 = 0.618$)

| $f_1$ | $f_2$ | $N_6$ | $N_7$ | $N_8$ | $N_9$ | $N_{10}$ |
|---|---|---|---|---|---|---|
| 0.055 | 0.133 | 442.547 | 1217.544 | 3349.377 | 9213.380 | 25343.176 |
| 0.117 | 0.167 | 397.765 | 1076.272 | 2911.938 | 7878.126 | 21313.416 |
| 0.176 | 0.200 | 358.775 | 955.301 | 2543.537 | 6772.119 | 18030.374 |
| 0.232 | 0.233 | 324.532 | 850.702 | 2229.961 | 5845.427 | 15322.685 |
| 0.284 | 0.267 | 294.641 | 760.691 | 1964.011 | 5070.970 | 13093.182 |
| 0.334 | 0.300 | 268.201 | 682.274 | 1735.811 | 4416.456 | 11237.317 |
| 0.382 | 0.333 | 244.612 | 613.300 | 1537.954 | 3857.105 | 9674.082 |
| 0.427 | 0.367 | 223.745 | 553.069 | 1367.449 | 3381.514 | 8362.886 |
| 0.470 | 0.400 | 205.261 | 500.447 | 1220.531 | 2977.366 | 7264.043 |
| 0.511 | 0.433 | 188.756 | 454.053 | 1092.661 | 2630.171 | 6332.352 |
| 0.551 | 0.467 | 173.641 | 412.088 | 978.442 | 2323.956 | 5521.096 |
| 0.588 | 0.500 | 160.519 | 376.089 | 881.650 | 2067.651 | 4850.485 |
| 0.624 | 0.533 | 148.568 | 343.693 | 795.587 | 1842.501 | 4268.540 |
| 0.658 | 0.567 | 137.851 | 314.943 | 720.029 | 1647.000 | 3768.858 |
| 0.691 | 0.600 | 128.161 | 289.255 | 653.313 | 1476.420 | 3338.050 |
| 0.723 | 0.633 | 119.338 | 266.119 | 593.885 | 1326.159 | 2962.799 |
| 0.753 | 0.667 | 111.465 | 245.667 | 541.858 | 1195.905 | 2640.780 |
| 0.782 | 0.700 | 104.352 | 227.389 | 495.863 | 1082.005 | 2362.257 |
| 0.810 | 0.733 | 97.889 | 210.945 | 454.900 | 981.588 | 2119.203 |
| 0.837 | 0.767 | 91.995 | 196.090 | 418.242 | 892.579 | 1905.821 |
| 0.863 | 0.800 | 86.685 | 182.843 | 385.885 | 814.811 | 1721.286 |
| 0.888 | 0.833 | 81.882 | 170.976 | 357.177 | 746.472 | 1560.673 |
| 0.912 | 0.867 | 77.534 | 160.333 | 331.664 | 686.295 | 1420.530 |
| 0.935 | 0.900 | 73.540 | 150.896 | 309.269 | 633.995 | 1299.934 |
| 0.957 | 0.933 | 70.149 | 142.519 | 289.584 | 588.470 | 1195.967 |
| 0.979 | 0.967 | 66.891 | 134.786 | 271.604 | 547.316 | 1102.946 |
| 1.000 | 1.000 | 64.000 | 128.000 | 256.000 | 512.000 | 1024.000 |

## 7. Analysis for the More General Dependent Game Trees

The analysis method just presented is a general one in that it can be applied to the multi-valued $n$-ary dependent game trees. We will show that the recursive equations for the average number of terminal nodes visited by the alpha-beta algorithm in these trees can also be derived. Based on these equations, the calculation of this number is reduced to matrix multiplication. If the $n$-ary dependent game tree is bi-valued, the recursive equations can be solved.

First, assume a bi-valued $n$-ary dependent game tree $G$ with height $h = 2k$ and $n \geq 2$, where the merit value of the root node is described by a probability $p_0$. For an interior Max node $g$, the successor values $\Phi(g_i)$ $(1 \leq i \leq n)$ are randomly determined by a set of conditional probabilities

$$\Pr[\Phi(g_1)=v_1, \Phi(g_2)=v_2, \cdots \Phi(g_n)=v_n \mid \Phi(g)=v_0]; \qquad (4.6)$$

for an interior Min node $g$, the values $\Phi(g_i)$ $(1 \leq i \leq n)$ are randomly determined by another set of conditional probabilities

$$\Pr[\Phi(g_1)=v'_1, \Phi(g_2)=v'_2, \ldots, \Phi(g_n)=v'_n \mid \Phi(g)=v'_0], \qquad (4.7)$$

where $(v_1, v_2, \ldots, v_n, v_0)$ and $(v'_1, v'_2, \ldots, v'_n, v'_0) \in S^{n+1}, S = \{0, 1\}$. As in Section 4, define $W_k$ $(L_k)$ as the average number of terminal nodes in $G$ visited by the alpha-beta algorithm with window [0, 1] when the root node of $G$ has a merit value 1 (0, respectively). When the root node $g$ of $G$ has a merit value $\Phi(g) = 1$, the probabilities for each possible merit value assignment of the children of $g$ can be determined using the conditional probabilities of (4.6). The probabilities $p_{1i}$ for merit value assignments $wi$ of the successors of the children of node $g$ can be determined by the conditional probabilities of (4.7). For each such merit value assignment $wi$, the nodes at height $2k-2$ visited by the alpha-beta algorithm and their merit values can be determined. If there are $r_{1i}$ win nodes and $s_{1i}$ loss nodes at height $2k-2$ to be visited by the alpha-beta algorithm for the assignment $wi$, the following equation can be derived

$$W_k = \sum_i p_{1i}(r_{1i} W_{k-1} + s_{1i} L_{k-1}),$$

which is equivalent to

$$W_k = a_0 W_{k-1} + b_0 L_{k-1}, \qquad (4.8)$$

for some real number $a_0$ and $b_0$. Similarly, If the root node $g$ of $G$ has a merit value

$\Phi(g) = 0$, an equation

$$L_k = a_1 W_{k-1} + b_1 L_{k-1}, \tag{4.9}$$

where $a_1$ and $b_1$ are real numbers, can be derived. Note that recursive equations (4.8) and (4.9) can be solved by the method presented in Section 4.

Now assume that $G$ is a multi-valued $n$-ary dependent minimax game tree, its root node value is described by a probability distribution $P_0(v)$ with $v \in S$, and for an interior Max node $g$, the successor values $\Phi(g_i)$ $(1 \le i \le n)$ are randomly determined by a set of conditional probabilities

$$\Pr[\Phi(g_1)=v_1, \Phi(g_2)=v_2, \cdots \Phi(g_n)=v_n \mid \Phi(g)=v_0]; \tag{4.10}$$

and for an interior Min node $g$, the values $\Phi(g_i)$ $(1 \le i \le n)$ are randomly determined by another set of conditional probabilities

$$\Pr[\Phi(g_1)=v'_1, \Phi(g_2)=v'_2, \ldots, \Phi(g_n)=v'_n \mid \Phi(g)=v'_0], \tag{4.11}$$

