A THEORETIC APPROACH FOR BINARY GAME TREE EVALUATION

by

BONING ZHAO

Submitted in partial fulfillment of the requirements

For the degree of Master of Science

Thesis Adviser: Vincenzo Liberatore

Electrical Engineer and Computer Science

CASE WESTERN RESERVE UNIVERSITY

Dec, 2019

A THEORETIC APPROACH FOR BINARY GAME TREE EVALUATION

Case Western Reserve University
Case School of Graduate Studies

We hereby approve the thesis[1] of

BONING ZHAO

for the degree of

Master of Science

Dr. Vincenzo Liberatore

Date: December 2, 2019

[1]We certify that written approval has been obtained for any proprietary material contained therein.

*Dedicated to *your*
*dedication message goes here**

Table of Contents

List of Figures

Acknowledgements

Above all I would like to thank my advisor, Professor Vincenzo Liberatre, for his assistance through the entire process of creating this thesis topic and finishing this research. At the beginning of this thesis I knew very little of the process of research, but Professor Liberatre provided advice and suggestion which helped me find a way to begin the experiment. Whenever I met problems, he guided me to out of stuck with patience. It is impossible to complete this thesis without him.

I also would like to express my thanks to my parents. During my first 24 years of life, they tried their best to support my life and study, and I even didn't acknowledge this for a long time. Their love and support is the reason I could be here.

Abstract

A THEORETIC APPROACH FOR BINARY GAME TREE EVALUATION

Abstract

by

BONING ZHAO

The Binary Decision tree model is perhaps the simplest model that computes Boolean functions: it charges only for reading an input variable. We proposed a theoretical binary game tree evaluation analysis process which could provide better lower bound compared with lower bound provided by previous analysis. Based on the concept of reluctant input, We study the power of randomness in this model with both directional and non-directional algorithm, and made the comparison of their efficiency with lower bound. These results are obtained via general and efficient methods for computing lower bounds on the probabilistic complexity and based on a special type of uniform binary decision tree, which we call it NOR-tree, which is an equivalent transformation of AND/OR tree. As a result of study, we finally build up a more accurate analysis process. In order to test its versatility, we also apply this analysis process to two different types of binary game tree: Fibonacci tree and skew-F tree. Both of these two have more complicated structure for game tree evaluation. The results shows that this process could also be used for getting other type of game trees lower bound.

1 Introduction

Game tree is defined as a rooted tree whose internal nodes at even distance from the root are labeled MIN and those at odd distance are labeled MAX.[1] Each leaf of a game tree is real number which called value. We evaluate the game tree by the following step: Every leaf return its value; Then every MIN node returns the smallest value by its children and the MAX node returns the largest. The aim of the evaluation process is to determine the root's value.

The evaluation of game trees is a central part in artificial intelligence and game theory,especially in game-playing programs. The children of a node means the options available to one or two players in a game. The leaves of a game tree represent the value of the game for one or two players. One player seeks to maximize this value, and another will try to minimize it.[2]For evaluation process, the algorithm will choose a leaf and reads its value. The number of this steps taken by an algorithm is the only issue we focus on, which means no other computation will be charged for the algorithm.

This thesis will be limited in binary tree, whose leaves value is 0 or 1. Therefore, we can transform each MAX node to Boolean OR operation and MIN node to Boolean AND operation, which leads the analysis on AND-OR tree.Furthermore, We investigate a simple model called Nor-tree for computation of Boolean functions. It is a binary

boolean decision tree which is a transformation is equivalent AND/OR tree with even height,which will be introduced in following section.

For instance, a uniform tree whose node has n children and all leaves are at distance k from the root. The evaluation of this game tree is to study the maximum number of steps to evaluate all instances of the game tree in same height and with same leaves. this problem consists of the trees structure and Boolean value of n^k leaves.

