To the Graduate Council:

I am submitting herewith a thesis written by Michael Christopher van Lent entitled "A Pruning Algorithm for One Player Games with Hidden Information." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.

David C Mutchler

Dr. David Mutchler, Major Professor

We have read this thesis
and recommend its acceptance:

Jean R. S. Blair

Michael D Vose

Accepted for the Council:

Lew Minkel

Associate Vice Chancellor
and Dean of The Graduate School

A PRUNING ALGORITHM FOR ONE PLAYER GAMES WITH HIDDEN INFORMATION

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Michael Christopher van Lent

May 1993

Acknowledgments

I would like to thank Dr. David Mutchler for his always enthusiastic assistance in the research and writing of this thesis. Thanks also to Dr. Blair and Dr. Vose for their interesting questions and helpful comments and to all my friends in the CS department for their support.

Abstract

This thesis makes three contributions to the development of search algorithms for one-player games with hidden information. The first contribution is the description and analysis of a new pruning technique, called information set pruning, that is specific to hidden information games. Second, the One-Player Pruning Algorithm (OPPA) is presented; it is a natural and important extension of the existing One-Player Algorithm to search game trees with hidden information. OPPA uses information set pruning and other pruning techniques to provide gains in search time by exploring fewer leaves while costing nothing in solution quality and very little in implementation complexity. Finally a general model of game trees with hidden information is suggested. This model has many advantages in terms of generality and clarification of the hidden information concept. It is hoped that further work on hidden information games will utilize this model as a standard form of hidden information game trees. These three contributions take the next logical step in the development of search algorithms for hidden information games and suggest many areas for future work.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

A basic problem solving technique in artificial intelligence is to represent the problem as a graph and then find a solution by searching that graph. This technique works particularly well for games as they are easily represented by trees and heuristic search of these trees frequently results in very competitive strategies. Many search techniques specifically designed for games, such as minimax[2] and alpha-beta[5], require that the game have no hidden information. A recently invented search algorithm that successfully handles hidden information is the One-Player Algorithm(OPA)[3]. In this thesis we will present a second search algorithm for game trees with hidden information which utilizes pruning to explore less of the tree than OPA while returning the same expected payoff.

Hidden information games are games in which part of the information making up the current state of the game is hidden. Board games, such as chess, are generally not hidden information games because the board, which fully describes

the state of the game, is visible to both players. Card games commonly are hidden information games because the cards in the opponent's hand, which are part of the game state, are hidden.

Like most card games blackjack involves hidden information. Blackjack is easily represented as a graph by letting each node represent a state of the game and each edge represent a decision to either hit or stay. Each game state consists of the cards held by the player and the cards held by the house. A hit move changes the game state by adding one card to the player's hand. A stay move leads to a terminal node or leaf where the payoff is decided. In blackjack part of the hidden information is the house's down card which the player is not allowed to see. Since the player doesn't know what the house's down card is he or she cannot differentiate 10 different game states(see figure 1.1). Therefore when defining a strategy for blackjack the same move must be made from these 10 game states.

In the tree representation of a game, hidden information manifests itself as a set of nodes which cannot be differentiated. A move or decision must be made without knowing which of these nodes is actually the current game state. Since a player cannot differentiate these nodes it is impossible for the player to choose a strategy which makes different moves from nodes that are not differentiable. This restriction is the major difference between games with and games without hidden information.

Although the concept of hidden information is generally discussed in relation to games it applies equally well in many other situations. The work in this thesis will be phrased in terms of searching game trees because game theory provides the terminology required to discuss hidden information, optimal strategies, and a variety of other concepts. The algorithm presented should not be viewed as

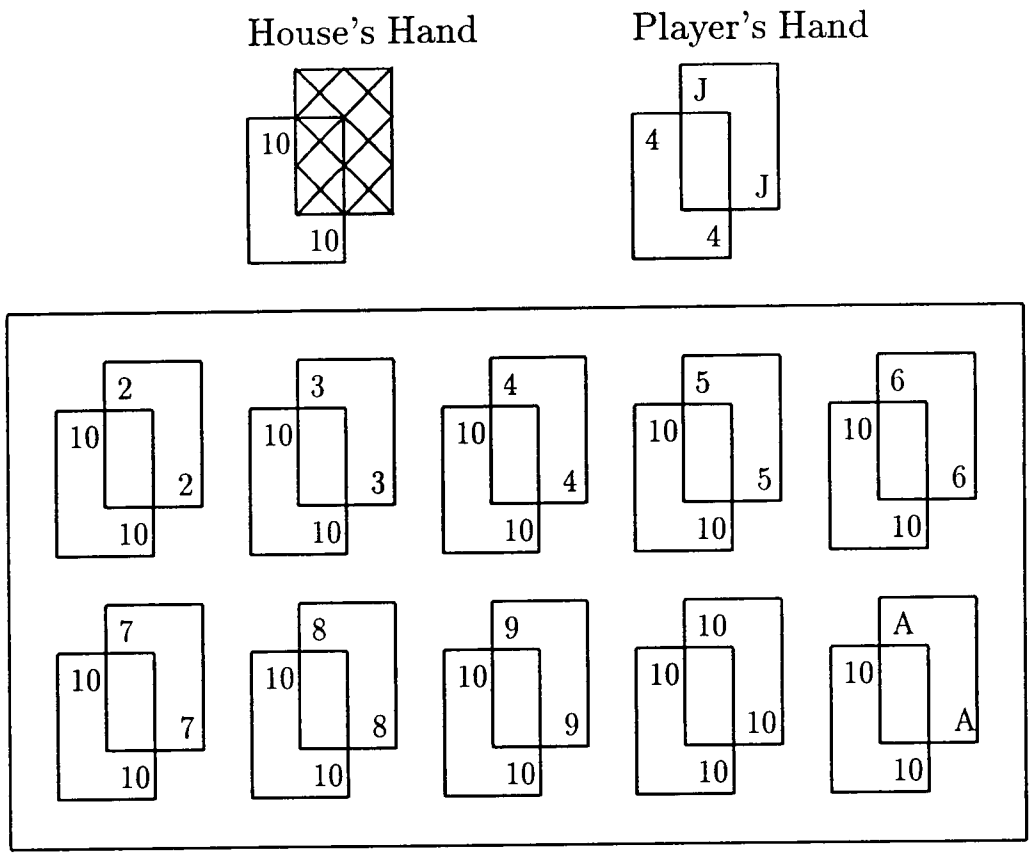

Figure 1.1: *A typical game state in blackjack. Above is the game state from the player's point of view with the house's down card hidden. Below are the house's hand for the 10 games states that cannot currently be differentiated. Note that the face cards are all worth 10 and therefore represent the same state.*

useful solely for game trees but as a general search technique for problems that involve hidden information.

Perhaps the best known algorithm for searching game trees is the minimax[2] algorithm which explores the entire tree to find an optimal strategy. A variation of minimax called alpha-beta[5] utilizes the concept of pruning to find an optimal solution usually without exploring the entire tree. An expansion of alpha-beta called *minimax[1] searches game trees with chance nodes and uses these chance nodes for even more pruning.

These three game tree searching algorithms have the common weakness that if the game tree being searched has hidden information then the strategy found is no longer guaranteed to be optimal. Recent work on this problem has resulted in the One-Player Algorithm[3] which, like minimax, explores the entire game tree and, when there is perfect recall, finds an optimal strategy even if there is hidden information involved. The perfect recall property of games will be discussed in the next chapter.

We will present the One-Player Pruning Algorithm (OPPA), an extension of the One-Player Algorithm, which includes some of the pruning techniques found in alpha-beta and *minimax as well a new type of pruning specific to games with hidden information. We will prove that on any game tree OPPA will find the same strategy as OPA. A general model of hidden information games will be developed and used to analyze the effectiveness of the new pruning technique in both the best and average case.

This thesis makes three contributions to the development of search algorithms for one-player games with hidden information. The first contribution is the description and analysis of a new pruning technique, called information set

pruning, that is specific to hidden information games. Second, the One-Player Pruning Algorithm is presented which is a natural and important extension of the One-Player Algorithm to search game trees with hidden information. OPPA uses information set pruning and other pruning techniques to provide gains in search time by exploring fewer leaves while costing nothing in solution quality and very little in implementation complexity. Finally a general model of game trees with hidden information is suggested. This model has many advantages in terms of generality and clarification of the hidden information concept. It is hoped that further work on hidden information games will utilize this model as a standard form of hidden information game trees. These three contributions take the next logical step in the development of search algorithms for hidden information games and suggest many areas for future work.

The next chapter will describe the One-Player Algorithm and in the process formally define many of the concepts discussed informally in this introduction. Chapter 3 will present the One-Player Pruning Algorithm and prove that OPPA and OPA find the same strategy. Chapter 4 will discuss games with hidden information and develop a general model for these games. Chapter 5 will use the model in a theoretical analysis of the effectiveness of hidden information pruning in the best case. Chapter 6 will utilize the model again to make an empirical analysis of the average case effectiveness of hidden information pruning. Finally, Chapter 7 will include ideas for future work.

Chapter 2

Definitions

The goal of this chapter is to define the terms used throughout this thesis and introduce the One Player Algorithm. This chapter is not meant to serve as an introduction to game theory or the development of game playing algorithms. An excellent introduction to game theory is provided in [7] and game playing algorithms are discussed in most artificial intelligence textbooks (for example [2, 8]).

The definitions in this chapter are divided into three sections. The first section formally defines a one player game and the hidden information and perfect recall properties. The second section describes a strategy in terms of the definition of a game. In the last section the One Player Algorithm is presented along with some of its properties.

2.1 One Player Game

Game theorists describe games in two ways called the extensive form and the normal form. When writing or discussing an algorithm it is usually easier to view a game in the extensive or tree form.

> *By a tree, we mean a rooted tree with an associated parent function. With regard to trees, we use without explanation terms like node, edge, path, depth, root, leaf, interior node, parent, child, ancestor, and descendant. (A node is both an ancestor and descendant of itself.) See any standard text on data structures for definitions of these terms.* [1]

As mentioned in the introduction a game can be represented as a tree if the states of the game are viewed as nodes and the moves as edges. Thus for any intermediate game state, or node, there are a number of possible alternatives (moves), or edges, which lead to new games states, children of the original node. When the game is over the payoff to a player is decided by a payoff function which takes the final game state, or leaf, and returns a score.

Many games involves some form of random possibilities such as rolling a die or shuffling cards. To account for this certain nodes of the game tree, called chance nodes, can be used to represent the random possibilities. A chance node has a child for each possible outcome of the random event and the edge to each child is weighted with the probability of that child's outcome occurring.

[1] As the work presented in this thesis extends the work of Blair, Mutchler, and Liu[3] most of the definitions will simply be quoted from that source.

In some games part of the information about the current game state is hidden from the player. When a player does not have all the information about the current game state that player will not be able to differentiate some of the nodes in the game tree. The sets of nodes that cannot be differentiated are called information sets. The effect of information sets will become more clear during the discussion of strategies in the next section.

These different elements of a one player game are combined into the following definition.

A <u>*one-player game*</u> Γ *consists of:*

- *A finite tree* $\mathcal{K}$ *called the* <u>*game tree*</u>*. The edges below any interior node* x *in* $\mathcal{K}$ *are the* <u>*alternatives*</u> *from* x*.*
- *A partition of the interior nodes in* $\mathcal{K}$ *into two classes: the* <u>*chance nodes*</u> *and the* <u>*player nodes*</u>*.*
- *For each chance node* x *in* $\mathcal{K}$*, a probability distribution on the alternatives from* x*.*
- *A partition of the player nodes in* $\mathcal{K}$ *into* p <u>*information sets*</u>*,* $I_1, \cdots, I_p$*. This partition must satisfy the following, for any nodes* x *and* y *in the same information set:*
  - *The number of children below* x *equals the number of children below* y*.*
  - *If* $x \neq y$*, then neither node is an ancestor of the other.*
- *A real-valued function* h*, called the* <u>*payoff function*</u>*, on the leaves of* $\mathcal{K}$*.*

Throughout this thesis Γ will be used to refer to both one player games and one player game trees including the chance node partition, information sets, and payoff function.

If a game has any nodes that cannot be differentiated then there must be some information about the game state that is being hidden from the player. Games for which this is true are said to have the hidden or imperfect information property.

> *When there exists an information set with more than one node in it, the game is said to exhibit imperfect information.*

Another property that a game may or may not have is called perfect recall. In a game with perfect recall the player can recall which alternatives have been taken in the game to reach the current node. The words "alternatives taken" refer to both the choices made by the player and the outcomes of the random events at chance nodes.

> *For information sets I_j and I_k in game Γ, we say that I_k follows I_j, written $I_k > I_j$, if $I_k \neq I_j$ and there exists nodes $x \in I_j$ and $y \in I_k$ such that y is a descendant of x. For any subset X of an information set I_j, the ith child of X is defined to be* $\{y \mid y \text{ is the } i\text{th child of a node in } X\}$. *A one-player game Γ has perfect recall if for every pair of information sets I_j and I_k in Γ such that $I_k > I_j$, there exists an i such that I_k lies entirely inside the forest of subtrees rooted at the ith child of I_j.*

2.2 Strategy

A strategy for a game is simply a specification of which alternative to chose for every possible player node in the game tree. Since nodes in an information set cannot be differentiated a strategy cannot specify different moves for different nodes in the same information set. Thus a strategy is a choice of an alternative from every node such that all nodes in the same information set chose the same alternative.