where $(v_1, v_2, \ldots, v_n, v_0)$ and $(v'_1, v'_2, \ldots, v'_n, v'_0) \in S^{n+1}$, and the merit value set is $S = \{i_1, \cdots, i_q\}$ for some $q \ge 2$. Assume all the subranges of $[i_1, i_q]$ are listed as $[s_m, s'_m]$ with $1 \le m \le M$ and $i_1 \le s_m < s'_m \le i_q$. Let $[s_1, s'_1] = [i_1, i_q]$, and let $V_k(j, m)$ with $1 \le j \le q$ and $1 \le m \le M$ denote the average number of terminal nodes visited by the alpha-beta algorithm with window $[s_m, s'_m]$ in $G$ when the root node $g$ of $G$ has a merit value $i_j$. By the conditional probabilities of (4.10), the probability for each possible merit value assignment of the children of $g$ can be determined when the merit value of the root node $g$ is given. Then the probabilities for the merit value assignments of the successors of the children of node $g$ can be determined by the conditional probabilities of (4.11). For each such merit value assignment, the nodes at height $2(k-1)$ visited by the alpha-beta algorithm along with the visiting windows and their merit values can be determined. In this way an equation of the following form can be established

$$V_k(j,m) = \sum_{1 \le t \le q,\ 1 \le r \le M} J_{jmtr} V_{k-1}(t,r)$$

for $1 \le j \le q$, $1 \le m \le M$, where $J_{jmtr}$ are real numbers. Define matrix $\mathbf{V}_k = [V_k(1,1), \cdots, V_k(q,1), \cdots, V_k(1,M), \cdots, V_k(q,M)]^T$, vector $\mathbf{J}_{jm} = [J_{jm11}, \cdots, J_{jmq1}, \cdots, J_{jm1M}, \cdots, J_{jmqM}]$, and real matrix $\mathbf{J} = [\mathbf{J}_{11}, \cdots, \mathbf{J}_{q1}, \cdots, \mathbf{J}_{jM}, \cdots, \mathbf{J}_{jM}]^T$. Since $\mathbf{J}$ is independent of the height $2k$ of the game tree, we have

$$\mathbf{V}_k = \mathbf{J}\,\mathbf{V}_{k-1}.$$

Since the boundary condition of the equation is $\mathbf{V}_0 = [1, \cdots, 1]^T$,

$$\mathbf{V}_k = \mathbf{J}^k \mathbf{V}_0.$$

The average number $v_k$ of terminal nodes visited by the alpha-beta algorithm in the multi-value $n$-ary dependent game tree of height $h = 2k$ can be calculated by

$$v_k = \mathbf{P}\mathbf{J}^k \mathbf{V}_0,$$

where vector $\mathbf{P} = [P_0(i_1), \cdots, P_0(i_q), 0, \cdots, 0]$. In other words, the average number of visited terminal nodes can be calculated by matrix multiplications.

The analysis method for dependent game tree has been generalized for multi-value $n$-ary trees, where $n > 2$. These generalized cases are defined by more parameters than the bi-valued binary tree. Given these parameters, we can also produce some tables similar to Tables 4.1-4.3. These tables can be used to analyze the performance of the alpha-beta algorithm. The analysis could provide some insight about the speed of a computer game playing program that emploies the alpha-beat algorithm as its inner search loop procedure.

## Appendix to Chapter 4: Solutions for $W_k$ and $L_k$

There are two methods, charateristic root and generating function, for solving equation (4.4), which can be found in the book of Mott *et al.* [18]. First, its solution can be related to the solutions of the following equation of $x$:

$$x^2-(4+\frac{1}{4}(1-f_1)(1-f_2))x+4=0,$$

which are

$$x_1=\frac{1}{8}\left[16+(1-f_1)(1-f_2)+(((1-f_1)(1-f_2))^2+32(1-f_1)(1-f_2))^{1/2}\right]$$
$$x_2=\frac{1}{8}\left[16+(1-f_1)(1-f_2)-(((1-f_1)(1-f_2))^2+32(1-f_1)(1-f_2))^{1/2}\right].$$

Therefore, the solution for $W_k$ is

$$W_k=Ax_1^k+Bx_2^k,$$

where $A$ and $B$ are constants to be determined by the boundary conditions

$$W_0=1, W_1=2+\frac{1}{4}(1-f_1)(3-f_2).$$

The constants $A$ and $B$ can be solved by

$$\begin{cases} A+B=1 \\ Ax_1+Bx_2=2+\frac{1}{4}(1-f_1)(3-f_2). \end{cases} \qquad (4.12)$$

The solution to (4.12) is

$$\begin{cases} A=\frac{1}{2}+\frac{(1-f_1)(5-f_2)}{2((1-f_1)(1-f_2))^2+32(1-f_1)(1-f_2))^{1/2}} \\ B=\frac{1}{2}-\frac{(1-f_1)(5-f_2)}{2((1-f_1)(1-f_2))^2+32(1-f_1)(1-f_2))^{1/2}}. \end{cases}$$

Similarly, the solution for $L_k$ is

$$L_k=A_1x_1^k+B_1x_2^k,$$

$$\begin{cases} A_1 = \frac{1}{2} + \frac{(1-f_1)(7+f_2)}{2((1-f_1)(1-f_2))^2 + 32(1-f_1)(1-f_2))^{1/2}} \\ B_1 = \frac{1}{2} - \frac{(1-f_1)(7+f_2)}{2((1-f_1)(1-f_2))^2 + 32(1-f_1)(1-f_2))^{1/2}} \end{cases}$$

The second method for solving (4.4) is based on reducing tl order of the recursive equation (4.4). Note that a real number $C$ can be found so that equation (4.4) is equivalent to

$$W_{k+1} - \frac{4}{C}W_k = C(W_k - \frac{4}{C}W_{k-1}).$$

In fact, $C$ is the solution of equation

$$C + \frac{2}{C} = 4 + \frac{1}{4}(1-f_1)(1-f_2). \tag{4.13}$$

The solutions to the above equation are

$$C_1 = \frac{1}{8}\left[16 + (1-f_1)(1-f_2) + (((1-f_1)(1-f_2))^2 + 32(1-f_1)(1-f_2))^{1/2}\right]$$
$$C_2 = \frac{1}{8}\left[16 + (1-f_1)(1-f_2) - (((1-f_1)(1-f_2))^2 + 32(1-f_1)(1-f_2))^{1/2}\right].$$

Therefore, equation (4.13) is equivalent to

$$W_{k+1} - \frac{4}{C_1}W_k = C_1^k(W_1 - \frac{4}{C_1}W_0).$$

Since $\frac{4}{C_1} = C_2$, we have

$$W_{k+1} - C_2W_k = C_1^k(W_1 - C_2W_0).$$

Similarly,

$$L_{k+1} - C_2L_k = C_1^k(L_1 - C_2L_0).$$

By the boundary conditions, the above linear recursive equations can be solved, and the details are omitted here.

Chapter 5

# Probability-Based Game Tree Pruning

## 1. Probability Evaluation

In game tree search, the utilities of the "horizon" nodes or terminal nodes in the search tree are evaluated to gain the best available information. There have been three methods of describing the evaluation results. The simplest but most inflexible method uses point-values, such as material-balance, to measure the merit value of a position. In the second method, upper and lower bounds are used to describe the possible merit values [10]. Palay suggests using a distribution function to describe the possible locations of the merit value of a node in the game tree[11]. In contrast, the point-value representation of merit value totally ignores our uncertainty about the merit value. It forces us to make a conclusion on the merit value of a position even when it is unclear. Another representation, range, also suffers from similar problems. For example, two positions cannot be distinguished if their corresponding ranges are the same. Probability distribution is a good alternative for describing the possible locations of the merit value of a position. The game tree search with probabilities or probability distributions as evaluation results will be called probability-based game tree search.