There are two ways to evaluate game tree: deterministic algorithm and randomized algorithm. For deterministic algorithm, the choice of the next step is a deterministic function of the values at the leaves read so far. For any deterministic function, there exists an instance that needs to read all the leaves. Therefore, the evaluating result for a deterministic algorithm is always n^k.[3]

A randomized algorithm is an algorithm which contains randomness as part of its logic. For game tree evaluation, a randomized algorithm means the choice of the next leaf to be read may be randomized. Using randomized algorithm is to achieve good performance in the "average" cases over all possible instances.[4] We will focus on randomized algorithm in this thesis, because randomized algorithm could benefit game tree evaluation compared with deterministic algorithm. A simple example, For a binary tree with height k. Any deterministic algorithm will have to evaluate all 2^k leaves.

A simple randomized algorithm which will take the probability $p < 1$ to choose which step to read. For AND node return 0, if a deterministic algorithm reads the 0 at second step, a randomized algorithm will have probability p to read it in the first step, then we don't need to visit the other one. Its expected number of step is $2-p$ which is less than 2. Similarly, for OR node, if it returns 1, the expected number for randomized algorithm is

$2-p$. This adversary could be used for proving the randomized algorithm works better than deterministic algorithm in evaluating game tree, which we will show in section 2.

In this thesis, we proposed a game tree evaluation process compared with the one in "Randomized Algorithm", in which the proof has some loose and the lower bound is not tight.[5] The game tree evaluation process is based on the game theory and Yao's Principle[6]. This process provide better lower bound analysis and could be expanded to evaluating not only uniform binary tree but also other types binary game tree.

In chapter 2, we provide background information of game tree, minimax algorithm and randomized algorithm, which are important for understanding the following analysis. We also introduce basic evaluation steps for nor-tree, a binary tree which equivalent to AND-OR tree. In chapter 3, we discuss the analysis of directional algorithm's and its lower bound on Nor-tree. In chapter 4, we present the lower bound proof for non-directional algorithm, which includes three different cases. We also make a comparison between directional algorithm and non-directional in order to find which is better. In chapter 5, we will introduce two different binary game tree, skew-F tree and Fibonacci tree, and show whether this game tree evaluation process could also be used for evaluating them. This step is to test versatility of this analysis process. In chapter 6, we conclude by describing the whole process and future work using this method.

2 Background and Definition

In this chapter, we will introduce some background of the algorithm and theory, which we used in the following research and proof. This part includes: benefit of randomized algorithm in game tree evaluation, minimax principle, Nor-tree definition and proof of its equivalence with AND-OR tree, Nor-tree evaluation process, reluctant input, which are all important in our research.

2.1 Background

2.1.1 Benefit of Randomized Algorithm

In previous chapter, we showed that randomized algorithm will have better expected number in evaluating compared with deterministic algorithm in some situation. We will prove this by induction. For an AND-OR tree T with height $2k$(denoted as T_{2k}), the expected number for deterministic algorithm is 4^k, now we argue that for the randomized algorithm we mentioned before the expected cost is at most 3^k.

The basis condition when $k = 1$ just follows the analysis we made in chapter 1. Then we assume that for T_{2k-1} the expected number needs to be read is 3^{k-1}. Then by induction, we firstly analyze a tree with an AND node as root. If the root evaluates to 1, then both of its children at OR node return 1. The expected cost of this situation is $2 \times 3^{k-1}$,

because both of the children need to be evaluated. On opposite, if the root evaluates to 0, the with probability $\frac{1}{2}$ the 0 node will be choose first, then the expected cost is $\frac{1}{2} \times 3^{k-1}$. Other side, with $\frac{1}{2}$ probability the 0 one will be visit in second step, then $2 \times \frac{1}{2} \times 3^{k-1}$. Then putting these two result together, we could determine the expected cost of this AND-OR tree is

$$\frac{1}{2} \times 3^{k-1} + 2 \times \frac{1}{2} \times 3^{k-1} = \frac{3^k}{2} \tag{2.1}$$

Similarly, we could also get root node is an OR node has the same expected cost. Therefore we could see that randomized algorithm is better than deterministic algorithm in game tree evaluating.