> *A strategy π on $\mathcal{K}$ is a function on the player nodes in $\mathcal{K}$ such that for any player nodes x and y in $\mathcal{K}$:*
>
> - *$\pi(x)$ is a child of x.*
> - *If x and y are in the same information set, $\pi(x)$ and $\pi(y)$ are the same alternative (i.e., if $\pi(x)$ is (say) the jth child of x, then $\pi(y)$ is the jth child of y).*

To define what it means for a strategy to be the optimal strategy it is first necessary to define the probability of an edge, the probability of a node, and expected payoff.

> *Given strategy π on game tree $\mathcal{K}$, the probability of edge $e = (x, y)$ under π, denoted $p_\pi(e)$, is defined as follows.*
>
> - *If parent node x is a chance node, then $p_\pi(e)$ is the probability of the alternative (x, y), under the probability distribution associated with x.*

- *If parent node x is a player node, then $p_\pi(e)$ is 1 if $\pi(x) = y$, else $p_\pi(e)$ is 0.*

For any node x in $\mathcal{K}$, the probability of node x under π, denoted $p_\pi(x)$, is defined to be $\prod p_\pi(e)$, where the product is over the edges e in the path from the root of $\mathcal{K}$ to x. (The probability of the root is defined to be 1.) The expected payoff under strategy π, denoted $H(\pi)$, is defined to be $\sum p_\pi(w)\, h(w)$, where the sum is over all leaves w in $\mathcal{K}$.

Given these definitions an optimal strategy for a game is defined as the strategy that gives the best expected payoff.

A strategy π is an optimal strategy for game tree $\mathcal{K}$ if it maximizes (over all strategies for $\mathcal{K}$) the expected payoff. A solution to a one-player game Γ is an optimal strategy for $\mathcal{K}$.

Finding the optimal strategy for a one player game can be proven to be NP-complete.

ONE-PLAYER GAME (OPG):

Instance: *A one-player game Γ and a real number K, where Γ consists of a finite tree $\mathcal{K}$; a partition of the interior nodes of $\mathcal{K}$ into a set of chance nodes C and a set of player nodes P; for each node $c_i \in C$ a probability distribution on the children of c_i; a partition of the nodes in P into information sets $I_1, I_2, \ldots, I_p$; and a real-valued function h on the leaves of $\mathcal{K}$.*

Question: *Is there a strategy π for $\mathcal{K}$ with expected payoff $H(\pi) \geq K$?*

Theorem 1 *OPG is NP-complete.*

The reader is referred to Blair, Mutchler, and Liu[3] for the proofs of the theorems presented in this chapter. The algorithm presented in the next section finds optimal strategies of the subclass of one player games that have perfect recall.

2.3 The One Player Algorithm

Before the One Player Algorithm can be described a couple of subsidiary operators need to be defined. The first of these is the S-operator which OPA uses to handle chance nodes.

For any set X of nodes in a game tree $\mathcal{K}$, the S-operator applied to X, denoted $S(X)$, returns the set obtained by replacing each chance node in X by its children, and repeating this action to the resulting set until the resulting set contains no chance nodes.

The effects of the chance nodes are not ignored, just moved to the leaves. This is done by calculating the probability that any given leaf will be reached independent of the current strategy.

For leaf x the reachable probability of x, denoted $p(x)$, is defined as follows. Let ρ be the "optimistic" pseudo-strategy in which every

alternative of every player node is "chosen" (i.e., $p_\rho(e) = 1$ for every edge e having a non-chance node parent). Then for leaf x, we have $p(x) = \prod p_\rho(e)$ where the product is over the edges e in the path from the root of $\mathcal{K}$ to x. Note the difference between $p(x)$ and $p_\pi(x)$: $p(x)$ is independent of any strategy, where as $p_\pi(x)$ is dependent on some particular strategy π.

For this thesis we will extend the definition of reachable probability to include the player nodes. The definition of the reachable probability of a player node is exactly the same as for a leaf. Then for player node x, we have $p(x) = \prod p_\rho(e)$ where the product is over the edges e in the path from the root of $\mathcal{K}$ to x.

In exploring a game tree without perfect recall the One Player Algorithm may find itself looking at a set of nodes that are in the same information set but do not make up the entire information set. Thus a partial information set is any subset of an information set.

Given these definitions the One Player Algorithm can be concisely described as a recursive function which computes the value of a set X of nodes.

The algorithm is encapsulated by the following recursive function to compute the value of a set X where the nodes in X all belong to a single level in the game tree. Recall that when X is a partial information set, it makes sense to speak of the children of X, which

are themselves sets of nodes.

$$V(X) = \begin{cases} \max\{V(S(Y)) \mid Y \text{ is a child of } X\} & \text{if } X \text{ is a partial information set} \\ \sum_{\substack{x \in X \\ x \text{ a leaf}}} p(x)\, h(x) + \sum_{Y \in \textit{info-sets}(X)} V(Y) & \text{otherwise} \end{cases} \tag{2.1}$$

The One Player Algorithm computes the value of a set of nodes but it can also be used to compute a strategy for a set of nodes.

The strategy computed by V, denoted $\underline{\pi^*}$, is the strategy obtained by using the above definition as a recursive algorithm to compute $V(S(\{root\ of\ \mathcal{K}\}))$

The One Player Algorithm is a very powerful algorithm in that it explores game trees in time linear in the number of nodes in the tree and returns the value of the optimal strategy if the game has perfect recall.

Theorem 2 *In a game with perfect recall, π^* is an optimal strategy and the maximal expected payoff is $V(S\{root\ of\ \mathcal{K}\})$.*

Chapter 3

The One-Player Pruning Algorithm

The One-Player Pruning Algorithm finds optimal strategies for one player games with perfect recall. There may or may not be hidden information and there may or may not be chance nodes. The only additional restriction that OPPA makes is that there be an upper bound U and a lower bound L on the payoffs at the leaves. This restriction is also made by multi-player alpha-beta and *minimax and in practice is not particularly limiting.

While the requirement that there be only one player may seem very restrictive there are many situations where a single player is all that is called for. A large class of obvious examples are the various solitaire card games. Almost all versions of solitaire are one player games with hidden information and therefore are well suited for OPPA. Another class of examples are games where one player's strategy is decided ahead of time, such as casino blackjack. Games of

this sort, where one player has only one alternative from any game state, can be easily represented as one player games by removing the predetermined player from the game tree. An additional large class of one player games stems from the fact that there is little difference between searching the graphical representation of a problem to find a solution and exploring a one player game tree. Thus OPA and OPPA are not only game playing algorithms but are also depth first search techniques that belong to the same class as A* and gradient descent.

As mentioned before perfect recall is simply the requirement that the past moves at player nodes and outcomes at chance nodes are remembered. Again this restriction is not particularly limiting as many one player games and search situations have the perfect recall property. In games that do not have perfect recall OPA and therefore OPPA will not return optimal strategies but can be used as heuristics to estimate the value of a node[3].

The remainder of this chapter will consist of a discussion of the various pruning techniques that OPPA uses, a description of the One-Player Pruning Algorithm and the proof that OPPA and OPA calculate the same expected payoff.

3.1 Pruning

Taking the One-Player Algorithm as a starting point the goal of this chapter is to describe the extensions made in the development of a pruning algorithm which calculates the same optimal expected payoff without exploring the entire game tree. This is done by pruning or not exploring parts of the game tree that are guaranteed to have no effect on the expected payoff. The concept of pruning is a powerful aid in exploring game trees. Game tree pruning is a crucial part of

chess playing programs such as Deep Thought to limit the amount of search needed to find successful strategies.

Two of the most common pruning algorithms for games without hidden information are alpha-beta[5] and *minimax[1]. Alpha-beta uses two types of pruning; immediate pruning and max-min pruning[6]. OPPA will also use immediate pruning however max-min pruning will not be implemented as it requires two players and would be ineffective on our one-player games. As an extension to alpha beta, *minimax adds an additional pruning technique called chance node pruning, to the two previously mentioned. Since chance node pruning works at chance nodes in any sort of game, including one player games, it will be used in OPPA's pruning arsenal.

In much the same way that chance nodes provide a new opportunity for pruning the addition of hidden information gives rise to another pruning technique, information set pruning. OPPA will use information set pruning in conjunction with immediate and chance node pruning to limit the amount of the game tree explored. Our analysis of the effectiveness of OPPA's pruning will concern itself mainly with information set pruning as both immediate pruning and chance node pruning have been analyzed elsewhere[1, 6].

3.1.1 Immediate Pruning

Immediate pruning occurs when a move is found that gives the best possible payoff (a sure win). When this move is found there is no point in searching the remaining alternatives as they cannot give a better payoff.

Our algorithm will implement immediate pruning by checking each time a better

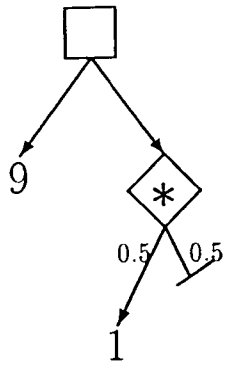

Figure 3.1: *A small game tree that exhibits chance node pruning. The square node is a player node and the diamond node is a chance node. The upper bound on payoffs at leaves is 10. The player already has a move which gives a payoff of 9 and the chance node can give at most 5.5 so the second child of the chance node is pruned.*

alternative is found to see if the value of that alternative is equal to the upper bound. If so then OPPA will prune the remaining alternatives and return the upper bound as the value of that game state.

3.1.2 Chance Node Pruning

Chance node pruning takes advantage of the fact that the value of a chance node is the weighted sum of its children. After partially exploring a chance node an upper bound on its possible value can be estimated by assuming that the unexplored children all take on the maximum possible payoff. If this value turns out to be less than the payoff of a previously explored strategy then the unexplored children under the chance node can be pruned (see figure 3.1).

OPPA, like OPA, doesn't explicitly calculate weighted sums at chance nodes but

uses the S-operator and reachable probability to move the calculation to the leaves. Because of this the chance node pruning in OPPA is also moved to the leaves by modifying the pruning bound that is passed down from chance nodes. While this has the advantage of making the algorithm simple and very similar to OPA it slightly obscures the operation of the chance node pruning.

3.1.3 Information Set Pruning

Information set pruning is very similar to chance node pruning in that it takes advantage of the calculation of a weighted sum. The One-Player Algorithm handles information sets by comparing the weighted sum of the various alternatives from the information set. As in chance node pruning when an alternative has been partially explored an upper bound on its final payoff can be estimated by assuming that the unexplored portion of the alternative takes on the maximal value. If this upper bound is lower that a previously explored alternative then the unexplored portion of the current alternative can be pruned (see figure 3.2).

The One-Player Pruning Algorithm combines chance node pruning and information set pruning into a single test. This makes the pruning condition somewhat difficult to follow but is more efficient than separate pruning conditions and allows us to retain similarity to the One-Player Algorithm. The proof of correctness will explain the somewhat complex pruning calculations.

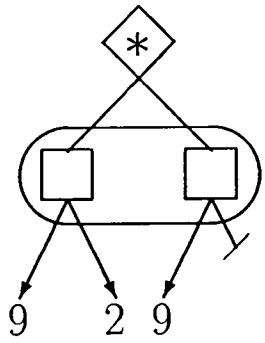

Figure 3.2: *A small game tree that exhibits information set pruning. The upper bound on payoffs at leaves is 10. The left alternative from the information set gives an average payoff of 9 while the right alternative can give at most 6 so the remaining portion of the right alternative is pruned.*

3.2 The Algorithm

The One-Player Pruning Algorithm consists of three main subroutines (`OPPA`, `Max-set`, and `Mixed-set`) and a variety of supporting functions. As the supporting functions are straightforward and problem dependent we will simply describe their function without providing pseudo-code. The three main subroutines will be presented in pseudo-code and explained in detail. Throughout the description of OPPA we will use X to denote a set of nodes and x to denote a single node.

3.2.1 Supporting Functions

The following functions are referred to by the One-Player Pruning Algorithm.

- `Partial-Information-Set(X)` takes a set of nodes X and returns true if they are all members of the same information set and false otherwise. Note that X does not need to contain all the nodes in the information set to satisfy `Partial-Information-Set`.

- `S(X)` takes a set of nodes and applies the S-operator as described in Section 2.3. Thus `S` takes a set of nodes and repeatedly replaces all the chance nodes with their children.

- `p(x)` takes a node and returns its reachable probability as described in Section 2.3.

- `Reachable-Prob(X)` takes a set of nodes X and returns the sum of their reachable probabilities.

$$\texttt{Reachable-Prob(X)} = \sum_{x \in X} p(x)$$

- `partition(X)` takes a set of nodes X and partitions them into leaves and partial information sets. `Partition` returns a set in which each member is either a leaf(atom) or a partial information set (set).