In computer game playing, move choice, or *decision making*, is based on game tree search. The study of probability-based game tree search is essentially independent of the study of decision making. The former focuses on gaining more information with less cost, the latter, on analyzing the gained information. If the uncertainty about the merit value is to be reflected in the evaluation, which is described with probabilistic distribution or probabilities, the search process must back up distributions from terminal nodes to the root. In computer game playing, the best-first search strategies [10, 11] combine decision making and game tree search naturally; the depth-first game tree search leaves the problem of decision making open. The existing probability-based move choice criterion, like the function *dominance-level* employed by algorithm PB* [11], can be applied to the probabilistic distributions backed up by depth-first search; if the decision cannot be made, some heuristics, like iterative deepening [19] or some other selective deepening, can be used to guide the depth-first search to gain more information. As a matter of fact, because of its lower space requirement and lighter operating overhead, depth-first game tree search is almost universally used in computer game playing programs. The study of probability-based depth-first search could lead to some heuristics for selective deepening and so reduce the errors and costs in game tree search. The depth-first search of game trees where the terminal nodes are evaluated with probability distributions, offers an interesting study of its own.

Since alpha-beta pruning is such a popular and powerful technique, it is tempting to apply it in probability-based game tree search. All the prior discussions about alpha-beta pruning have been restricted to point-value game tree search. The problem of how to apply the alpha-beta pruning technique to probability-based game tree search will be addressed in this chapter. First, an evaluation representation method is presented, and a probability-based alpha-beta pruning scheme is proposed for that method. The result shows that even though the merit value of a node is described by probabilities rather than a point-value, $\alpha$-$\beta$ bounded windows can still be used to cut off the search of some nodes. It can be proven that probability-based alpha-beta pruning is optimal in the sense that for some ordering of the successor nodes in a search tree, it will search the least number of terminal nodes to get the probabilities that describe the root position of the search tree. It is shown that the probability-based alpha-beta algorithm can be viewed as a generalization of the alpha-beta pruning employed by point-value search algorithms[4]. Several variations or applications of the probability-based algorithm are also presented. One of them applies alpha-beta pruning to range-based game tree search. The heuristic information available at interior nodes of a search tree can also be used to improve the $\alpha$-$\beta$ bounds. It also shows how alpha-beta pruning can be incorporated into probability-based best-first search. Finally, the efficiency of this probability-based game tree pruning technique is tested with a C-language program which generates random trees.

## 2. A Representation Method for Probability Distributions

The representation method for probability distribution functions proposed by Palay [11] is to represent each distribution by a fixed set of points. This set of points can be chosen in one of two ways. One way of preselecting points is to choose a set of values from the domain of the distributions, and represent a distribution by the probabilities for these values. The second way is to choose a set of points from the range of the distribution [0, 1], and represent the distribution by the corresponding domain values.

In practice, an expert consulted for determining an evaluation function, or for some other expert system design, can only provide the probabilities for some individual possible outcomes, rather than a complete continuous probability distribution function. Based on this observation, the merit value $\Phi(g)$ of a position $g$ will be described as a discrete random variable, and probability functions, rather than prouability distributions, will be used to describe the evaluation results. The domain of the utility values is a finite integer interval $[-i, i]$, for an integer $i > 0$. The merit value $\Phi(g)$ of a position $g$ is described by a set of probabilities, or a probability function $P_g$, such that $P_g(v)$ is the probability that the merit value $\Phi(g)$ is equal to $v$, for any $v \in [-i, i]$. A list of integer-real number pairs,

$$<v_1, P_g(v_1)>, <v_2, P_g(v_2)>, \ldots, <v_k, P_g(v_k)>, \tag{5.1}$$

is used to represent the probability function. The data structure (5.1) is called a VP-list if $P_g(v_j) > 0$, for each $j = 1, \ldots, k$. In the following, the terms probability function and VP-list are used synonymously. Usually, the first components of the pairs in a VP-list are in the ascending order, i.e., $v_t < v_{t+1}$ for $t = 1, ..., k-1$, and the domain of the utility value of a position is supposed to be a finite integer range $[-i, i]$, where $i > 0$ is an integer.

There are several advantages when the evaluation results are represented as probability functions, or VP-lists. First, only the domain values for which the probabilities are positive will appear in a VP-list. This requirement can reduce the space used for representing probability distributions with a preselected set of domain values, since many probabilities may be 0 or 1. The following is an example of the probability distribution representation[11]. Given a distribution

$$F(v) = \begin{cases} 0 & \text{if } v < 0 \\ v/100 & \text{if } 0 \le v \le 100 \\ 1 & \text{if } v > 100, \end{cases} \tag{5.2}$$

assume that the domain of the distribution is the range from -1000 to 1000. For each value $v$ such that $v < 0$, the corresponding probability is zero; similarly, for each value $v$ such that $v > 100$, the corresponding probability is one. Therefore, if the presected set of domain values that are used to represent this distribution are evenly distributed in the range [-1000, 1000], for 95% of the points the distribution function will have the trivial value—0 or 1. To reduce this space waste, Palay suggests an improvement[11] that only the values $v$ for which the distribution $F(v)$ is greater th^n zero and less than 1 will be stored. In fact, this method may not reduce the potential wasted space in more general situations. For example, in the distribution

$$F(v) = \begin{cases} (v+1000)/10 & \text{if } v < -995 \\ 0.5 & \text{if } -995 \le v \le 995, \\ 0.5+(v-995)/10 & \text{if } v > 995, \end{cases}$$

for all the domain values $v$ except $v$ = -1000 and 1000, the corresponding distribution values $F(v)$ are greater than 0 and less than 1. In the probability function representation method, it is easy to recognize that for all the values $v$ between -995 and 995 the probabilities $P(v) = 0$, and these values are not included in the VP-list. Note that this kind of distribution could be used very often for computer game playing. For example, if the only feature that can be recognized for a game position is a trapped piece, since the material value $v'$ of this piece is fixed, the domain values for which the probabilities are not equal to 0 may be $-v'$ and 0 only, where the domain value 0 represents the estimation that the trapped piece will not be lost.

To keep enough information in a distribution or VP-list, the representation should be accurate, or it should include enough domain values for which the probabilities are not equal to 0. Making use of the distribution (5.2) as an example, suppose the preselected set of domain values for the distribution are all the integers that are divisible by 100. Since all the important information about the distribution occurs between 0 and 100, the actual nature of this distribution is lost; this fact is observed by Palay[11]. Although the effect of this problem can be alleviated by increasing the number of preselected domain values, the space requirement will also be increased and this method cannot solve the problem caused by the "blindly" preselected set of domain values. In a VP-list, since all the domain values correspond to positive probabilities, every pair in it contains some useful information. This fact implies that VP-list will make full use of the space occupied by it, the more items it has, the more information is included in it. The VP-list representation is also flexible. For example, if we want to reduce the number of pairs in a VP-list, we can compare the distance between each pair of values and combine two pairs $<v_i, p_i>$ and $<v_{i+1}, p_{i+1}>$ into $<(v_i+v_{i+1})/2, p_i + p_{i+1}>$, if their domain values $v_i$ and $v_{i+1}$ are the closest.

Another distribution representation method suggested by Palay[11] is choosing a set of points from the range [0.0, 1.0] of the distributions. This representation method also suffers from some problems of inaccuracy and inefficiency. In this representation, only the domain values for which the corresponding distribution has a value in the preselected set of range values will be kept. Therefore, when multiplying two different distributions to back up them, we have to make use of interpolation to calculate extra values. Interpolation will result in both inaccuracy and inefficiency.

By the above analysis, it can be seen that although the VP-list representation for probability functions uses two numbers for a domain value, because both of the two numbers play some meaningful role in probability back-up, this representation method provides a relatively accurate and efficient data structure to organize the probabilities.