2.1.2 Minimax Principle

Minimax Algorithm is widely used in decision making and game theory in order to find the optimal move for a specific player. The minimax principle is the only known general technique for proving lower bounds of randomized algorithms. It only could be applied to algorithms those could finish in finite time. In minimax algorithm, we assuming that both the player and his opponent play optimally. There is an example for illustrating the basic method of it:

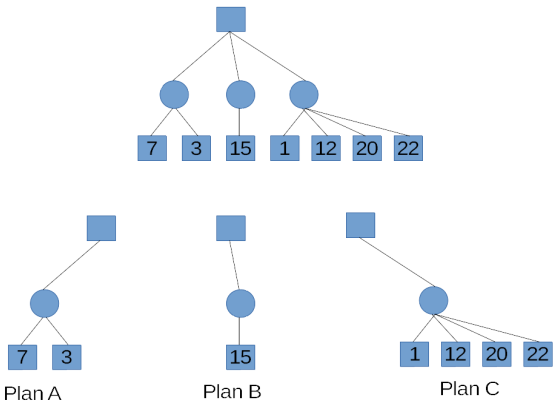


Figure 2.1. example of minimax principle

Let square be player A ,and circle representing player B. Each leaf represents a possible benefit for player A. The aim of this game is for B to minimize A's benefit and A will try to maximize his benefit. If player A choose Plan A, then player B has two choices, in order to minimize player A's benefit. B will choose 3. For Plan B, player B has no choices but to let player A get 15. At last if plan C is chosen, then player A could only get 1 benefit. Therefore in order to maximize his benefit, player A will choose plan B.

In order to understand the minimax principle theoretically. We will introduce some basic concept in game theory. One important concept is two-person zero-sum game, in which minimax principle could be applied. Let's say there are two player T and F. They will choose from three tarot card: Fool, Justice, Judgement. They don't know what card the opposite choose. Then they will show the card at the same time. The winner is determined by the following rules: The Fool beats Justice and wins 5 dollar; Justice wins the Judgement and Judgement will beat the Fool. Both of this two cases wins 3 dollar.If they choose the same card, then it comes to a draw. The game described above os an instance of a two-person zero-sum game. A two-person zero-sum game always could be represented as a payoff matrix. Like the game above could be:

Table 2.1. Matrix for tarot game

T \ F	Fool	Justice	Judgement
Fool	0	5	-3
Justice	-5	0	3
Judgement	3	-3	0

Let's set the matrix as M. Every row could denote the strategy chosen by player T and every column could be the strategy for player F.Then the entry M_{ij} is the amount paid

from T to F when T chooses strategy i and F chooses j. Therefore, F will try to minimize the payoff and T will try to maximize the payoff. Because both player T and F has no idea about what strategy their opponent will choose. If T choose strategy i, then the payoff will be $min_j M_{ij}$ and no matter what F will choose the payoff is guaranteed. We could see the optimal strategy for player T is to maximize $min_j M_{ij}$.Similarly, We could get the player F's optimal payoff is $min_j max_i M_{ij}$. For all payoff matrices we have:

Lemma 1. $max_i min_j M_{ij} \leq min_j max_i M_{ij}$

When the equality exists, which mean the game is said has a solution and the value of the game is the equal to the lower bound of payoff from player trying to maximize it or the upper bound of payoff from player trying to minimize it. The solution also could be called as saddle point[7]. However, not all games have a solution under this type strategy. Like in tarot game, Value for T is 3 and F is -3. Both of the player will try to improve their payoff as much as possible. So far we discussed is about pure or deterministic strategy. In order to solve the game has no solutions, we should use strategies with randomization in it, which are called mixed or randomized strategies. A randomized strategy is a probability distribution on the set of all possible strategies. For example, in tarot game(Let's say payoff matrix M has m rows and n columns), player T will have probability p_i to choose row i as his strategy and the randomized strategy for T is a vector $\mathbf{p} = (p_1, ..., p_n)$. Similarly, let's define randomized strategy for F as $\mathbf{q} = (q_1, ..., q_m)$, which denotes the probability distribution on columns. Therefore we could get the expectation payoff as:

$$E[pay] = p^T Mq = \sum_{i=1}^{n} \sum_{j=1}^{m} p_i M_i j q_j$$

Similarly to how we analyze deterministic strategy, we could get The optimal lower bound for player T is $L = max_p min_q p^T Mq$ and optimal upper bound for player F is $U =$

$min_q max_p p^T Mq$. Then here comes a well-known Minimax Theorem of von Neumann[8].For any two-person zero-sum game specified by a payoff matrix M:

Theorem 1. $max_{\mathbf{p}} min_{\mathbf{q}} \mathbf{p}^T M\mathbf{q} = min_{\mathbf{q}} max_{\mathbf{p}} \mathbf{p}^T M\mathbf{q}$

which means that this type of game always has a solution and a pair of randomized strategies (**p**,**q**), which helps reach the solution is the saddle point of the game. We also could easily find that $\mathbf{p}^T M\mathbf{q}$ is a linear function of **q** and it could be simplified by setting 1 to the smallest coefficient q_j and 0 to others. We could find that if player F knows the probability distribution **p**, then his optimal solution will be a deterministic strategy. The theorem 1 also could be lead to a simplified version which invented by Loomis[9]:

Theorem 2. $max_{\mathbf{p}} min_j \mathbf{p}^T Mu_j = min_{\mathbf{q}} max_i e_i^T M\mathbf{q}$ *(where u_i means a unit vector with a 1 in i position and 0s in others)*

2.1.3 Yao's Principle

Yao's principle, which also called 'minimax principle' claims that the expected cost of randomized algorithm on the worst-case input is no better than the expected cost for a worst-case probability distribution on the inputs of the deterministic algorithm that performs best against that distribution.[6] Let's use the tarot game in previous game to illustrate this. We set player F as the one who apply the algorithm(every column), and player T as adversary choose the input(every row).Let's say the payoff from player F to player T is the valuable cost of the performance of an algorithm with rows as a finite set of all possible inputs and columns as a finite set of possible deterministic algorithms that always produces a correct solution. Clearly, player F will choose an algorithm which minimizes the payoff, and the player T will try to maximize it.

In this problem, we could observe that the number of distinct inputs is equal to the number of distinct deterministic algorithms which could promise finish with correct

solution. A deterministic strategy for player T is an input and for player F is a deterministic algorithm. A randomized strategy for T is a distribution over all inputs and a randomized strategy for F is a distribution over the space of deterministic algorithms[10]. We could see this is a Las Vegas randomized algorithm[11]. Let's set L_F is the worst case algorithm cost for all deterministic algorithm and U_T is the best case algorithm cost for the worst deterministic algorithm choice. We denote L_F as the deterministic complexity for the game and U_T as the randomized complexity[12]. In this thesis, we will focus on complexity which is the expected algorithm cost of the best deterministic algorithm with worst distribution on input[13]. This complexity is smaller than deterministic one because the algorithm knows the distribution.[14]

From Loomis Minimax Principle, we could get the complexity equals the least possible costs with any randomized algorithm. Now we could apply the Minimax Principle as: Let G be a two-person zero-sum game problem. It has a finite set **I** as input and a finite set **D** as the set of all possible, always correct deterministic algorithms. For instance I ∈ **I** and D ∈ **D**, we denote C(I,D) as the cost for deterministic algorithm over input I. The probability distributions on **I** and **D** are **p** and **q**. Then we could get theorem 1 and 2 as :

$$max_{\mathbf{p}} min_{\mathbf{q}} E[C(\mathbf{I_p}, \mathbf{D_q})] = min_{\mathbf{q}} max_{\mathbf{p}} E[C(\mathbf{I_p}, \mathbf{D_q})] \tag{2.2}$$

$$max_{\mathbf{p}} min_{D \in \mathbf{D}} E[C(\mathbf{I_p}, \mathbf{D})] = min_{\mathbf{q}} max_{I \in \mathbf{I}} E[C(I, \mathbf{D_q})] \tag{2.3}$$