- `h(x)` takes a leaf node and returns the payoff at that leaf.

3.2.2 OPPA

As mentioned the One-Player Pruning Algorithm consists of three main subroutines. The top level function, `OPPA`, takes a set of nodes X and a pruning value P. If the sum of the expected payoffs of the forest of subtrees rooted in X is greater than P then `OPPA` returns that expected payoff. If the expected payoff

sum is less than P then P is returned. By calling OPPA on a game tree with the lower bound as the initial pruning value the expected payoff of the game tree is returned (see theorem 4).

```
OPPA(X,P)
    if Partial-Information-Set(X)
       Max-Set(X,P)
    else
       Mixed-Set(X,P)
```

The OPPA subroutine doesn't do any exploration itself. Instead it calls one of two specialized routines to do the actual exploration. The Max-set routine is called when the set of nodes to be explored is a single partial information set and Mixed-set is called in any other case.

The Max-set routine takes a partial information set X and a pruning value P. As with OPPA if the expected payoff of the partial information set is greater than P then that expected payoff is returned; otherwise P is returned. The only pruning that takes place in Max-set is the immediate pruning which is done by the pruning condition temp = U.

```
Max-Set(X,P)
       best = P
       for each Y a child of X
           temp = OPPA(S(Y),best)
           if temp > best
              if temp = (U * Reachable-Probability(X))
```

```
                return(U * Reachable-Probability(X))
            best = temp
        return(best)
```

Simply stated `Max-set` makes a recursive call to `OPPA` for each of the alternatives from X and keeps track of the best alternative. More specifically, for each alternative i from X `Max-set` creates a set Y consisting of the ith child of each node in X. `OPPA` is called recursively on each of these sets of nodes. If the expected payoff returned by OPPA is the upper bound then immediate pruning takes place. Otherwise the expected payoff of the best alternative (as computed by `OPPA`) is stored if it is greater than P. Finally the value returned by `Max-set` is either the expected payoff of the best alternative or P whichever is greater.

`Max-set` corresponds to the top line in the recursive statement of the One-Player Algorithm (see equation 2.1) with the only additions being immediate pruning and the P value which acts as a lower bound on the value returned.

The `Mixed-set` routine is the most complex of the three due to the combination of chance node pruning and information set pruning. `Mixed-set` takes a set of player nodes X and a pruning value P. The set of nodes passed to `Mixed-set` can be any collection of player nodes and leaves; it should not contain chance nodes. The value returned by `Mixed-set` is the sum of the weighted payoffs of the leaves and information sets in X or the pruning value P, whichever is greater.

```
Mixed-Set(X,P)
        sum = 0
        prob-left = Reachable-Prob(X)
```

```
Z = partition(X)
for each member Y of Z
    if Y is an atom
       prob-left = prob-left - p(Y)
       sum = sum + h(Y) * p(Y)
    else
       prob-left = prob-left - Reachable-Prob(Y)
       sum = sum + OPPA(Y,Max(P - (sum + U * prob-left),
                              L * Reachable-Prob(Y)))
    if (sum + U * prob-left) ≤ P
       return(P)
return(sum)
```

To calculate this expected payoff `Mixed-set` examines every member Y of `partition(X)` in order. If Y is a leaf then the leaf's weighted payoff (payoff multiplied by reachable probability) is added to the running sum. If Y is a partial information set then the expected payoff is found by a recursive call to `OPPA` and added to the running sum. In either case the value of `prob-left` is updated to always contain the total reachable probability of the portion of X yet to be explored. This value is used in calculating the new pruning value that is passed to `OPPA`. The new pruning value is never set to be less than the minimum possible value of the information set being explored which is that information set's reachable probability times the lower bound.

If the pruning condition is satisfied then either chance node pruning or information set pruning has occurred. In this case the expected payoff of X is guaranteed to be less than or equal to P so no further exploration is needed and

P is returned. Otherwise the final value of the sum of expected payoffs is returned.

`Mixed-set` corresponds to the second line in the recursive statement of the One-Player Algorithm (see equation 2.1). The sole addition is the pruning condition which implements both chance node and information set pruning.

3.3 Proof of Correctness

Our proof will consist of four parts, two theorems and two lemmas. The lemmas simply assert that OPA will never return an expected payoff that is less than the lower bound value L or more than the upper bound U. Both lemmas are proven by induction on the depth of recursion of OPA.

The first theorem is by far the most complex of the four parts. This theorem states that given a pruning value P if the expected payoff of a set of nodes X as calculated by OPA is greater than P then OPPA will return the same value as OPA. If the expected payoff returned by OPA is less than or equal to P then OPPA will return P instead. This proof is by induction on the number of recursive calls necessary for OPPA to explore the set of nodes X.

The final and most important theorem combines the lemmas and previous theorem to state that OPPA when called with the lower bound as an initial pruning value always returns the same expected payoff that is calculated by OPA.

3.3.1 Definitions

Since the following four proofs all make use of many similar variables we will define them here rather than defining them separately for each proof.

- Γ is any one player game which meets the following restrictions.
  - The lower bound L is a real number such that for every leaf x in Γ

$$h(x) \geq L$$

  - The upper bound U is a real number such that for every leaf x in Γ

$$h(x) \leq U$$

- The S-operator, S, is defined as $S(X) = \cup_{x \in X} T(x)$ where $T(x)$ replaces a chance node with its children,

$$T(x) = \begin{cases} \{x\} & \text{if } x \text{ is a player node} \\ S(\{y | y \text{ is a child of } x\}) & \text{if } x \text{ is a chance node} \end{cases}$$

- The One-Player Algorithm, V, is recursively defined as

$$V(X) = \begin{cases} \max\{V(S(Y)) \mid Y \text{ is a child of } X\} & \text{if } X \text{ is a partial information set} \\ \sum_{\substack{x \in X \\ x \text{ a leaf}}} p(x)\, h(x) + \sum_{Y \in \textit{info-sets}(X)} V(Y) & \text{otherwise} \end{cases}$$

- The One-Player Pruning Algorithm, $OPPA$, is represented by pseudo-code

```
OPPA(X,P)
    if Partial-Information-Set(X)
       Max-Set(X,P)
    else
       Mixed-Set(X,P)

Max-Set(X,P)
       best = P
       for each Y a child of X
           temp = OPPA(S(Y),best)
           if temp > best
              if temp = (U * Reachable-Probability(X))
                 return(U * Reachable-Probability(X))
              best = temp
       return(best)

Mixed-Set(X,P)
         sum = 0
         prob-left = Reachable-Prob(X)
         Z = partition(X)
         for each member Y of Z
             if Y is an atom
                prob-left = prob-left - p(Y)
                sum = sum + h(Y) * p(Y)
             else
                prob-left = prob-left - Reachable-Prob(Y)
                sum = sum + OPPA(Y,Max(P - (sum + U * prob-left),
```

```
                                        L * Reachable-Prob(Y)))
        if (sum + U * prob-left) ≤ P
            return(P)
    return(sum)
```

3.3.2 Lemma 1

Lemma 1 *Given L, V, and S as defined above and any set of nodes X in Γ,*

$$V(S(X)) \geq \sum_{x \in S(X)} p(x) * L$$

Proof: The proof is by induction on the number of recursive calls to V necessary to compute $V(S(X))$.

Base Case: In the base case there is only one call to V needed to calculate $V(S(X))$ which means that $S(X)$ is a set of leaves. By the definition of V we know that in this case $V(S(X)) = \sum p(x) * h(x)$. Using the definition of L we can write this as $V(S(X)) \geq \sum p(x) * L$.

Inductive Case: We will separate the inductive case into two subcases: when $S(X)$ is a partial information set and when $S(X)$ is not a partial information set.

Case 1: $S(X)$ is a partial information set.

The definition of V states that in this case

$$V(S(X)) = max\{V(S(Y))|Y \text{ is a child of } S(X)\}$$

Let Y_{max} be the child of $S(X)$ on which the max is achieved. Since the calculation of $V(S(Y_{max}))$ requires one less recursive call, the inductive

hypothesis says that $V(S(Y_{max})) \geq \sum_{y \in S(Y)} p(y) * L$. Since the total reachable probability of each child is equal to the total reachable probability of the parent,

$$V(S(X)) \geq \sum_{y \in S(Y_{max})} p(y) * L = \sum_{x \in S(X)} p(x) * L$$

.

Case 2: $S(X)$ is not a partial information set.

The definition of V states that in this case

$$V(S(X)) = \sum_{x \text{ is a leaf}} p(x) * h(x) + \sum_{Y \in \text{ info set}(X)} V(Y)$$

Using the definition of L we have

$$V(S(X)) \geq \sum_{x \text{ is a leaf}} p(x) * L + \sum_{Y \in \text{ info set}(X)} V(Y)$$

Furthermore the calculation of $V(Y)$ takes one less recursive call than $V(S(X))$ and therefore the inductive hypothesis says that for each information set $V(Y) \geq \sum_{y \in Y} p(y) * L$. This gives us

$$V(S(X)) \geq \sum_{x \text{ is a leaf}} p(x) * L + \sum_{Y \in \text{ info set}(X)} \sum_{y \in Y} p(y) * L$$

Since every node in $S(X)$ is either a leaf or a member of one of the partial information sets the above equation is equivalent to

$$V(S(X)) \geq \sum_{x \in S(X)} p(x) * L$$

Thus in either case lemma 1 is shown to be true.

□

3.3.3 Lemma 2

Lemma 2 *Given U, V, and S as defined above and any set of nodes X in Γ,*

$$V(S(X)) \leq \sum_{x \in S(X)} p(x) * U$$

Proof: The proof of Lemma 2 is very similar to the proof of Lemma 1.

Base Case: In the base case there is only one call to V needed to calculate $V(S(X))$ which means that $S(X)$ is a set of leaves. By the definition of V we know that in this case $V(S(X)) = \sum p(x) * h(x)$. Using the definition of U we can write this as $V(S(X)) \leq \sum p(x) * U$.

Inductive Case: We will separate the inductive case into two subcases: when $S(X)$ is a partial information set and when $S(X)$ is not a partial information set.

Case 1: $S(X)$ is a partial information set.

The definition of V states that in this case

$$V(S(X)) = max\{V(S(Y))|Y \text{ is a child of } S(X)\}$$

Let Y_{max} be the child of $S(X)$ on which the max is achieved. Since the calculation of $V(S(Y_{max}))$ requires one less recursive call, the inductive hypothesis says that $V(S(Y_{max})) \leq \sum_{y \in S(Y)} p(y) * U$. Since total reachable probability of each child is equal to the total reachable probability of the parent,

$$V(S(X)) \leq max\{\sum_{y \in S(Y)} p(y) * U|Y \text{ is a child of } S(X)\} = \sum_{x \in S(X)} p(x) * U$$

.

Case 2: $S(X)$ is not a partial information set.

The definition of V states that in this case

$$V(S(X)) = \sum_{x \text{ is a leaf}} p(x) * h(x) + \sum_{Y \in \text{ info set}(X)} V(Y)$$

Using the definition of U we have

$$V(S(X)) \leq \sum_{x \text{ is a leaf}} p(x) * U + \sum_{Y \in \text{ info set}(X)} V(Y)$$

Furthermore the calculation of $V(Y)$ takes one less recursive call than $V(S(X))$ and therefore the inductive hypothesis says that for each information set $V(Y) \leq \sum_{y \in Y} p(y) * U$. This gives us

$$V(S(X)) \leq \sum_{x \text{ is a leaf}} p(x) * U + \sum_{Y \in \text{ info set}(X)} \sum_{y \in Y} p(y) * U$$

Since every node in $S(X)$ is either a leaf or a member of one of the partial information sets the above equation is equivalent to

$$V(S(X)) \leq \sum_{x \in S(X)} p(x) * U$$

Thus in either case lemma 2 is shown to be true.

□

3.3.4 Theorem 3

Theorem 3 *Given X, V, and $OPPA$ as defined above*

$$OPPA(S(X), P) = \begin{cases} V(S(X)) & \text{if } V(S(X)) > P \\ P & \text{if } V(S(X)) \leq P \end{cases}$$

Proof: We will prove theorem 3 by induction on the number of recursive calls to $OPPA$ needed to calculated $OPPA(S(X), P)$.

Base Case: In our base case there is only one call to $OPPA$ needed to explore $S(X)$. Because of this $OPPA$ must not call `Max-Set` to explore $S(X)$ in the base case, since `Max-Set` requires at least one additional call to $OPPA$. Therefore in the base case $OPPA$ will call `Mixed-Set` to explore $S(X)$ which means that $S(X)$ cannot be a single partial information set.

Furthermore if $S(X)$ contains any partial information sets then pruning must occur before they are reached since exploring each partial information set requires an additional call to $OPPA$. Thus when only one call to $OPPA$ is needed to explore $S(X)$ we know that $S(X)$ contains leaves that are explored and may or may not contain partial information sets which are never explored.