The negamax game tree search method described in Section 2 of Chapter 1 will be exploited in this chapter, so that one player's winning will be another's loss, and *vice versa*. In probability-based game tree search, the operands are probability functions, and this reversed viewing method can be reflected by the function $P\_NEG$. Let $P_g$ be the VP-list describing the possible locations of the merit value $\Phi(g)$ of position $g$. The probability $P\_NEG(P_g)(v)$, for any $v \in [-i, i]$, is defined as the probability that the random variable $-\Phi(g)$ is equal to $v$:

$$P\_NEG(P_g)(v) = \Pr[-\Phi(g) = v].$$

Therefore, if a player is making use of a probability function $P$ to represent the merit value of a position, the opponent will make use of $P\_NEG(P)$ to describe that same position. By the definition of $P\_NEG$, it is easy to deduce the following formula:

$$P\_NEG(P)(v) = \Pr[\Phi(g) = -v] = P(-v)$$

for any $v \in [-i, i]$. The above equations imply that if a merit value is represented from the point of view of one player by a VP-list

$$<v_1, p_1>, <v_2, p_2>, \ldots, <v_k, p_k>,$$

then it will be described by the opponent as

$$<-v_k, p_k>, <-v_{k-1}, p_{k-1}>, \ldots, <-v_1, p_1>.$$

In other words, to get the probability function for the opponent, just change the sign of the domain value component of each pair in the VP-list.

Searching a game tree means backing up the probabilities of the terminal nodes to the root position. The back-up from its successor positions $g_1, g_2, \ldots g_n$ to a node $g$ can be described by function $P\_MAX$, the operands of which are also probability functions. For the probability functions $P_{g1}, P_{g2}, \ldots P_{gn}$, $P\_MAX$ can be defined in the following way:

When $n = 1$,

$$P_g = P\_MAX(P_{g1}) = P_{g1};$$

when $n = 2$, for any $v \in [-i, i]$,

$$\begin{aligned} P_g(v) &= P\_MAX(P_{g1}, P_{g2})(v) \\ &= P_{g1}(v)P_{g2}(v) + P_{g1}(v)\sum_{t=-i}^{v-1} P_{g2}(t) + P_{g2}(v)\sum_{t=-i}^{v-1} P_{g1}(t); \end{aligned}$$

when $n \geq 3$,

$$\begin{aligned} P_g &= P\_MAX(P_{g1}, P_{g2}, \ldots, P_{gn}) \\ &= P\_MAX(P\_MAX(P_{g1}, P_{g2}), \ldots, P_{gn}). \end{aligned}$$

Similar formulae can be used to define another function—$P\_MIN$, which is used to describe the complement operation of $P\_MAX$:

When $n = 1$,

$$P\_MIN(P_{g1}) = P_{g1};$$

when $n = 2$, for any $v \in [-i, i]$,

$$\begin{aligned} &P\_MIN(P_{g1}, P_{g2})(v) \\ &= P_{g1}(v)P_{g2}(v) + P_{g1}(v)\sum_{t=v+1}^{i} P_{g2}(t) + P_{g2}(v)\sum_{t=v+1}^{i} P_{g1}(t); \end{aligned}$$

when $n \geq 3$,

$$P\_MIN(P_{g1}, P_{g2}, \ldots, P_{gn}) = P\_MIN(P\_MIN(P_{g1}, P_{g2}), \ldots, P_{gn}).$$

## 3. Probability-Based Alpha-Beta Pruning

Two concepts will be used in the pruning process game trees that returns the same value as the exhausted search. The lower bound of a probability function $P$, denoted as $lower\_bound(P)$, is defined as the smallest domain value $v \in [-i, i]$ for which $P(v) \neq 0$; the upper bound, $upper\_bound(P)$, is the greatest $v \in [-i, i]$ for which $P(v) \neq 0$.

**Proposition** 5.1. Given two probability functions $P_1$ and $P_2$,

(1) if $lower\_bound(P_1) \geq upper\_bound(P_2)$, then

$$P\_MAX(P_1, P_2) = P_1;$$

(2) if

$$lower\_bound(P_1) < upper\_bound(P_2)$$

and

$$lower\_bound(P_2) < upper\_bound(P_1),$$

then

$$P\_MAX(P_1, P_2) \neq P_1$$

and

$$P\_MAX(P_1, P_2) \neq P_2.$$

*Proof.* By the definitions of $lower\_bound$, $upper\_bound$ and $P\_MAX$. □

Suppose the current state of a node $g$ is described by $P_g$ in the alpha-beta search and that of one successor node $g_j$ is described by $P_{g_j}$. The current state of $g$ will be described by

$$P\_MAX(P_g, P\_NEG(P_{g_j})).$$

The property of operation *P_MAX* presented in Proposition 5.1 suggests that only if $lower\_bound(P_g) \geq -lower\_bound(P_{g_j})$ can the search of the remaining successors of $g_j$ be cut off. Based on this observation, a probability-based alpha-beta algorithm can be designed.

The standard alpha-beta pruning technique analyzed in Chapter 4, which uses $\alpha$-$\beta$ bounded windows to limit the backed-up point-values, will be generalized for probability-based game tree search. The lower bounds of the probability functions that describe the current states of the ancestor nodes will be used in setting the cut-off bounds $\alpha$ and $\beta$ before searching a node; these bounds are also improved dynamically when searching the subtree rooted at the node.

**Algorithm 5.1.** Probability-Based Alpha-Beta Pruning Algorithm.

The recursive function P_AB($g$ : node; $\alpha$, $\beta$: integer) will return a VP-list, the upper bound of which is less than or equal to $\beta$ and the lower bound greater than or equal to $\alpha$. In this function, two local variables (TP0 and TP1, respectively) maintain the current state of position $g$ and that of the successor being explored. The special VP-list $<v, 1>$ with a single integer-real pair represents a probability function that takes value $v$ with probability 1.

```
function P_AB(g : node; α, β: integer): VP-list;
var
  TP0, TP1: VP-list;
  j: integer;
begin
1.  if g is a terminal node then
```

```
        return $P\_MIN(P\_MAX(P_g, <\alpha, 1>), <\beta, 1>)$;
      comment: Probability function $P_g$ is obtained by evaluating $g$
                from the corresponding player's viewpoint.
2.  determine the successor positions, $g_1, ..., g_n$, of $g$, where $n > 0$;
3.  TP0 := $<\alpha, 1>$;
4.  for $j$ := 1 to $n$ do
    begin
5.      TP1 := $P\_NEG$(P_AB($g_j$, $-\beta$, $-\alpha$));
6.      TP0 := $P\_MAX$(TP0, TP1);
7.      $\alpha$ := $lower\_bound$(TP0)
8.      if $\alpha = \beta$
          then return TP0;
    end;
9.  return TP0;
end.
```

The properties of function P_AB proposed in the following lemma, which generalizes the conditions (4.1), will be used to prove the above algorithm.

**Lemma 5.1.** If $P_g$ is the probability function that describes the strength value of a position $g$ in a negamax game tree search, and integers $\alpha < \beta$, then

P_AB($g$, $\alpha$, $\beta$) = $<\alpha, 1>$, if $upper\_bound(P_g) \le \alpha$;

P_AB($g$, $\alpha$, $\beta$) = $P\_MIN(P\_MAX(P_g, <\alpha, 1>), <\beta, 1>)$,

if $upper\_bound(P_g) > \alpha$ and $lower\_bound(P_g) < \beta$;

P_AB($g$, $\alpha$, $\beta$) = $<\beta, 1>$, if *lower_bound*($P_g$) $\geq \beta$.