Then the Yao's Minimax Principle comes:

Theorem 3.

$$min_{D \in \mathbf{D}} E[C(\mathbf{i_p}, D)] \leq max_{I \in \mathbf{I}} E[C(I, \mathbf{D_q})]$$

Yao's Principle is a very useful tool for proving the lower bound for randomized algorithm. It claims the expected cost of the optimal deterministic algorithm for a randomized chosen input distribution is the lower bound for the cost of optimal randomized algorithm for problem. The reduction to a lower bound on deterministic algorithms makes Yao's principle is very powerful in proving lower bound.[15]

2.1.4 Nor-Tree Evaluation

In previous chapter. We introduce the binary tree evaluation could be transformed to an AND-OR tree evaluation by limited the leaves value with in 0 and 1. In this thesis, we will use a more simplified tree model which called nor-tree, its logic is following:

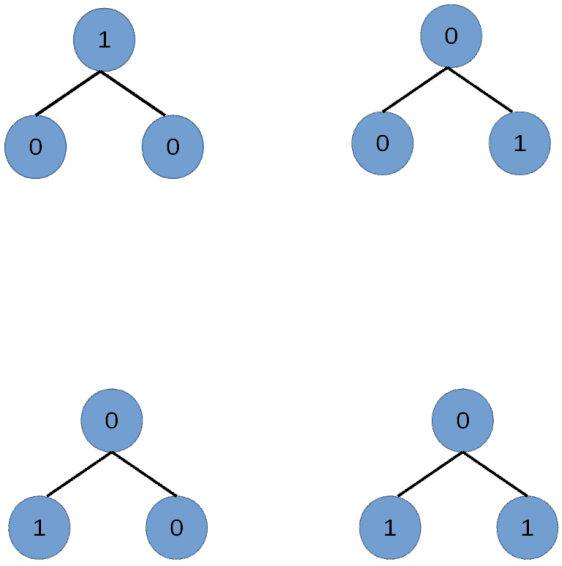
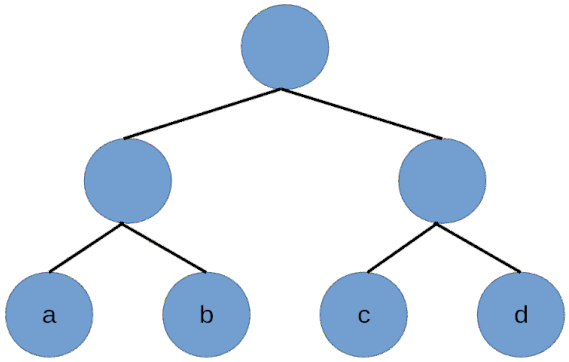


Figure 2.2. Nortree logic

A balanced binary tree whose all leaves are at the distance 2k from root is equal to the same structure nor-tree. Firstly, we already proved that this is equivalent to the same height AND-OR tree. Then Let's see when k = 1. Let's denote the 4 notes as a,b,c,d. The value of the root of AND-OR tree is $(a \bigvee b) \bigcap (c \bigvee d)$ and the nor-tree's is $\neg(a \bigvee b) \bigcap \neg(c \bigvee d)$.

We could see this two is equal. Assuming that k=n-1 this claim is correct. Similarly, we could prove when k=n, this is also correct.