If pruning does not occur then $S(X)$ must consist of only leaves in which case $\texttt{Mixed-Set}(S(X), P) = \sum_{x \in S(X)} p(x) * h(x)$. The definition of V states that when $S(X)$ consists of only leaves $V(S(X)) = \sum_{x \text{ is a leaf}} p(x) * h(x)$. Thus it is obvious that when pruning does not occur $\texttt{Mixed-Set}(S(X), P) = V(S(X))$.

At this point it is useful to define sum_i as the value of the variable `sum` in `Mixed-set` after the ith leaf has been explored. Notice that $sum_i = \sum_{x_1}^{x_i} p(x) * h(x) = V(\{x_1, .., x_i\})$.

If pruning does occur then the pruning condition $sum_i + (U * probability - left) \leq P$ has been satisfied before exploring any partial information set. By lemma 2 after any number of leaves have been explored

$$V(\{x_1, .., x_i\}) + (U * probability - left)$$

$$= sum_i + (U * probability - left)$$
$$\geq V(S(X))$$

So if the pruning condition is satisfied then $V(S(X)) \leq P$ in which case `Mixed-Set` should return P which it does.

Combining these two cases results in,

$$OPPA(X,P) = \begin{cases} \sum p(x) * h(x) & \text{if } V(S(X)) > P \\ P & \text{if } V(S(X)) \leq P \end{cases}$$

and since $V(S(X)) = \sum p(x) * h(x)$ when $V(S(X)) > P$ this proves the base case.

Inductive Case: We will divide the inductive case into two subcases,

1. $S(X)$ is a partial information set.
2. $S(X)$ is not a partial information set.

Case 1: $S(X)$ is a partial information set.

When $S(X)$ is a partial information set $OPPA(S(X),P) = \texttt{Max-Set}(S(X),P)$ and

$$V(S(X)) = max\{V(S(Y))|Y \text{ is a child of } S(X)\}$$

To prove theorem 3 in Case 1 we will show that $OPPA(S(X),P)$ calculates $max\{P, V(S(X))\}$. This is equivalent to proving theorem 3 since if $V(S(X)) > P$ then $max\{P, V(S(X))\} = V(S(X))$ and if $V(S(X)) \leq P$ then $max\{P, V(S(X))\} = P$. Therefore $max\{P, V(S(X))\}$ returns the exact value that we would like $OPPA(S(X),P)$ to return.

Notice that each child Y_i of $S(X)$ is explored through a call to $OPPA(S(Y_i), P)$. Since each of these requires one less recursive call to $OPPA$ the inductive hypothesis says that for every child Y_i of $S(X)$

$$OPPA(S(Y_i), P) = \begin{cases} V(S(Y_i)) & \text{if } V(S(Y_i)) > P \\ P & \text{if } V(S(Y_i)) \leq P \end{cases}$$

Since $V(S(X)) = max\{V(S(Y))|Y \text{ is a child of } S(X)\}$ if $max\{P, V(S(X))\} = P$ then for every child of $S(X)$ it must be true that $V(S(Y_i)) \leq P$. Therefore every call to $OPPA(S(Y_i), P)$ will be less than or equal to P. If this happens then the test `temp > best` in `Max-Set` will never be true and the initial value of `best` which is P will be returned. Thus if $max\{P, V(S(X))\} = P$ is true $OPPA$ will return P which is what is required by theorem 3.

If $max\{P, V(S(X))\} > P$ then there must be some child Y_{max} of $S(X)$ such that $V(S(X)) = V(S(Y_{max})) > P$. By the definition of V if $V(S(X)) = V(S(Y_{max}))$ the value of $V(S(Y_{max}))$ is greater than or equal to the the value of $V(S(Y_i))$ for any child $Y_i \neq Y_{max}$. Since $V(S(Y_{max})) > P$ the inductive hypothesis says that $OPPA(S(Y_{max}), P) = V(S(Y_{max}))$ and furthermore the value of $OPPA(S(Y_{max}), P)$ is greater than or equal to the value of $OPPA(S(Y_i), P)$ for any child $Y_i \neq Y_{max}$.

When the first child of $S(X)$ for which $OPPA(S(Y), P) = V(S(Y_{max}))$ is explored by `Max-Set` the variable `temp` will be assigned the value $OPPA(S(Y), P)$ which equal $V(S(Y_{max}))$ which in turn equals $V(S(X))$. Since this is the first child for which $OPPA(S(Y), P) = V(S(Y_{max}))$, the `temp > best` test will be true. If $OPPA(S(Y), P) = \sum_{x \in S(X)} p(x) * U$ then immediate pruning will occur and $\sum_{x \in S(X)} p(x) * U$ will be returned. Since $\sum_{x \in S(X)} p(x) * U = OPPA(S(Y), P) = V(S(Y_{max})) = V(S(X))$, theorem 3 will

be satisfied. If immediate pruning does not occur then
$OPPA(S(Y), P) = V(S(Y_{max})) = V(S(X))$ is returned which also satisfies theorem 3.

Therefore if $S(X)$ is a partial information set then
$OPPA(S(X), P) = max\{P, V(S(X))\}$ and theorem 3 is proven.

Case 2: $S(X)$ is not a partial information set.

In this case $OPPA(S(X), P) =$ `Mixed-Set`$(S(X), P)$ and

$$V(S(X)) = \sum_{x \text{ is a leaf}} p(x) * h(x) + \sum_{Y \in \text{ info set}} V(Y)$$

.

Again we will use sum_i to represent the value of the variable `sum` in `Mixed-set` after the ith member of Z has been explored. We will prove by induction on the number of members of Z that

$$Mixed - Set(S(X), P) = \begin{cases} V(S(X)) & \text{if } V(S(X)) > P \\ P & \text{if } V(S(X)) \leq P \end{cases}$$

Base Case: The base case is when Z contains only one member.

If the single member of Z is a leaf then $sum_1 = p(x_1) * h(x_1)$ and
$V(S(X)) = p(x_1) * h(x_1)$ so $sum_1 = V(S(X))$. Since Z has only one member, when the pruning condition is reached `prob-left` will equal 0. Thus the pruning condition is reduced to $sum_1 \leq P$. If the pruning condition is true then
$V(S(X)) \leq P$ is also true and P is correctly returned. Otherwise
$sum_1 = V(S(X))$ is returned as specified by theorem 3.

If the single member of Z is a partial information set Y then
$sum_1 = OPPA(Y, max(P - (sum + U * prob - left), L * Reachable - Prob(Y)))$.

The calculation of the value of Y involves one less recursive call and therefore the top level inductive hypothesis applies.

If $P - (sum + U * prob - left) \geq L * Reachable - Prob(Y)$ then the recursive call is actually $OPPA(Y, P)$ since $sum = 0$ and $prob - left = 0$. In this case the top level inductive hypothesis states

$$OPPA(Y,P) = \begin{cases} V(Y) & \text{if } V(Y) > P \\ P & \text{if } V(Y) \leq P \end{cases}$$

which matches theorem 3 since $V(S(X)) = V(Y)$ and
$OPPA(S(X), P) = OPPA(Y, P)$.

If $P - (sum + U * prob - left) < L * Reachable - Prob(Y))$ then by the top level inductive hypothesis and lemma 1,
$OPPA(Y, L * Reachable - Prob(Y)) = V(Y)$. Thus when the pruning condition is reached $sum_1 = V(Y) = V(S(X))$. If the pruning condition is not true then $V(S(X))$ is returned which satisfies theorem 3. If the pruning condition is true then P is returned which again satisfies theorem 3.

Inductive Case: In the inductive case Z has n members. The inductive hypothesis states that theorem 3 holds for Z with $n - 1$ or fewer members thus we are assured that after $n - 1$ of the n members of Z have been explored either pruning has occurred and P has been returned or
$sum_{n-1} = V(S(\{Z_1, ..., Z_{n-1}\}))$.

If pruning has occurred then the last member of Z does not get explored since no matter what the value of this final member, sum_n will not be greater than P.

If pruning does not occur and the final member of Z is a leaf then
$sum_n = sum_{n-1} + p(x_n) * h(x_n)$ and

$V(S(X)) = V(S(\{Z_1, ..., Z_{n-1}\})) + p(x_n) * h(x_n)$. By the inductive hypothesis $sum_{n-1} = V(S\{(Z_1, ..., Z_{n-1}\}))$ so after the leaf has been explored $sum_n = V(S(X))$. When the pruning condition is reached(for the final time) `prob-left` is 0 so the pruning condition is reduced to $sum_n \leq P$. If the pruning condition is true then $V(S(X)) \leq P$ is also true and P is returned as specified by theorem 3. If pruning doesn't occur then $V(S(X)) > P$ and $sum_n = V(S(X))$ is returned which matches theorem 3. Therefore if the final member of Z is a leaf then theorem 3 holds.

If the final member of Z is a partial information set Y then

$$sum_n = sum_{n-1} + OPPA(Y, max(P - (sum + U * prob - left), L * Reachable - Prob(Y)))$$

The calculation of the value of Y requires one less recursive call so the top level induction hypothesis is applicable. Since Y is the last member of Z the value of $prob - left$ equals 0 and the variable `sum` is sum_{n-1} which equals $V(S(\{Z_1, ..., Z_{n-1}\}))$ as mentioned above.

If $P - (sum + U * prob - left) \geq L * Reachable - Prob(Y))$ then the top level inductive hypothesis can be stated as

$$OPPA(Y, P - sum_{n-1}) = \begin{cases} V(Y) & \text{if } V(Y) > P - sum_{n-1} \\ P - sum_{n-1} & \text{if } V(Y) \leq P - sum_{n-1} \end{cases}$$

If $V(Y) > P - sum_{n-1}$ then $sum_n = sum_{n-1} + V(Y) = V(S(X))$. In this case $V(S(X))$ is greater than P so the pruning condition will fail and $V(S(X))$ will be returned. If $V(Y) \leq P - sum_{n-1}$ then $sum_n = sum_{n-1} + P - sum_{n-1} = P$ which makes the pruning condition true and P is returned. This matches theorem 3 because $V(Y) \leq P - sum_{n-1}$ can be written as $V(Y) + sum_{n-1} \leq P$ which is $V(S(X)) \leq P$ since $V(Y) + sum_{n-1} = V(S(X))$.

If $P - (sum + U * prob - left) < L * Reachable - Prob(Y))$ the top level inductive hypothesis can be stated as

$$OPPA(Y, L * RP(Y)) = \begin{cases} V(Y) & \text{if } V(Y) > L * RP(Y) \\ L * RP(Y) & \text{if } V(Y) \leq L * RP(Y) \end{cases}$$

where RP is used as an abbreviation for $Reachable - Prob$. By lemma 1 $V(Y)$ cannot be less than $L * Reachable - Prob(Y)$ so in either case $OPPA(Y, L * Reachable - Prob(Y))$ returns $V(Y)$ and therefore $sum_n = sum_{n-1} + V(Y) = V(S(X))$. If the pruning condition is true then $V(S(X)) \leq P$ must also be true since $prob - left = 0$. Thus when $V(S(X)) \leq P$ is true P is returned as required by theorem 3. If the pruning condition fails then $sum_n = V(S(X))$ is returned and again the requirements of theorem 3 are met.

□

3.3.5 Theorem 4

Theorem 4 *Given* Γ, L, V, S, *and* $OPPA$ *as defined above*

$$OPPA(\{root\ of\ \Gamma\}, L) = V(S(\{root\ of\ \Gamma\}))$$

Proof: Our proof will be divided into two cases, where $V(S(\{\text{root of } \Gamma\})) > L$ and where $V(S(\{\text{root of } \Gamma\})) = L$. By lemma 1 and the fact that $p(\{\text{root of } \Gamma\}) = 1$ $V(S(\{\text{root of } \Gamma\}))$ cannot be less than L.

Case 1: $V(S(\{\text{root of } \Gamma\})) > L$

In this case $OPPA(\{\text{root of } \Gamma\}, L) = V(S(\{\text{root of } \Gamma\}))$ by theorem 3.

Case 2: $V(S(\{\text{root of } \Gamma\})) = L$

If $V(S(\{\text{root of } \Gamma\})) = L$ then $OPPA(\{\text{root of } \Gamma\}, L) = L$, again by theorem 3. Thus $V(S(\{\text{root of } \Gamma\})) = OPPA(\{\text{root of } \Gamma\}, L)$.

□

Chapter 4

The Hidden Information Model

Before any analysis of the effectiveness of the One Player Pruning Algorithm can be started a general model of one player games with hidden information must be decided on. The model described here, which we call the hidden information model, has many advantages. As described in the next section the hidden information model is based on the trees of actual hidden information games, in particular a class of card games. A second advantage is the simplicity of the model which aids in the implementation and generation of game trees of various sizes. And finally the hidden information model has the important advantage in that it allows us to focus on information set pruning by guaranteeing that no chance node pruning will occur.

4.1 Motivation

The hidden information model will be based on two very basic properties of card games; shuffling the deck and discovering a little of the hidden information at each move. In a game tree the process of shuffling the deck corresponds to a chance node at the root of the tree with a very large number of equally likely children. Each of the 52! children represents a possible configuration of the deck after it has been shuffled. In most card games none of the players know the configuration of the deck after shuffling, so all the children of the chance node (the entire first level of the tree) make up a single large information set.