*Proof.*

These three loop invariant statements can be summarized by the following generalization,

$$\text{P\_AB}(g, \alpha, \beta) = P\_MIN(P\_MAX(P_g, <\alpha, 1>), <\beta, 1>).$$

The separation of this statement is just for convenience in proving Theorem 5.1 later. Lemma 5.1 will be proved by induction on the height of position $g$ in the search tree.

If the height of $g$ is 0, statement 1 will return a probability function

$$P\_MIN(P\_MAX(P_g, <\alpha, 1>), <\beta, 1>).$$

Therefore, the loop invariant is true for a terminal node $g$.

Suppose that the lemma is true for all nodes of height less than $h$, and $g$ is a position whose height in the search tree is $h$. In the **for** statement of function P_AB, statement 5 returns a probability function

$$\text{TP1} = P\_NEG(P\_MIN(P\_MAX(P_{g_j}, <-\beta, 1>), <-\alpha, 1>))$$

$$= P\_MIN(P\_MAX((P\_NEG(P_{g_j})), <\alpha, 1>), <\beta, 1>)).$$

If *lower_bound*(TP1) is equal to $\beta$ for some $j$, by the definition of the backed-up strength value, we must have *lower_bound*($P_g$) $\geq \beta$. Therefore,

$$\text{P\_AB}(g, \alpha, \beta) = P\_MIN(P\_MAX(P_g, <\alpha, 1>), <\beta, 1>)$$

$$= <\beta, 1>.$$

This justifies the return of statement 8. Otherwise, we can prove

$$P\_MIN(P\_MAX(P_g, <\alpha, 1>), <\beta, 1>)$$

$$= P\_MIN(P\_MAX(P\_MAX(P\_NEG(P_{g1}), \ldots, P\_NEG(P_{gn})), <\alpha, 1>), <\beta, 1>)$$

$$= P\_MAX(P\_NEG(P\_MIN(P\_MAX(P_{g1}, <-\beta, 1>), <-\alpha_1, 1>)), \ldots,$$

$$P\_NEG(P\_MIN(P\_MAX(P_{gn}, <-\beta, 1>), <-\alpha_n, 1>))),$$

for any integers $\alpha_1, \ldots, \alpha_n$, where $\alpha = \alpha_1 \le \alpha_2 \le \ldots \le \alpha_n = \max(lower\_bound(P_g), \alpha)$. Hence the lemma is proved. □

By the above lemma, given a position $g$ as the root of a search tree, probability function $P_g$ can be calculated by calling

$$\text{P\_AB}(g, -i, +i).$$

**Theorem** 5.1. For any position $g$, function call P_AB($g$, $-i$, $+i$) will return a probability function that describes the strength value of $g$. □

Note that in function P_AB we can replace statement 1 by

```
if $g$ is a terminal node then
    return $P_g$,
```

provided we change statement 8 into

```
if $\alpha \ge \beta$
  then return TP0.
```

But statement 1

```
return $P\_MIN(P\_MAX((P_p), <\alpha, 1>), <\beta, 1>)$
```

made the proof of Lemma 5.1 easier, and it usually returns a VP-list with tighter bounds.

## 4. A Generalization of Standard Alpha-Beta Pruning

Probability-based alpha-beta pruning algorithm, in fact, is a generalization of the standard alpha-beta algorithm analyzed by Knuth and Moore[4]. If the evaluation result for each terminal node $g$ is a point-value, i.e.

$$P_g(v) = 1$$

for some domain value $v$, then the function P_AB has a form that is essentially the same as the algorithm F2 encoded by Knuth and Moore[4]. Therefore, our probability-based alpha-beta algorithm will have similar properties to the point-valued version. In particular, since for any search tree there always exists an ordering of nodes for which the alpha-beta algorithm examines the fewest terminal nodes, an equivalent property must exist for the probability-based version.

**Theorem 5.2.** Probability-based pruning is optimum in the sense that for any search tree and any algorithm, denoted as algorithm X, that computes the probability function for the root position, there is a way of ordering successor nodes so that every terminal node examined by the probability-based alpha-beta pruning method under this reordering is also examined by the algorithm X.

Before proving the theorem, let us introduce a symbol $\lambda$ into the functions *P_NEG*, *P_MAX* and *P_MIN*. The symbol $\lambda$ stands for undetermined value for which the expressions backed up by the game tree search is simplified by the following equations:

$$\lambda \times r \Rightarrow \lambda, \lambda \times \lambda \Rightarrow \lambda,$$

$$\lambda + t \Rightarrow \lambda, \lambda + \lambda \Rightarrow \lambda,$$

where $\times$ and $+$ are the arithmetic multiplication and addition, $r$ is a non-zero real number, and $t$ is any real number. When the symbol $\lambda$ is multiplied by 0, the result is 0 and the symbol $\lambda$ will not be rolled up. Then the "probability function" $P_g$ for a node $g$ in the given search tree will be defined as follows:

If $g$ is a terminal node left unexamined by algorithm X, for each value $v \in [-i, i]$

$$P_g(v) = \lambda;$$

if $g$ is an examined terminal node, then $P_g$ is determined by the evaluation function of algorithm X;

if $g$ is an interior node, then

$$P_g = P\_MAX(P\_NEG(P_{g1}), \dots, P\_NEG(P_{gn})).$$

If there is only one value $v \in [-i, i]$ for which $P_g(v) = \lambda$, then let

$$P_g(v) = 1 - \sum_{j \neq v} P_g(j).$$

By induction on the height of a node $g$ in the given search tree, we can prove that if function $P_g(v) = \lambda$ for some $v \in [-i, i]$, then different probability functions can be assigned to the unexamined terminal nodes to get different backed-up probability functions for the node $g$. The appearance of symbol $\lambda$ in a probability function $P_g$ means that algorithm X cannot solve the game tree rooted at node $g$, because by assigning different probabilities to unexamined terminal nodes, $P_g$ will be changed.

For each such generalized probability function $P_g$, two integer ranges, $C_g$ and $U_g$, can be defined so that the lower bound of $C_g$ is the minimum domain value $v$ for which $P_g(v) \neq \lambda$ and $P_g(v) > 0$, and the upper bound of $C_g$ is the maximum domain value $v$ for which $P_g(v) \neq \lambda$ and $P_g(v) > 0$. Similarly, the lower bound of $U_g$ is the minimum domain value $v$ for which $P_g(v) = \lambda$, the upper bound of $U_g$ is the maximum domain value $v$ for which $P_g(v) = \lambda$. Note that for a position $g$, one, but not both, of the ranges $C_g$ and $U_g$ may be empty.

**Lemma** 5.2. For a node $g$, if both the ranges $C_g$ and $U_g$ are not empty, then one of the following two situations must be true.

$$lower\_bound(C_g) > upper\_bound(U_g),$$

or

$$lower\_bound(U_g) > upper\_bound(C_g).$$

*Proof.*

By induction on the height of the position $g$ in the given game tree, and the details are omitted here. $\square$

The above lemma characterizes the structure of the probability function $P_g$ for any position $g$. In fact, we can prove that if the range $U_g$ is nonempty, then for any $v \in U_g$, we have

$$P_g(v) = \lambda.$$

For any such value $v$, $P_g(v)$ can assume at least two (in fact, infinitely many) different probabilities by independently varying the probability function of an unexamined terminal node.

According to the ranges $U_g$ and $C_g$, we will say that the position $g$ is *solved*, *semi-solved* or *live* if the range $U_g$ is empty, not empty and not equal to $[-i, i]$, or equal to $[-i, i]$, respectively. Note that the algorithm X computes the probability function for the strength value of a position only if the status of the corresponding node is solved.