(a ∨ b)∩ (c ∨ d)=¬(a ∨ b)∩¬(c ∨ d)

Figure 2.3. AND/OR tree equivalent to Nor-tree with Same Height

Then for the evaluation of this nor-tree we have:

$$h(x) = \begin{cases} h_{left}(x_L) + h_{right}(x_R) & if\ algorithm = L\ and\ left\ equals\ to\ 0 \\ h_{left}(x_L) & if\ algorithm = L\ and\ left\ equals\ to\ 1 \\ h_{left}(x_L) + h_{right}(x_R) & if\ algorithm = R\ and\ right\ equals\ to\ 0 \\ h_{right}(x_R) & if\ algorithm = R\ and\ right\ equals\ to\ 1 \end{cases}$$

which could be described as:

evaluation of nor-tree

```
1:NorEvaluation(node)
2:if (node.type = "LEAF") then
3:    return(node.value)
4:end if
```

```
5:if (node.type = "Nor") then
6:  Toss a coin
7:  if (Heads) then
8:  p1 = NorEvaluation(node.left)
9:  if(p1 = 1) then
10:   return 0
11:  else
12:  p2= NorEvaluation(node.right)
13:    if(p2 = 0)
14:      return 1
15:    else
16:      return 0;
17:    end if
18:  end if
19:end if
```

Yao's principle has its flexibility in the choice of input distribution. In "Randomized Algorithm", the author just choose the distribution by arbitrary and ignored conditional probability in internal nodes. By experiments simulates the leaves value distribution with algorithm we find that the algorithm does not effect the best payoff of the tree, and we could find the pattern of the optimal nor-tree is:

(1) The value of the root is always 1.

(2) When the value of node equals to 0, the value of two leaves will has 1/2 probability to be (0,1) and 1/2 to be (1,0);

In this thesis, we will use L(k,i) to denote the min expected leaves visited by algorithm of node in k level whose value is i(payoff for a tree of height k, its value is i, and k is even).

2.2 Definitions

We define the value of a leaf as its input value (leaf label), and the value of an internal node as true if both children are false, and false otherwise. We define an algorithm for a NOR tree T as a process to find a specific ordering σ of T's leaves. It always starts with a leaves ordering. Based on the value of the vertices's already been determined, it will rearrange the ordering in order to determine the root's value by visiting as less leaves as possible.

The prefix of σ of length t will be denoted as σ_t. Given a leaf labeling and an ordering σ, we associate a logical time-stamp to each tree vertex as follows. We will say that a leaf u is evaluated at time t if σ_t is the shortest prefix of σ containing u. It is immediate to see that u is evaluated at time t if and only if the tth element of σ is u. If u is an internal node whose value is false, we will say that u is evaluated at time t if σ_t is the shortest prefix such that both children of u have been evaluated at time t or earlier. Conversely, if u is an internal node whose value is true, u is evaluated at time t if σ_t is the shortest prefix such that at least one child of u whose value is false has been evaluated at time t or earlier. Intuitively, the definition captures the idea that the value of an internal node becomes available as soon as there is the minimum amount of information needed to evaluate it. We define the cost of a NOR tree algorithm as the time at which the root is evaluated. Note that once the root is evaluated, the remaining suffix of σ is irrelevant to

σ's cost. In light of this observation, we will say that σ omits leaf u if leaf u appears after the root has been evaluated. Similarly, we say that σ' omits an internal node x if node x appears after the root has been determined.

Note that any tree traversal implicitly defines a leaf ordering σ as the subsequence of the traversal order containing only the leaves. Conversely, given a leaf ordering σ', we define its canonical traversal as the vertex ordering whose leaf subsequence is σ', where a true internal node u appears at the earliest point after both of its children, and where a false internal node u appears at the earliest point after one of its true children. It is immediate to see that the evaluation order of σ can be reconstructed from its canonical traversal.

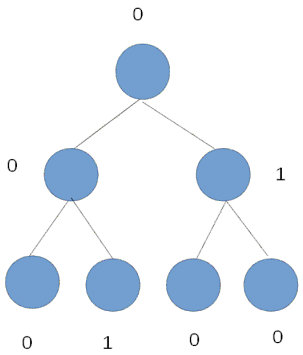


Figure 2.4. A NOR-tree of height 2 with a binary labeling of the leaves and calculated values in the internal nodes.