In a similar fashion the trees in the hidden information model will have a chance node at the root with a large number of equally likely children in a single information set. We will refer to the number of children of this root chance node as B. Obviously it would be impossible to search game trees for which $B = 52!$ however the game trees in the hidden information model will maintain the flavor of the real games by assuring that B is much larger than the branching factor of any non-root node in the tree.

The second aspect of card games that the hidden information model mimics is the gradual process of discovering parts of the hidden information until the end of the game when everything is known. For example, in bridge and many other card games a play consists of each player uncovering one card. By observing which cards were uncovered a good player learns not only what cards were held but can also guess at what cards the opponent may or may not still have. Another excellent example is card counting which is calculating the probability that a certain card will be on top of the deck based on knowledge of which cards

have already been played. The amount of information that is gained at each play varies from game to game but to maintain perfect recall the player must at least gain the knowledge of which move he or she made.

The hidden information model will include as one of its parameters a measure of the amount of information that is learned with each play. This parameter, ι, will have a minimum value of 1 in which case the only information learned with a move is what move was made. The maximum value of ι will be B which will represent a game where all the hidden information is uncovered after one move. Thus when $\iota = B$ level 2 of the game tree will have one node in each information set. The next section will describe in more detail how the value of ι is used in the hidden information model.

4.2 The Hidden Information Model

As mentioned above the trees in the hidden information model have a chance node at the root with B equally likely children. All of these children are player nodes and beneath each is a complete subtree consisting entirely of player nodes. The parameter b is the branching factor of every node in the tree with the exception of the single chance node at the root (which has branching factor B). Thus every player node will be the root of a complete $b$ary tree. The depth of the entire tree from the chance node to the leaves is represented by d. Thus a tree in the hidden information model has only one chance node (at the root) and $B(\frac{b^d-1}{b-1})$ non-chance nodes including leaves.

As mentioned previously there are B player nodes all in a single information set at level 1 of a tree in the hidden information model. The value of the ι

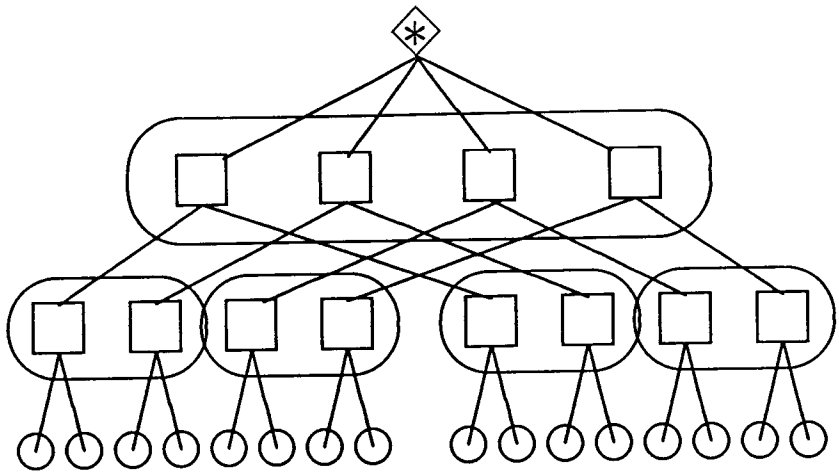

Figure 4.1: *The game tree in the hidden information model that is specified by the parameters* $\iota = 2$, $b = 2$, *and* $d = 3$. *The square nodes are player nodes and the circular nodes are leaves.*

parameter is used to determine how that large information set is successively broken into smaller information sets by successive moves. In particular, given an information set at any point in the game with n nodes, its children under each alternative will be broken into ι information sets each with $\frac{n}{\iota}$ nodes. For example in the tree for which $\iota = 2$ and $b = 2$, level 2 will consist of 4 information sets each with $\frac{B}{2}$ nodes (see figure 4.1).

If the value of B is defined as $B = \iota^{d-1}$ then the leaves naturally appear at level d of the tree when all the hidden information has been learned i.e. when the information sets would have only one node each. This also makes the value of B much larger than b in most cases[1]. This allows us to uniquely determine any game tree in the hidden information model by the three parameters ι, b, and d. By varying these parameters a wide variety of different trees can be generated.

[1]The $\iota = 1$ case is discussed below.

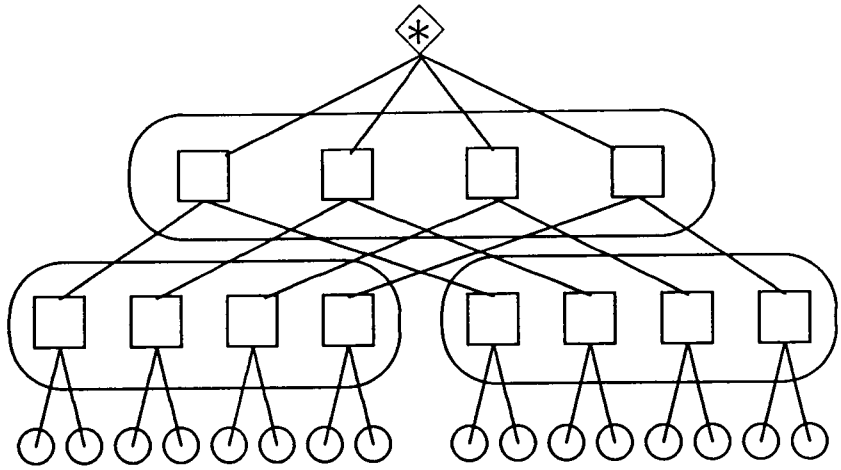

Figure 4.2: *The game tree in the $\iota = 1$ special case of the hidden information model that is specified by the parameters $\iota = 1$, $b = 2$, $d = 3$, and $B = 4$.*

Trees for which $\iota = 1$ are different from the other trees in the hidden information model for two reasons. First, the information sets do not get successively smaller with successive moves. Because of this there is not any natural way for the leaves to appear as there is for other values of ι. To avoid this problem when $\iota = 1$ the information sets remain completely intact until level d at which point they are broken into leaves (see figure 4.2).

The second difference between the $\iota = 1$ special case and the other trees in the hidden information model is that defining $B = \iota^{d-1}$ causes problems since the chance node will have only one child. To have hidden information there must be at least one information set containing two or more children. In addition this invalidates the $B > b$ similarity to card games. To avoid this game trees in the hidden information model for which $\iota = 1$ are described by four parameters, ι,b, and d as above plus a value for B.

4.3 Advantages of the Hidden Information Model

Perhaps the most important advantage of the hidden information model is its similarity to the game trees of card games. The model makes some of the important aspects of these hidden information games clear through the ι parameter. Testing the One Player Pruning Algorithm on the hidden information model should provide an approximation of how effective the algorithm would be on card games and other real hidden information games.

A second advantage is that the trees in the hidden information model all maintain perfect recall. Since nodes in the same information set always come from the same alternative of parents who are also in the same information set there is no way that the perfect recall property can be lost. This is important as OPPA requires perfect recall to guarantee the calculation of the optimal expected payoff.

A third advantage stems from the conjecture that any game tree with chance nodes can be represented by a similar game tree with a single chance node at the root. Thus the trees in the hidden information model might in some sense be equivalent to game trees with many chance nodes. A formal statement of this conjecture would be

Conjecture 1 *Given any game tree Γ with chance nodes, a new game tree Γ' can be constructed by combining all the chance nodes in Γ into a single chance node and at the root of Γ' and making other minor modifications. A strategy π on Γ can be easily modified into π' on Γ' such that π and π' have the same*

expected payoff.

This conjecture is believed to be true however the proof will be left as a area for future work.

A final advantage is that chance node pruning will never occur when exploring trees in the hidden information model. For chance node pruning to occur a pruning bound must be passed to a chance node at some point in the tree. The trees in the hidden information model have only one chance node at the root where it cannot receive a pruning bound. Thus the analysis of the effectiveness of the One Player Pruning Algorithm in the next two chapters focuses on information set pruning. This is appropriate as the effectiveness of chance node pruning has already been analyzed[1].

Chapter 5

Best Case Analysis

Rather than analyze the pruning of the One Player Algorithm as a whole this chapter will concentrate on information set pruning. The other two pruning techniques used in OPPA have already been analyzed and the reader is referred to Ballard[1] for an analysis of chance node pruning and Korf[6] for immediate pruning.

5.1 Best case game trees

At the beginning of his analysis of chance node pruning Ballard states that

> *specifying a subclass for study...involves deciding on (a) the* overall structure *of the trees; (b) a way of assigning* values *to the leaves; and (c) the* criterion *to be measured.*

As the goal of this chapter is to analyze information set pruning in the best case we must specify a subclass that represents the trees on which the information set pruning in OPPA will be most effective. We will argue informally that the specified subclass is the subclass on which information set pruning is maximally effective but we will not give a formal proof of this.

The overall structure of our game trees will be as specified by the hidden information model described in the previous chapter. Thus the best case analysis will involve ι, b, and d, the three parameters of the hidden information model[1]. For the criterion to be measured we will choose the number of leaves examined in the exploration of the tree. This choice of criterion follows a trend started by the analysis of alpha beta[5] and continued by the analysis of *minimax[1] and multiplayer alpha beta[6]. Therefore our best case analysis of information set pruning will specify the number of leaves examined in the best case as a function of ι, b, and d.

To specify the best case subclass completely it remains only to choose a way of assigning values to the leaves. As this is to be a best case analysis the leaf values must be chosen to give the most pruning possible. To do this we will divide the leaves into two types. Critical leaves will be the "leftmost" leaves on which the pruning values for the rest of the tree are based. In the best case the pruning bound should be high so critical leaves will be assigned the upper bound of 10. Non-critical leaves will be all the leaves that are not critical leaves. Maximum pruning will occur when non-critical leaves have very low values so they will be assigned the lower bound of 0.

[1]The $\iota = 1$ special case will not be dealt with.

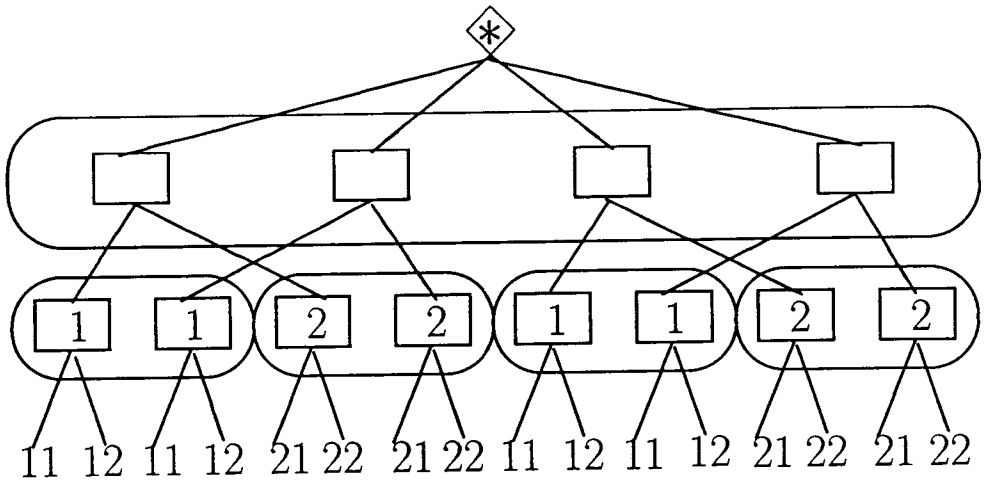

Figure 5.1: *The game tree specified by $\iota = 2$, $b = 2$, and $d = 3$ with each node labeled according to our alternate Dewey decimal system.*

In defining critical leaves it is useful to assign coordinate numbers to the nodes of the tree as in the "Dewey decimal system"[4]. In this system each player node in the game tree is assigned a sequence of positive numbers representing the path taken to that node from its ancestor player node at level 1. The player nodes at level 1 will all be assigned the empty sequence. Any other node in the tree whose parent's sequence is $a_1, ..., a_n$ will be assigned the sequence $a_1, ..., a_n, i$ where the ith alternative was taken from the parent to reach that node. So if a node's coordinate number was 2,1,3 it would be reached by taking the second alternative from a player node at level 1, then the first alternative from a node labeled 2 at level 2, and finally the third alternative from a node labeled 2,1 at level 3 (see figure 5.1).

Please note that the labeling system described here is a slightly modified version of the true Dewey decimal system. In this system the node labels are not

distinct. In fact there are exactly B nodes with any given label[2], one in each of the B subtrees that has a level 1 player node as a root.

Given this alternate Dewey decimal system the critical leaves are defined to be the B leaves which are labeled by a sequence with 1 at every position[3]. The leaves that are not labeled with all 1's are non-critical leaves. A player node is a critical node if it is labeled with the empty sequence or all 1's and a critical information set is made up of critical nodes[4].