**Lemma** 5.3. For any solved interior node $g$ in the given tree, one of its successors $g_t$ must be a solved node such that the upper bound of $C_{g_t}$ is less than or equal to the lower bound of $U_{g_j}$, for all $j \neq t$.

*Proof.*

Without loss of generality, suppose node $g$ has only two successors, $g_1$ and $g_2$. First, let us assume both ranges $U_{g1}$ and $U_{g2}$ are non-empty. We will show that this assumption will lead to a conclusion that node $g$ is not a solved position.

Let $U_{g1} = [l_1, u_1]$ and $U_{g2} = [l_2, u_2]$, and suppose $u_1 \leq u_2$. Then

$$P\_MAX(P\_NEG(P_{g1}), P\_NEG(P_{g2}))(v) = \lambda$$

for any $v \in [-u_1, -l_1]$. This shows that $g$ cannot be solved by algorithm X.

Now assume range $U_{g1}$ is empty and $U_{g2}$ is not empty. If the upper bound of $C_{g1}$ is greater than the lower bound of $U_{g2}$, the probability function $P_g$ will have

$$P_g(-c_1) = \lambda$$

and

$$P_g(-c_1+1)=\lambda,$$

where $c_1$ is the upper bound of the range $C_{g1}$. Therefore, we must have $c_1 \le l_2$, and this implies the conclusion presented in Lemma 5.3. □

If the range $U_g$ is understood as the ***uncertain part*** of the domain range $[-i, i]$ for the position $g$, we will say that the alpha-beta algorithm P_AB($g$, α, β) *visits* the certain part of position $g$ when the intersection of the two ranges $[\alpha, \beta]$ and $U_g$ consists of at most one integer, which is either α or β.

**Lemma 5.4.** If algorithm P_AB visits an interior solved node $g$, then there is an ordering of the nodes under $g$ such that P_AB will visit a successor $g_j$ of $g$ only if the algorithm X visits it, and when P_AB visits $g_j$, P_AB will visit its certain part.

*Proof.*

Let P_AB visit $g_t$ first, as determined by Lemma 5.3. Note that $g_t$ is such a solved node that the upper bound of $C_{g_t}$ is less than or equal to the lower bound of $U_{g_j}$ for all $j \ne t$. Then the reset of α and β will make P_AB either visit the certain part of a successor, $g_j$ ($j \ne t$), or return from $g$. □

**Lemma 5.5.** If algorithm P_AB visits the certain part of a interior node $g$, there is an ordering of the nodes under $g$ such that P_AB will visit a terminal node under $g$ only if that terminal node is visited by algorithm X.

*Proof.*

This lemma is proved by induction on the height of $g$ in the given search tree. In the following, it is assumed that $g$ is visited with a window $[\alpha, \beta]$. By Lemma 5.4, we can suppose position $g$ is semi-solved and, for simplicity, it has only two successors $g_1$

and $g_2$. If both of the ranges $U_{g1}$ and $U_{g2}$ are subsets of [-*upper_bound*$(U_g)$, -*lower_bound*$(U_g)$], when algorithm P_AB visits $g_1$ or $g_2$, P_AB will also visit the certain part, and the lemma is true by the induction hypothesis. If one of the ranges, for example, $U_{g1}$, is not contained in [-*upper_bound*$(U_g)$, -*lower_bound*$(U_g)$], then, because

$$P\_NEG(P_g) = P\_MIN(P_{g1}, P_{g2}),$$

we must have

$$-lower\_bound(U_g) < upper\_bound(U_{g1}).$$

This fact implies

$$-\beta \geq upper\_bound(U_{g2})$$

and

$$P_{g2}(v) = 0$$

for any $v \in$ [*upper_bound*$(U_{g2})+1$, $i$], or $-\alpha \leq \min($*lower_bound*$(U_{g1})$, *lower_bound*$(U_{g2}))$. In the former case, if P_AB visits $g_2$ first, then P_AB will visit the certain part of node $g_2$ with window $[-\beta, -\alpha]$, and the function P_AB($g_2$, $-\beta$, $-\alpha$) returns $<-\beta, 1>$. Therefore, P_AB will prune $g_1$. In the latter case, the visiting window $[-\beta, -\alpha]$ does not intersect with either of the two ranges, $U_{g1}$ or $U_{g2}$. This proves the lemma.
□

Lemma 5.5 implies that there is an ordering of successor nodes in the given search tree such that if P_AB visits the certain part of a node $g$ that is visited by algorithm X, then P_AB will visit only those nodes under $g$ that are also visited by algorithm X. We now turn to the proof of Theorem 5.2.

*Proof* of Theorem 5.2.

Because the root position $g_0$ of the given tree must be a solved node, P_AB($g_0$, $-i$,

$i$ ) visits the certain part of $g_0$. By Lemma 5.5, Theorem 5.2 follows. □

Since a search tree is finite, there must be an algorithm, say A, that visits the least number of terminal nodes among all the algorithms that solve the search tree. Theorem 5.2 implies that by ordering successors in the search tree, P_AB will visit only those terminal nodes that are visited by the algorithm A. Therefore, by reordering successor nodes in a search tree, the algorithm P_AB will visit the least number of terminal nodes among all the algorithms that solve the tree.

## 5. Applications

Some variations or applications of the probability-based alpha-beta algorithm P_AB are now presented. It is expected that the algorithm P_AB will have as many applications as its point-value version, as studied by Knuth and Moore[4]. For example, P_AB can be used in the iterative deepening search version for point-value game tree search[19], and also to other well-studied alpha-beta pruning based algorithms[17]. Only three examples of the applications of P_AB will be briefly described here.

### 5.1. Range-Based Game Tree Search

In range-based game tree search, the merit value $\Phi(g)$ of a position $g$ is described by a range $[l, u]$, where $l$ and $u$ are the lower and upper bounds of all possible merit values of $g$ [10]. If $g$ is a terminal node in the search tree, the bounds are returned by an evaluation function; otherwise, they will be backed-up from its successors $g_1, ..., g_n$ by

$$u = \max(-l_1, ..., -l_n) \text{ and } l = \max(-u_1, ..., -u_n),$$

where $g_1, ..., g_n$ are described by the ranges $[l_1, u_1], ..., [l_n, u_n]$, respectively.

The probability-based alpha-beta algorithm can be directly used as a range-based one if we assume that the merit value $\Phi(g)$ for any terminal node $g$ is uniformly distributed among the values of the range $[l, u]$, and then ignore the probabilities of the values between the lower and upper bounds of the backed-up probability function for the root position. In this way, the range for the possible merit values of an initial position can be backed up by P_AB. Since the formulas for the operations of range-based game tree search are much simpler than those for probability operations, the following range functions $R\_NEG$, $R\_MAX$ and $R\_MIN$ can be substituted for $P\_NEG$, $P\_MAX$ and $P\_MIN$ in the function $P\_AB$

$$R\_NEG([l, u]) = [-u, -l],$$

$$R\_MAX([l_1, u_1], \ldots, [l_n, u_n]) = [\max(l_1, \ldots, l_n), \max(u_1, \ldots, u_n)]$$

$$R\_MIN([l_1, u_1], \ldots, [l_n, u_n]) = [\min(l_1, \ldots, l_n), \min(u_1, \ldots, u_n)]$$

to get the so called range-based alpha-beta pruning algorithm R_AB. Note that this range-based algorithm makes use of depth-first traversing to search a tree, and so is different from B*[10].

```
function R_AB($g$ : node; $\alpha$, $\beta$: integer): range;
var
  TR0, TR1: range;
  j: integer;
begin
1.  if $g$ is a terminal node then
        return $R\_MIN(R\_MAX(R_g, [\alpha, \alpha]), [\beta, \beta])$;
    comment: Range $R_g$ is obtained by evaluating $g$
                 from the corresponding player's viewpoint.
2.  determine the successor positions, $g_1, \ldots, g_n$, of $g$, where $n > 0$;
```

```
3. TR0 := [$\alpha$, $\alpha$];
4. for $j$ := 1 to $n$ do
    begin
5.     TR1 := $R\_NEG$(R_AB($g_j$, -$\beta$, -$\alpha$));
6.     TR0 := $R\_MAX$(TR0, TR1);
7.     $\alpha$ := $lower\_bound$(TR0)
8.     if $\alpha = \beta$
          then return TR0;
    end;
9. return TR0;
end.
```