The critical leaves are important because they determine the pruning bounds passed to the rest of the tree. The larger the values of the critical leaves, the larger the pruning bounds for the rest of the tree will be which will cause more pruning. Thus we will give all the critical leaves the value $U = 10$.

Making the non-critical leaf values small insures that once the first of a set of non-critical leaves is examined by `Mixed-set` the pruning condition will be true and the rest of the set will be pruned. To allow this to happen we will assign the non-critical leaves the value $L = 0$.

5.2 Pruning Analysis

The best case analysis is most easily explained by representing the number of leaves examined as a pair of recursive functions. Once these functions have been

[2]Since $\iota > 1$ the value of B is ι^{d-1}.

[3]Our critical leaves will consist of only what Knuth's analysis of alpha beta calls type 1 critical leaves.

[4]In the hidden information model no information set can consist of a combination of critical and non-critical nodes.

explained they will be solved to give a closed form equation for the number of leaves examined in the best case.

The first recursive function, $E_c(n)$, will calculate the number of leaves examined in exploring a critical information set n levels above the leaves. The second recursive function, $E_{nc}(n)$, calculates the number of leaves examined in exploring a non-critical information set n levels above the leaves. In the hidden information model there is only one critical information set at each level and it can be shown that the same number of leaves are examined while exploring each non-critical information set on a given level. Thus E_c will give the same value for all the critical information sets on a level and $E_{nc}(n)$ will do the same for the non-critical information sets.

The recursive functions for E_c and E_{nc} are (explanations to follow)

$$E_c(n) = \iota E_c(n-1) + (b-1)E_{nc}(n-1) \tag{5.1}$$

$$E_{nc}(n) = bE_{nc}(n-1) \tag{5.2}$$

$$E_c(0) = E_{nc}(0) = 1$$

Notice that when $n = 0$ the information set to be explored is 0 levels from the leaves. In this case it is actually a leaf rather than an information set that is being explored. Obviously the exploration of a leaf requires one leaf to be examined which explains the two base cases.

To explore a critical information set `Max-set` will make a call to `OPPA` for each of the b alternatives each time passing ι information sets to be explored. The first call will consist of the ι critical information sets under the first alternative. Each of these critical information sets is one level closer to the leaves and therefore

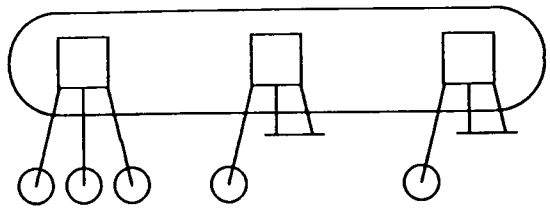

Figure 5.2: *The leaves examined for a critical information set 1 level above the leaves when $\iota = 3$ and $b = 3$. The ι "leftmost" leaves are examined and for the other $b - 1$ alternatives 1 leaf is examined and the $\iota - 1$ others are pruned.*

requires the examination of $E_c(n-1)$ leaves. Thus exploring the first alternative of a critical information set requires $\iota E_c(n-1)$ leaf examinations (see figure 5.2).

For each of the other $(b-1)$ alternatives `OPPA` will call `Mixed-set` which explores the first information set and then prunes the remaining $\iota - 1$ information set. Since the information sets that are examined are not under the first alternative they are not critical information sets and require $E_{nc}(n-1)$ leaf examinations each. Thus for each of these $(b-1)$ alternatives one non critical information set is explored and the rest are pruned. Thus exploring the $(b-1)$ alternatives that follow the first alternative requires $(b-1)E_{nc}(n-1)$ leaves to be examined.

Therefore the exploration of a critical information set n levels above the leaves requires $\iota E_c(n-1)$ leaves for the first alternative and $(b-1)E_{nc}(n-1)$ for the remaining $(b-1)$ alternatives as shown by equation 5.1.

To explore a non-critical information set `Max-set` again will make b calls to `OPPA` which in turn calls `Mixed-set`. For each of the b calls `Mixed-set` will explore the first of the ι information sets and prune the rest (see figure 5.3). Since these information sets are all the children of a non-critical information set they are also non-critical information sets. Thus the exploration of a non-critical

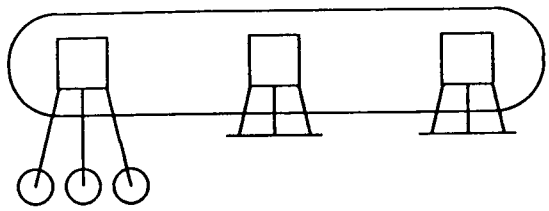

Figure 5.3: *The leaves examined for a non-critical information set 1 level above the leaves when $\iota = 3$ and $b = 3$. For each alternative 1 leaf is examined and the $\iota - 1$ others are pruned.*

information set n levels above the leaves requires the exploration of one non-critical information set for each of b alternatives for a total of $bE_{nc}(n-1)$ leaves examined as shown by equation 5.2.

Using these recursive functions the total number of leaves examined in exploring a tree can be found by solving E_c on the single critical information set at level 1 of the tree which is $d-1$ levels above the leaves. Thus the number of leaves examined in exploring a game tree from the hidden information model is $E_c(d-1)$. The correctness of this analysis will be verified experimentally in the next section.

It is easy to solve equation 5.2 to give the equivalent non-recursive formula,

$$E_{nc}(n) = b^n \tag{5.3}$$

Substituting this into equation 5.1 results in,

$$E_c(n) = \iota E_c(n-1) + (b-1)b^{n-1}$$

which can in turn be solved to give a non-recursive formula,

$$E_c(n) = \iota^n + (b-1)\sum_{j=0}^{n-1} b^j \iota^{n-j-1} \tag{5.4}$$

The equivalence of equation 5.1 and equation 5.4 can easily be proven by induction. The base case of equation 5.1 says that $E_c(0) = 1$ and for equation 5.4 $E_c(0) = \iota^0 + (b-1)\sum_{j=0}^{-1} b^j \iota^{n-j-1} = 1$. For the inductive step

$$\begin{aligned} E_c(n+1) &= \iota E_c(n) + (b-1)b^n \\ &= \iota(\iota^n + (b-1)\sum_{j=0}^{n-1} b^j \iota^{n-j-1}) + (b-1)b^n \\ &= \iota^{n+1} + (b-1)\sum_{j=0}^{n-1} b^j \iota^{(n+1)-j-1} + (b-1)b^n\iota^0 \\ &= \iota^{n+1} + (b-1)\sum_{j=0}^{n} b^j \iota^{(n+1)-j-1} \end{aligned}$$

To simplify equation 5.4 the summation is reduced to a closed form expression,

$$\begin{aligned} \sum_{j=0}^{n-1} b^j \iota^{n-j-1} &= \iota^{n-1}\sum_{j=0}^{n-1}(\frac{b}{\iota})^j \\ &= \iota^{n-1}(\frac{(\frac{b}{\iota})^n - 1}{\frac{b}{\iota} - 1}) \\ &= \frac{b^n - \iota^n}{\iota(\frac{b}{\iota} - 1)} \\ &= \frac{b^n - \iota^n}{b - \iota} \end{aligned}$$

Substituting this into equation 5.4 gives

$$\begin{aligned} \iota^n + (b-1)\frac{b^n - \iota^n}{b-\iota} &= \frac{\iota^n b - \iota^{n+1} + b^{n+1} - b\iota^n - b^n + \iota^n}{b-\iota} \\ &= \frac{b^{n+1} - b^n + \iota^n - \iota^{n+1}}{b-\iota} \\ &= \frac{b^n(b-1) + \iota^n(1-\iota)}{b-\iota} \end{aligned}$$

Thus the One Player Pruning Algorithm relying solely on information set pruning will need to examine

$$\frac{b^{d-1}(b-1) + \iota^{d-1}(1-\iota)}{b-\iota} \tag{5.5}$$

leaves in the exploration of the best case trees described above. This is clearly much less than the total number of leaves which is $(\iota b)^{d-1}$. This seems to bear some resemblance to the equation for the number of leaves alpha beta explores in the best case. Knuth proves that in the best case alpha beta will explore $2b^{d/2}$ leaves out of b^d total leaves. When ι is similar to b and d is large equation 5.5 says that approximately db^d leaves will be explored out of b^{2d} total leaves.

5.3 Experimental verification of the pruning analysis

To confirm the correctness of the analysis presented in the previous section a slightly modified version of the One Player Pruning Algorithm was implemented and tested on a large number of best case game trees. This will not prove that the subclass specified is strictly the best case. Instead the goal of the experiment is to check that equation 5.5 correctly predicts the number of leaves examined for the given subclass of trees.

To run this experiment the One Player Pruning Algorithm was implemented in Common Lisp (see Appendix A) as were the supporting functions needed to generate and explore the best case trees. Since the analysis above only takes information set pruning into account the lines in `Max-set` that perform immediate pruning were commented out[5].

The correctness of the implementation of OPPA was checked in two ways. First,

[5]It is easy to see that in the best case with immediate pruning only the B critical nodes are examined.

the program was run on some small game trees for which the correct expected payoff could be checked by hand. Second, two versions of the OPPA program, one with pruning and one with the pruning conditions removed, were run on identical large game trees and the returned expected payoffs were compared. In both cases the implementation was found to calculate the correct expected payoff.

To check the validity of the given analysis the implementation of OPPA was used to compare the predicted number of leaves examined to the actual number of leaves examined. All the best case trees in the hidden information model for which $2 \leq \iota \leq 8$, $2 \leq b \leq 8$, and $2 \leq d \leq 6$ were checked. When $\iota \neq b$ equation 5.5 was used to predict the number of leaves examined and equation 5.4 was used when $\iota = b$. On all 144 of these trees the predicted number of leaves exactly matched the actual number of leaves that the OPPA implementation examined. Some of the data from this experiment is presented in figure 5.4 and table 5.1.

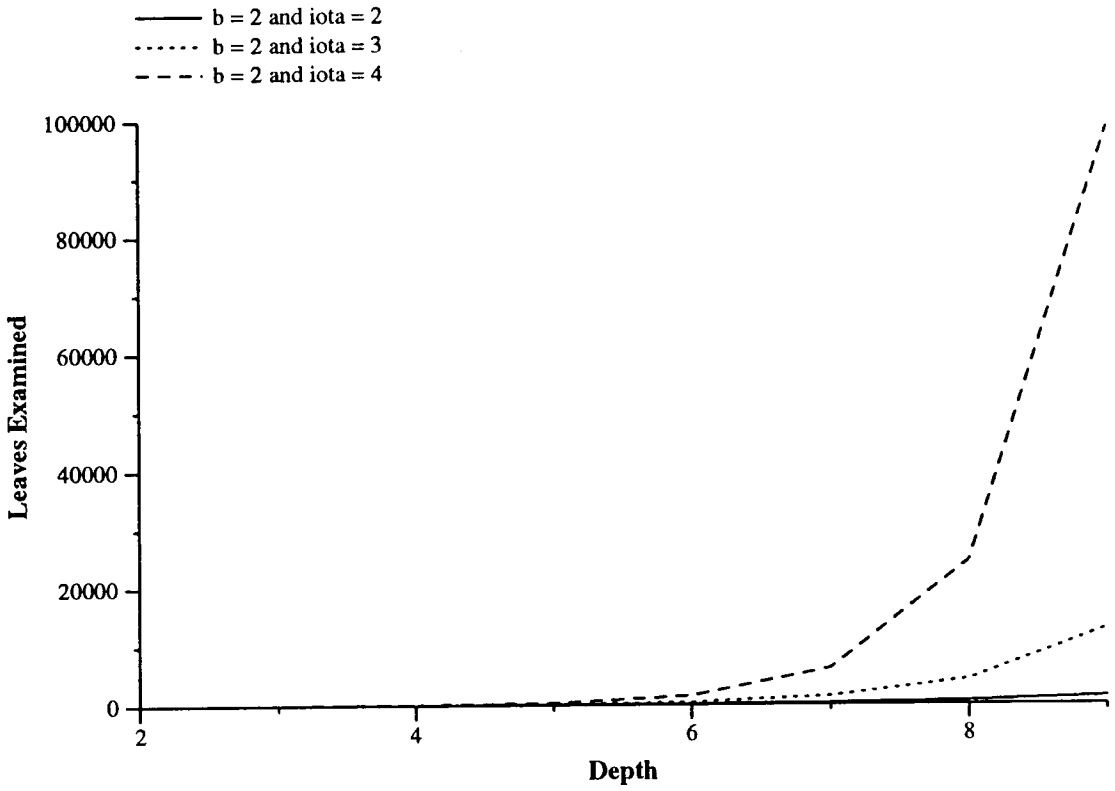

Figure 5.4: *The number of leaves examined in the best case for various values of* b, ι *and* d. *Because of the symmetry of equation 5.5 the data for* $b = 3$, $\iota = 2$ *is exactly the same as* $b = 2$, $\iota = 3$. *The same holds true for* $b = 4$, $\iota = 2$ *and* $b = 2$, $\iota = 4$.