## 5.2. Informed Game Tree Search

Ibaraki[20] recently proposed an "informed" game tree search model based on the availability of some heuristic information, embodied as upper bound, $U\_bound(S)$, and lower bound, $L\_bound(g)$, on the merit value of an interior node $g$. The utilization of this kind of information has been recognized as a key factor for designing good game-playing programs[20][17]. The heuristic information available at interior nodes can be used in probability-based alpha-beta pruning by replacing the statement 3 of P_AB with the following series of statements:

```
3.1    $\alpha$ := max($\alpha$, $L\_bound(S)$);
3.2    $\beta$ := min($\beta$, $U\_bound(S)$);
3.3    if $\alpha = \beta$ then return <$\beta$, 1>;
3.4    TP0 := <$\alpha$, 1>;
```

Because the new window after the execution of statement 3.4 is usually a proper subset of the parameter window, this narrower window can be used to prune more nodes.

## 5.3. Probability-Based B*—PB*

The generalization of B* algorithm, PB*[11], makes use of a best-first search strategy. Since relatively reliable bounds for a terminal position can be generated with some such technique as the null-move[21], we can suppose that the expansion of a terminal node in the best-first search, and back-up of information from its successors will not increase the upper bound or decrease the lower bound of the possible merit values of the expanded node. This assumption is similar to the one employed by Ibaraki[20].

It can be proved that PB* works in the model proposed in Section 2 as well. As a matter of fact, the probability-based alpha-beta pruning technique can be incorporated into PB* to cut off some nodes from the search. In PB*, the following three operations are repeated until a best-move is found: find a potential best node, find a path from the root position of search tree to a terminal, and expand the terminal node. Suppose a node $g$ is called with a search window $[\alpha, \beta]$. When we choose one node $g_j$ from the successors $g_1, ..., g_n$ of a node $g$ according to the ProveBest or DisproveRest strategy (cf. [11]), the probability function

$$P = P\_MIN(P\_MAX(P\_NEG(P\_MIN(P_{g_1}, ..., P_{g_w})), <\alpha, 1>), <\beta, 1>)$$

can be used to set a new $\alpha$ by

$$\alpha := lower\_bound(P).$$

If $\alpha = \beta$, then the node $g_j$ should not be searched and a new strategy should be chosen; otherwise, the node $g_j$ will be visited with search window $[-\beta, -\alpha]$. When a node $g$ is expanded, each successor $g_j$ is examined as above, and if $\alpha = \beta$, the successors will not be included in the search tree. In this way, both the search time and the memory required to store the search tree will be reduced.

## 6. Pruning Efficiency

Random trees can be used to assess the pruning efficiency of algorithm P_AB. Given integers $d$ and $n$, a random tree can be generated so that each interior node has less than $n$ successor nodes and the tree consists of at most $d$ levels. The lower and upper bounds of the probability function for the merit value of each node is also generated randomly. An assumption is that the root position of each such random tree has a preselected probability function for its merit value. Therefore, using a variation on an earlier scheme [17], the probability function (or its bounds) for the root position is generated first, followed by the probability functions for the successor positions. To generate consistent probability functions for the successor nodes, if a node $g$ has lower and upper bounds $l$ and $u$ respectively, one of the successors is randomly chosen and its upper bound is set to the minimum upper bound $-l$. The minimum lower bound $-u$ is similarly assigned to another successor. The lower bounds of other successor nodes will be determined by randomly choosing integers between $-u$ and $i$, and the upper bounds by choosing integers between $-l$ and $i$, where $i$ is the domain bound. Since the lower bound and upper bound of a successor are chosen randomly and independently, the former may be greater than the latter; if this is so, the two bounds are exchanged.

For the efficiency experiment on the probability-based alpha-beta pruning scheme, ten random trees were generated for each combination of $d$ and $n$, where $d = 5, ..., 8$ and $n = 4, ..., 7$. Here, the domain value $i$ was set to 6, i.e., the domain of the probability functions was the integer range [-6, 6]. With these settings, Table 5.1 presents the total number of terminal nodes for each set of ten trees. In the recursive function P_AB, probability functions are passed as parameters, but only their lower and upper bounds are used in the pruning. In this case, since only the bounds of the probability functions are used, P_AB reduces to the range-based pruning algorithm, R_AB. For our test data, the number of terminal nodes visited by P_AB (or R_AB) for each combination of $d$ and $n$

is shown in Table 5.2. Table 5.3 presents the efficiency of the pruning and also shows that the relative efficiency, as measured by the fraction of terminal nodes that are pruned by P_AB, increases as either the width or depth of search tree increases. Informally speaking, the larger the search tree, the greater the fraction of nodes that will be pruned by probability-based alpha-beta.

Table 5.1. The Number of Terminal Nodes in Ten (*d*, *n*)-Random-Trees

| Depth (*d*) | Width Limit (*n*) | | | |
|---|---|---|---|---|
| | 4 | 5 | 6 | 7 |
| 5 | 323 | 919 | 1943 | 2022 |
| 6 | 623 | 2865 | 5621 | 9827 |
| 7 | 2149 | 9529 | 21820 | 30402 |
| 8 | 3537 | 26371 | 70289 | 220723 |

Table 5.2. The Number of Terminal Nodes Visited by P_AB

| Depth (*d*) | Width Limit (*n*) | | | |
|---|---|---|---|---|
| | 4 | 5 | 6 | 7 |
| 5 | 250 | 692 | 1100 | 1044 |
| 6 | 538 | 1481 | 3074 | 3827 |
| 7 | 1371 | 5110 | 10076 | 11528 |
| 8 | 2593 | 14268 | 29867 | 72564 |

Table 5.3. The Efficiency of P_AB for Different (*d*, *n*)

| Depth (*d*) | Width Limit (*n*) | | | |
|---|---|---|---|---|
| | 4 | 5 | 6 | 7 |
| 5 | .22 | .24 | .43 | .48 |
| 6 | .13 | .48 | .45 | .61 |
| 7 | .36 | .46 | .53 | .62 |
| 8 | .26 | .45 | .57 | .67 |

Chapter 6

# Conclusions

## 1. Contributions

This thesis studies computer game playing. It provides some new results for the topics of game tree modeling, minimax pathology investigation, pruning efficiency analysis, and position evaluation representation. It focuses on the application of probabilities. These results are summarized in the following paragraphs.