Table 5.1: *The number of leaves examined in the best case for various values of b, ι and d.*

b	ι	d	Leaves Examined	Total Leaves	Percent Examined
2	2	2	3	4	75.0
		3	8	16	50.0
		4	20	64	31.3
		5	48	256	18.8
		6	112	1024	10.9
		7	256	4096	6.3
		8	576	16384	3.5
		9	1280	65536	2.0
2	10	2	11	20	55.0
		3	112	400	28.0
		4	1124	8000	14.1
		5	11248	$1.6 * 10^5$	7.0
		6	112496	$3.2 * 10^6$	3.5
		7	1124992	$6.4 * 10^7$	1.8
		8	11249984	$12.8 * 10^8$	0.9
		9	112499968	$25.6 * 10^9$	0.4
10	10	2	19	100	19.0
		3	280	10000	2.8
		4	3700	$1 * 10^6$	0.37
		5	46000	$1 * 10^8$	0.046
		6	$5.5 * 10^5$	$1 * 10^{10}$	0.0055
		7	$6.4 * 10^6$	$1 * 10^{12}$	0.00064
		8	$7.3 * 10^7$	$1 * 10^{14}$	0.000073
		9	$8.2 * 10^8$	$1 * 10^{16}$	0.0000082

Chapter 6

Average Case Analysis

As shown in the previous chapter information set pruning can make a large difference in the number of leaves examined in the exploration of a one player game tree with hidden information. In this chapter an analysis of the effectiveness of information set pruning in the average case will be done. To test this the number of leaves examined by the modified implementation of the One Player Pruning Algorithm in the exploration of game trees with random leaf values will be measured.

6.1 Experiment

The average case analysis of information set pruning will be an empirical analysis based on OPPA's behavior on a new subclass of trees. As with best case trees we will use the overall structure described by the hidden information model and use the number of leaves examined as a criterion of pruning success. The

average case subclass will differ from the best case subclass in the method of assigning leaf values. For the average case subclass the leaf values will be assigned randomly.

A number of different techniques for generating random leaf values in games trees have been suggested[1, 5, 6]. We will partially imitate the analysis of alpha beta[5] by choosing the value of each leaf independently, however unlike that analysis the leaf values will not be distinct. Thus whenever a leaf value is needed the standard Lisp random number generator is called on to supply a number between the lower bound of $L = 0$ and the upper bound of $U = 10$. The longest test run requires about 3 million random numbers which is well under the period of the random number generator.

The experiment will consist of four sets of trials with various values of ι and b. The first set of trials will be run on trees in the hidden information model for which $\iota = 2$, $b = 2$, and d ranges from 2 to 9. The results of this first trial will serve as a basis against which the other trials can be compared. For the second set of trials $\iota = 2$, $b = 10$ and d will range from 2 to 7. This should provide some insight into the effect the branching factor has on information set pruning. The third set of trials will explore the effect of the ι parameter by setting $\iota = 10$ while $b = 2$ and d ranges from 2 to 5. Finally the fourth set of trials will explore the very large trees for which $\iota = 10$ and $b = 10$ and d again ranges from 2 to 5. Larger values of d for the latter three cases were not possible due to the limits of Austin Kyoto Common Lisp and time.

As the leaves are randomly generated the number of leaves examined will vary from run to run. To account for this variation 5 different trees were generated

and explored for each set of ι, b, and d values[1].

6.2 Results

The results of each trial are summarized by a table and a graph. The complete data for each run is given in Appendix B. For each d value the tables give the average of the 5 runs, the total number of leaves in the tree, the percent of the total leaves examined on average and the 90% confidence interval. Each trial is also represented as a graph of leaves examined vs. depth. Included on each graph is an exponential curve fit which will be discussed in the analysis section.

6.3 Analysis

Given that the total number of leaves is $(\iota b)^{d-1}$ it seems reasonable to expect that an equation representing the number of leaves explored in the average case might look something like $x(\iota b)^{y(d-1)}$. Additional evidence for this is the success with which the exponential curve fitting routine was able to match the data.

The exponential equations (of the form $x(\iota b)^{y(d-1)}$) for each of the four trials are given in table 6.1. The variation in x and y suggest that $x(\iota b)^{y(d-1)}$ probably doesn't exactly capture the average case behavior but it does provide a reasonable estimate.

These equations show that information set pruning decreases the number of

[1]Exploring a tree for which $\iota = 10$, $b = 10$, and $d = 5$ took approximately 35 hours and thus only one tree of this size was tested.

Table 6.1: *The behavior of information set pruning in the average case as estimated by the exponential curve fits.*

		b = 2	b = 10
ι	2	$0.25(\iota b)^{0.97(d-1)}$	$0.49(\iota b)^{0.89(d-1)}$
	10	$0.05(\iota b)^{1.00(d-1)}$	$0.01(\iota b)^{0.99(d-1)}$

leaves that must be examined by a constant factor as expressed by the coefficient x. Unfortunately information set pruning seems to have a minor effect at best on the coefficient of the exponent y. A powerful pruning algorithm such as alpha beta will significantly reduce the coefficient of the exponent and therefore explore a smaller part of the tree as the depth increases. Information set pruning on the other hand will explore approximately the same fraction of the tree for all depths.

As shown in tables 6.2 and 6.3 and figures 6.1 and 6.2 the fraction of the total leaves examined decreases as b gets larger. In addition as b is increased the coefficient of the exponent decreases. As shown in tables 6.4 and 6.5 and figures 6.3 and 6.4 increasing ι seems to cause a larger fraction of the total number of leaves to be examined. Most card games will have large b and ι values. Thus for these games information set pruning could decrease the fraction of leaves examined by a constant factor which would be the same for trees of varying depths.

Results for $\iota = 2$, $b = 2$

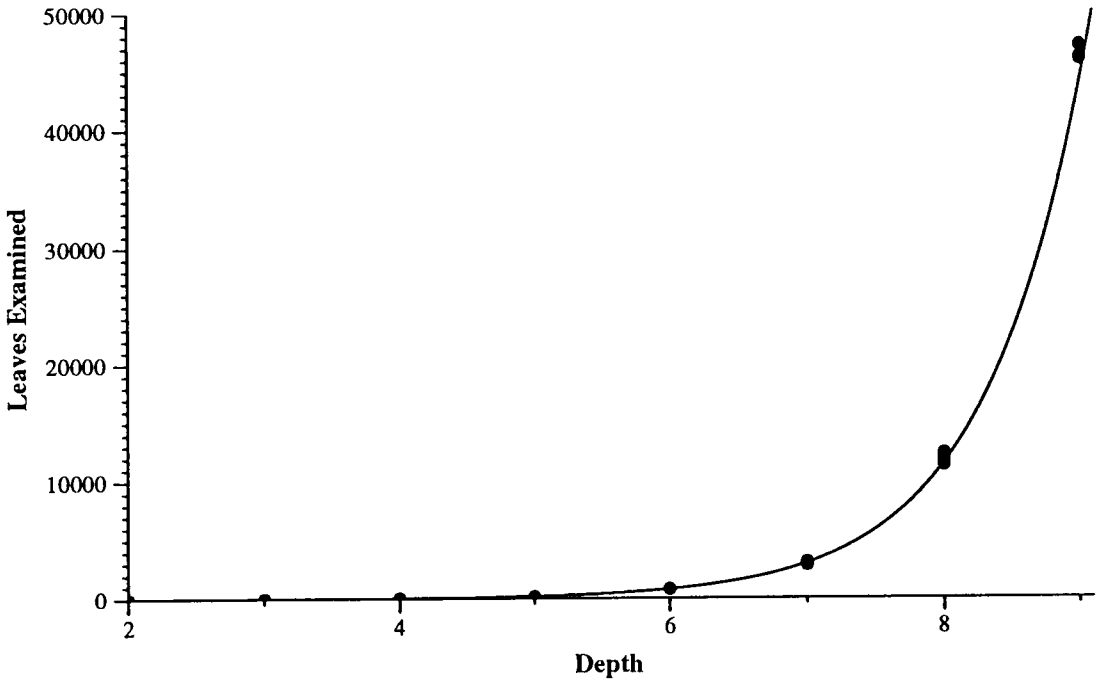

Figure 6.1: *Data for $\iota = 2$ and $b = 2$ with an exponential curve fit*

Table 6.2: *Results of the test runs in trial 1 with $\iota = 2$ and $b = 2$.*

Depth	Average	Total	% Examined	90% Confidence
2	3.8	4	95.0	(3.06 , 4.54)
3	14.6	16	91.3	(12.72 , 16.48)
4	50.4	64	78.8	(38.61 , 62.19)
5	188.8	256	73.8	(168.20 , 209.40)
6	779.0	1024	76.1	(730.85 , 827.15)
7	2989.4	4096	73.0	(2783.73 , 3195.07)
8	11825.4	16384	72.2	(11047.92 , 12602.88)
9	46384	65536	70.8	(45407.79 , 47360.21)

Results for $\iota = 2$, $b = 10$

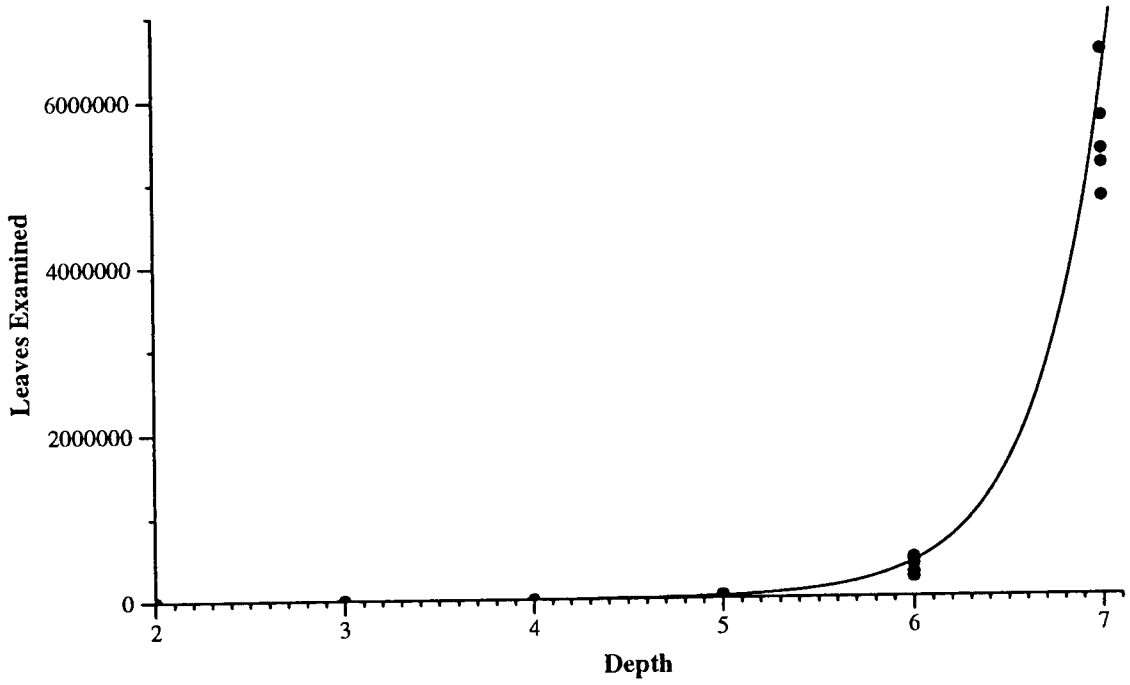

Figure 6.2: *Data for $\iota = 2$ and $b = 10$ with an exponential curve fit*

Table 6.3: *Results of the test runs in trial 1 with $\iota = 2$ and $b = 10$.*

Depth	Average	Total	% Examined	90% Confidence
2	14.8	20	74.0	(10.54 , 19.06)
3	183.6	400	45.9	(135.40 , 231.80)
4	2005.8	8000	25.1	(1628.88 , 2382.72)
5	33702.6	160000	21.1	(23362.62 , 44042.58)
6	365272.6	3200000	11.4	(198785.97 , 531759.23)
7	5500949.8	64000000	8.6	(4408465.59 , 6593434.01)

Results for $\iota = 10$, $b = 2$

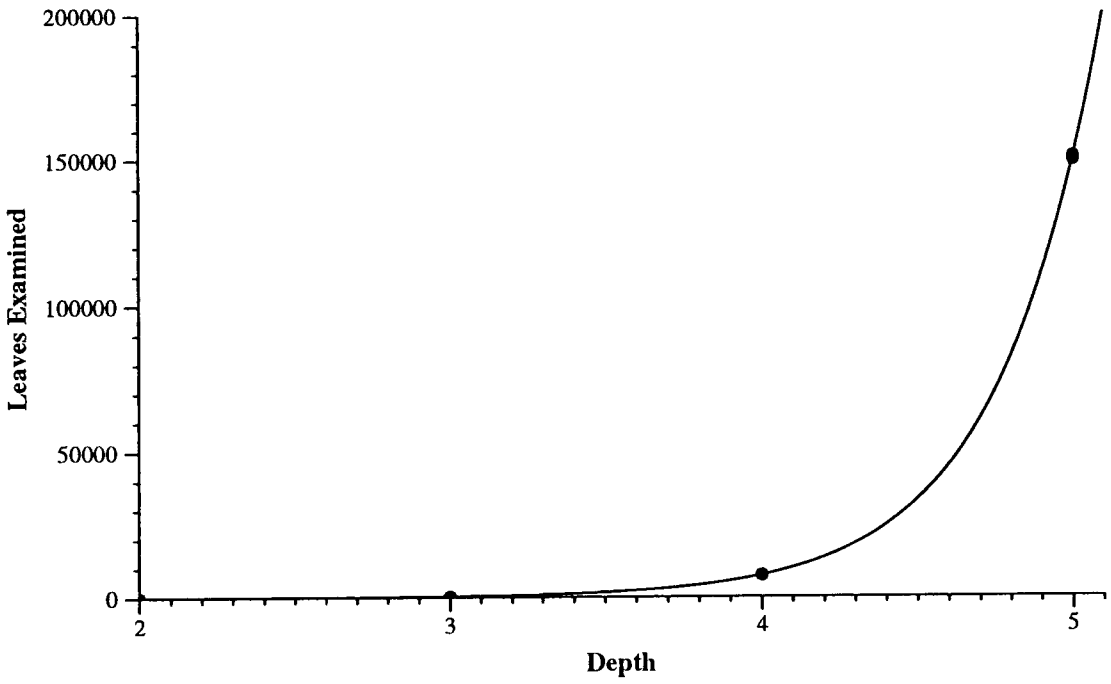

Figure 6.3: *Data for $\iota = 10$ and $b = 2$ with an exponential curve fit*

Table 6.4: *Results of the test runs in trial 1 with $\iota = 10$ and $b = 2$.*

Depth	Average	Total	% Examined	90% Confidence
2	19.4	20	97.0	(17.94 , 20.86)
3	377.4	400	94.4	(249.17 , 505.63)
4	7479.2	8000	93.5	(6504.14 , 8454.26)
5	149938.4	160000	93.7	(148963.34 , 150913.46)

Results for $\iota = 10$, $b = 10$

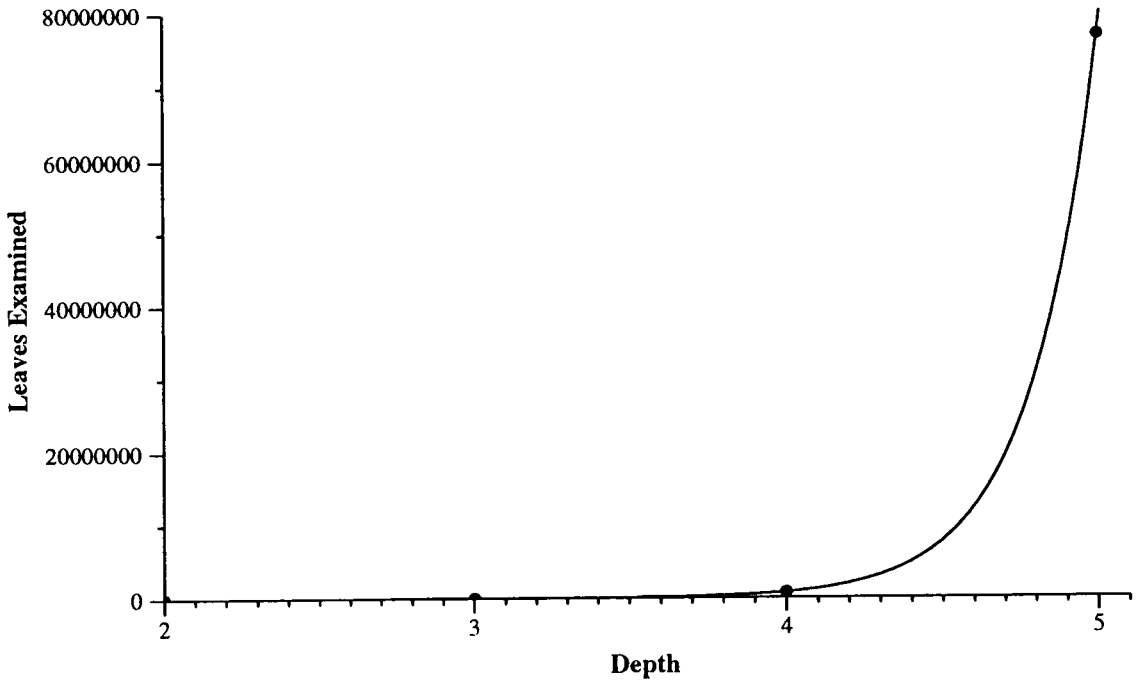

Figure 6.4: *Data for* $\iota = 10$ *and* $b = 10$ *with an exponential curve fit*

Table 6.5: *Results of the test runs in trial 1 with* $\iota = 10$ *and* $b = 10$. *No confidence interval is given for* $d = 5$ *as only one run of this depth was made.*

Depth	Average	Total	% Examined	90% Confidence
2	88.2	100	88.2	(85.04 , 91.36)
3	7711.6	10000	77.1	(7319.50 , 8103.70)
4	768944.6	1000000	76.9	(758966.56 , 778922.64)
5	76731628	100000000	76.7	

Chapter 7

Conclusion and Future Work

7.1 Future Work

The work in this thesis poses many new questions and suggests a number of areas for future work. One interesting open question is the optimality of information set pruning, i.e. is there pruning in one player hidden information games that the three pruning techniques in OPPA miss. The eventual goal of this work would be an algorithm that can be proven to explore the minimum amount of the tree necessary. Obviously such an optimal pruning algorithm for games with hidden information would have to include information set pruning.

The SSS* algorithm[9] does the same types of pruning as alpha beta but explores the tree in a best first fashion. This gives SSS* an advantage on certain types of game trees. Since OPPA is strictly a depth first search there is a corresponding best first type algorithm which might do more pruning than OPPA on certain types of trees. The development of this algorithm would be

interesting and might turn out to be successful on games with large ι values.

Another area that might be interesting to pursue would be the implementation of the One Player Pruning Algorithm on real hidden information games. Some examples of games that might be applicable are solitaire card games or casino blackjack. There are some difficult issues that would have to be addressed such as how to handle the very large information set that represents shuffling the cards.

An algorithm that explores two player games with hidden information is a very interesting and challenging area for future work. If such an algorithm could be developed it would probably lead to much stronger programs for playing card games.

Finally the conjecture presented in section 4.3 has been left unproven. Proving Conjecture 1 would probably be relatively easy and might suggest other classes of equivalent game trees.

7.2 Conclusion

In the introduction the three contributions that this thesis makes to the development of search algorithms for hidden information games were detailed. The first contribution was the description and analysis of information set pruning. In Chapter 5 the best case analysis of information set pruning proved that for certain trees in the hidden information model information set pruning can reduce the number of leaves that need to be examined to,

$$\frac{b^{d-1}(b-1)+\iota^{d-1}(1-\iota)}{b-\iota}$$

Chapter 6 showed through empirical experimentation that in the average case information set pruning reduces the number of leaves that need to be examined to a fraction of the total.

The second contribution mentioned in the introduction is the presentation of the One Player Pruning Algorithm (OPPA). OPPA explores one player games with hidden information using information set pruning, chance node pruning, and immediate pruning to limit the search. In Chapter 3 the One Player Pruning Algorithm was proven to calculate optimal strategies for one player games with perfect recall.

The final contribution is the development of a general model of game trees with hidden information. Chapter 4 presented the hidden information model and its similarities to the game trees of real card games. In addition some advantages of this model were pointed out including its simplicity, the clarification of the concept of learning the hidden information through the ι parameter, and the possible equivalence to games with multiple chance nodes.

These three contributions take the next logical step in the development of search algorithms for hidden information games and raise a number of new questions. Some of these areas for future work include possible different algorithms for one player hidden information games, new pruning techniques, and implementations of the One Player Pruning Algorithm on real games.

BIBLIOGRAPHY

Bibliography

[1] Bruce W. Ballard. The *-minimax search procedure for trees containing chance nodes. *Artificial Intelligence*, 21, 1983.

[2] Avron Barr and Edward Feigenbaum. *The Handbook of Artificial Intelligence*, volume 1. William Kaufmann, Inc., Los Altos, California, 1981.

[3] Jean R. S. Blair, David Mutchler, and Cheng Liu. Heuristic search in one-player games with hidden information. Technical report, University of Tennessee at Knoxville, June 1992.

[4] Donald E. Knuth. *Fundamental Algorithms: The Art of Computer Programming*, volume 1. Addison-Wesley, Reading, Mass., 1969.

[5] Donald E. Knuth and Ronald W. Moore. An analysis of alpha-beta pruning. *Artificial Intelligence*, 6, 1975.

[6] Richard E. Korf. Multi-player alpha-beta pruning. *Artificial Intelligence*, 48, 1991. Research Note.

[7] R. Duncan Luce and Howard Raiffa. *Games and Decisions, Introductions and Critical Survey.* Dover Publications, New York, 1985.

[8] Elaine Rich and Kevin Knight. *Artificial Intelligence.* McGraw Hill, New York, 1991.

[9] G.C. Stockman. A minimax algorithm better than alpha-beta? *Artificial Intelligence*, 12, 1979.

APPENDICES

Appendix A

Lisp Implementation of OPPA

```
(defun OPPA (X prune)
  (if (partial-info-set X)
      (max-set X prune)
      (mixed-set X prune)))

(defun max-set (X prune)
  (let (best moves child temp)
    (setq best prune)
    (setq moves (plausible-moves X))
    (dolist (M moves)
            (setq child (move M X))
            (setq temp (OPPA (S child) best))
            (unmove M X)
            (if (= temp (* U (prob-sum (S child))))
                (let ()
```

```
                (setq best (* U (prob-sum (S child))))
                (return best)))
          (if (> temp best)
              (setq best temp)))
    best))

(defun mixed-set (X prune)
  (let (sum prob-left curr-prob)
    (setq sum 0)
    (setq prob-left (prob-sum X))
    (setq parts (partition X))
    (dolist (elt parts)
            (cond ((atom elt)
                   (setq sum (+ sum (* (p elt) (h elt))))
                   (setq prob-left (- prob-left (p elt))))
                  (t
                   (setq curr-prob (prob-sum elt))
                   (setq prob-left (- prob-left curr-prob))
                   (setq sum (+ sum (OPPA elt
                                          (max (- prune
                                                  (+ sum (* U prob-left)))
                                               (* curr-prob L)))))))
            (if (<= (+ sum (* U prob-left)) prune)
                (return prune)))
     sum))
```

Appendix B

Complete Results for Average Case Trials

Leaves examined for the runs when $\iota = 2$ and $b = 2$

Depth	Run 1	Run 2	Run 3	Run 4	Run 5	Average	Total
2	4	4	3	4	4	3.8	4
3	15	13	16	15	14	14.6	16
4	54	57	48	39	54	50.4	64
5	177	197	186	178	206	188.8	256
6	789	819	773	776	738	779.0	1024
7	3022	2817	3112	3089	2907	2989.4	4096
8	11261	12324	12246	11856	11440	11825.4	16384
9	47083	46045	46963	46035	45794	46384.0	65536

Leaves examined for the runs when $\iota = 2$ and $b = 10$

Depth	Run 1	Run 2	Run 3	Run 4	Run 5	Average	Total
2	16	15	18	14	11	14.8	20
3	170	190	230	176	152	183.6	400
4	2077	2214	1823	2206	1709	2005.8	8000
5	23507	32368	37773	39422	35443	33702.6	160000
6	389980	291490	231700	470666	442527	365272.6	3200000
7	6518133	5166001	5329943	4767073	5723599	5500949.8	64000000

Leaves examined for the runs when $\iota = 10$ and $b = 2$

Depth	Run 1	Run 2	Run 3	Run 4	Run 5	Average	Total
2	20	18	19	20	20	19.4	20
3	378	382	382	374	371	377.4	400
4	7497	7543	7516	7344	7496	7479.2	8000
5	150267	149032	150004	150607	149782	149938.4	160000

Leaves examined for the runs when $\iota = 10$ and $b = 10$

Depth	Run 1	Run 2	Run 3	Run 4	Run 5	Average	Total
2	86	91	89	87	88	88.2	100
3	7865	7289	7806	7821	7777	7711.6	10000
4	768247	772539	769560	759213	775164	768944.6	1000000
5	76731628					76731628.0	100000000

Vita

Michael Christopher van Lent sprung forth upon the world on the thirty first day of May 1968 in the fair town of Palo Alto California. A short three years later he and his parents Marjorie and Peter van Lent traded the gentle weather of California for the harsh winters of Canton New York. While in Canton, Michael grew to approximately his present size and in the process graduated from Hugh C. Williams High School.

The following four years were winnowed away attending Williams College in the aptly named Williamstown Massachusetts. Upon receiving a Bachelor of Arts degree with honors in Computer Science, Michael left the eternal winters of the northeast to join the computer science graduate program at the University of Tennessee.

With the completion of this thesis and the Master of Science degree that it entitles him to Michael is planning to attend the University of Michigan in hot pursuit of a Ph.D.