In Chapter 2, a new probabilistic model, called node-dependent minimax game tree, for game trees is proposed. In this model, the node-value dependence is described with conditional probabilities. Instead of randomly assigning merit values to leaf nodes, which has been commonly done, the root node of a uniform tree is assigned a merit value randomly first, then the values of leaf nodes are randomly determined by the conditional probabilities in a top-down manner. In a node-dependent minimax game tree, the values of leaf nodes are no longer independent of each other, although they can be identically distributed random variables. The node-dependent minimax game tree has some interesting properties, which can be utilized to simulate real games. These properties are not present in an independent random uniform game tree. They are exemplified with bi-valued binary dependent minimax game trees. An interesting property is that for any real number $p_0$ with $0 < p_0 < 1$, pairs of values of dependent factors $f_1$ and $f_2$, which are defined as conditional probabilities, can be found so that both the root and the leaf nodes have the same probability $p_0$ to receive value 1 (or 0). By varying the dependent factors in a game tree, the first player Max may have more chances to win the game but there are fewer win leaf nodes in the tree. The flexibility of node-dependent minimax game trees reflects the great variety of real games.

Chapter 3 investigates the curious phenomenon of minimax pathology, which appears in theoretical study but seldomly in practice or can be avoided, is investigated. A class of two-player zero-sum perfect-information board-splitting games, which are related to dependent minimax game trees, are used to obtain some new observations. First, minimax search error is modeled with bi-valued evaluation functions. It is shown that for some games and evaluation functions, minimax back-up reduces the error propagation. In other words, for some combinations of the measures of node-value dependence and evaluation errors, the probability for making a wrong decision at the root node is reduced by a deeper search. This gives a positive proof that minimax back-up can benefit game tree search. Mathematical formulas are developed for computing the probabilities of making a correct decision in node-dependent trees when searching to different depths with a piece-counting evaluation function. The calculation provides strong evidence of a relationship between minimax pathology and node-value dependence. The analysis results imply that the strong correlation of the sibling node values in real games is a good explanation for the success of minimax search.

Chapter 4 presents a new method for analyzing the efficiency of the most commonly used game tree search algorithm, the alpha-beta pruning algorithm. This method works on the node-dependent minimax game trees and follows a top-down recursive pattern. The effect of a tight window on the efficiency of alpha-beta pruning is also discussed. Under the assumption of alpha-beta pruning with tight window, recursive equations for the average number of terminal nodes visited in a bi-valued binary node-dependent game tree are derived. The analysis of the effect of node-value dependence on the pruning efficiency is based on their solutions.

In Chapter 5, the efficient search scheme, depth-first traversal, is introduced into probability-based game tree search. It is shown that when the utility of a node is described by probabilities, $\alpha$-$\beta$ bounded windows can be used to cut off some subtrees from search. It is also shown that probability-based alpha-beta pruning can be viewed as a generalization of the standard alpha-beta game tree search algorithm and that it inherits some good properties from its point-value version. Several variations of probability-based alpha-beta game tree pruning are presented, one of which is the "degeneration" of this probability-based algorithm into a range-based one. It is indicated that this probability-based game tree pruning technique would have as many applications as its point-value version. Probability experiments are used to show that i· can be exploited to effectively prune the search of some subtrees.

## 2. Future Work

In this thesis, the new game tree model, the node-dependent minimax game tree, is essentially a mathematical model. No attempts have been made to relate its parameters to real games like chess, checkers and GO. An open problem is how to quantify the conditional probabilities of the node-dependent minimax game trees for these real games. In this way, tne relationship between this new model and real games can be revealed.

The node-dependent minimax game trees have been discussed for analyzing the alpha-beta algorithm and investigating minimax pathology. There are many other game tree search algorithms. How to analyze other game tree search algorithms with the dependent game trees is a problem to be solved.

Although the node-dependent game tree has a good property that influence of the probability for the root node to take a merit value on the probability for a terminal node to take the merit value is decreased along with the increase of the height of the tree, we still can introduce more randomness into this model. One of the possible improvements introduces some randomness into each level. In this way, the effects of upper level nodes could be reduced. This randomness introduction is another new problem. For another possible improvement, we can delay the merit value determining for the nodes at several upper-levels, and randomly choose merit values for the nodes at an interior level.

As shown in the last chapter, the probability-based alpha-beta pruning algorithm exploits depth-first search in a game tree. A natural problem is how to make use of the probabilities provided by evaluation function to conduct a more "intelligent" search. In other words, the search front should be determined by the probabilities of the current leaf nodes in the search tree rather than by a predefined depth limit. If the methods of the probability-based B*, presented by Palay [11] and the probability-based alpha-beta pruning algorithm, presented here, could be combined, both the search efficiency and search quality could be increased.

## References

1. J. Pearl, *Heuristics: Intelligent Search Strategies for Computer Problem Solving*, Addison-Wesley Publishing Company, Reading, Massachusetts, 1984.

2. J. Pearl, Asymptotic Properties of Minimax Trees and Game-Searching Procedures, *Artificial Intelligence 14*, (1980), pp. 113-138.

3. D. S. Nau, The Last Player Theorem, *Artificial Intelligence 18*, (1982), pp. 53-65.

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

5. S. H. Fuller, J. G. Gaschnig and J. I. Gillogly, *Analysis of the Alpha-Beta Pruning Algorithm*, Department of Computer Science Report, Carnegie-Mellon University, July 1973.

6. D. S. Nau, An Investigation of the Causes of Pathology in Games, *Artificial Intelligence 19*, (1982), pp. 257-278.

7. J. Pearl, On the Nature of Pathology [illegible] me Searching, *Artificial Intelligence 20*, (1983), pp. 427-453.

8. D. F. Beal, Benefits of Minimax Search, *Advances in Computer Chess 3*, (1982), pp. 17-24.

9. M. M. Newborn, The efficiency of the Alpha-Beta Search on Trees with Branch-dependent Terminal Nodes Scores, *Artificial Intelligence 8*, (1977), pp. 137-153.

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

11. A. J. Palay, *Searching with Probabilities*, Pitman Advanced Publishing Program, Boston, 1985. Also, Ph. D. Thesis, Department of Computer Science Report, Carnegie-Mellon University, 1983.

12. G. M. Baudet, On the Branching Factor of the Alpha-Beta Pruning Algorithm, *Artificial Intelligence 10*, (1978), pp. 173-199.

13. M. M. Newborn, Computer Chess: Recent Progress and Future Expectation, in *Information Technology*, J. Moneta (ed.), 1978, pp. 189-192.

14. K. Thompson, Computer Chess Strength, in *Advances in Computer Chess 3*, M. R. B. Clarke (ed.), 1982, pp. 55-56.

15. D. F. Beal, An Analysis of Minimax, *Advances in Computer Chess 2*, (1980), pp. 103-109.

16. A. L. Brudno, Bounds and Valuations for Abridging the Search of Estimates, *Problems of Cybernetics 10*, (1963), pp. 225-241. Translation of Russian original appearing in Problemy Kibernetiki 10, pp. 141-150.

17. T. A. Marsland, A. Reinefeld and J. Schaeffer, Low Overhead Altւ rnatives to SSS*, *Artificial Intelligence 31*, (1987), pp. 185-199.

18. J. L. Mott, A. Kandel and T. P. Baker, *Discrete Mathematics for Computer Scientist*, Reston Publishing Company, Inc., Reston, Virginia, 1983.

19. D. J. Slate and L. R. Atkin, Chess 4.5-The Northwestern University Chess Program, in *Chess Skill in Man and Machine*, P. Frey (ed.), Springer-Verlag, 1977, pp. 82-118.

20. T. Ibaraki, Generalization of Alpha-Beta and SSS* Search Procedures, *Artificial Intelligence 29*, (1986), pp. 73-117.

21. D. Beal, A Generalized Quiescence Search Algorithm, *Artificial Intelligence*, (to appear) , 1988. Also, Experiments with the null move, *Advances in Computer Chess 5*, Elsevier, pp. 65-79.