Figure 2.4 shows an example of these definitions. Suppose that an algorithm visits the four leaves from left to right. Then, the corresponding leaf ordering is $\sigma' = (a, b, c, d)$. The canonical ordering σ can be constructed as follows. First, since σ' is a subsequence of σ, then the leaves appear in the order a, b, c, d within σ Since e is a false internal node, node e appears in σ' immediately after its true child a. Since f is a true internal node, node f appears in σ' immediately after c and d. Finally, since g is false, node

g appears in σ' immediately after f. By putting these constraints together, we obtain that $\sigma' = (a, e, b, c, d, f, g)$. It is immediate to see that a better algorithm would visit only three leaves if $\sigma_3 = (a, c, d, b)$ and $\sigma'_3 = (a, e, c, d, f, g, b)$. and that the optimal visit enables the tree evaluation by visiting only the leaves c and d with $\sigma_2 = (c, d, a, b)$ and $\sigma'_2 = (c, d, f, g, a, e, b)$. Note that the algorithm σ_2 omits the leaves a and b.

3 Analysis of Directional Algorithm

In this chapter, we will analyze the process of deterministic directional algorithm for nor-tree evaluation. With computing the lower bound of this algorithm, we will compare its difference with general evaluation process.

3.1 Directional Algorithm

We begin by analyzing a class of directional algorithms' lower bounds, as per the following definition.

Definition 1. *A deterministic directional algorithm for a NOR tree T is the leaf sequence in the order visited by a depth-first traversal of T except that it omits the visit of a child of a vertex u if u's value has already been determined. A randomized directional algorithm is a probability distribution over deterministic directional algorithms.*

We claim that any randomized directional algorithm must visit $L(k,1)$ leaves, where $L(k,1)$ will be defined shortly. The proof relies on Yao's principle, whereby it is sufficient to show a probability distribution on the leaf labels that forces any deterministic directional algorithm to visit $L(k,1)$ leaves in the expectation. The probability distribution is built recursively for a tree of height k from the probability distribution of trees of height $k-1$. The distribution assigns labels in such a way that a tree of height k always evaluates to true. Therefore, both root sub-trees of height $k-1$ evaluate to false. In turn, if a sub-tree must evaluate to false, then leaf values will be assigned in such a way that

one of the sub-trees evaluates to true and the other to false, and the two alternatives are chosen with probability 1/2. Such a probability distribution on leaf labels will be termed a reluctant input.

Let $L(k,i)$ $(k = 0,1,2,\ldots;\ i = 0,1)$ be the expected number of leaves visited by a deterministic depth-first pruned algorithm in correspondence to a reluctant input for a tree of height k which evaluates to true or false if $i = 1$ or 0 respectively.

Then the recurrence relationship is:

$$L(k,1) = 2L(k-1,0) \tag{3.1}$$

$$L(k,0) = L(k-1,1) + \frac{1}{2}L(k-1,0) \tag{3.2}$$

$$L(0,i) = 1 \quad (i = 0,1) \tag{3.3}$$

3.2 Lower Bound Analysis

In this section, we will show that $L(k,0) \geq \rho_{k+1} a^{k+1}/2$, where $a = (1+\sqrt{33})/4 \approx 1.6861$, and $\lim_{k\to\infty} \rho_k \approx 1.10927$. Then, by (3.1), we have that

$$L(k,1) = 2L(k-1,0) \geq \rho_{k+1} a^k \tag{3.4}$$

First of all, we could get the initial condition by calculation:

$$L(1,0) = 1 \tag{3.5}$$

$$L(1,1) = 2 \tag{3.6}$$

$$L(2,1) = 3 \tag{3.7}$$

Define the sequence ρ_k as the solution to the recurrence in previous section for $L(k,0)$, with the initial condition above and (3.1):

$$\rho_k = \frac{\rho_{k-1}}{2a} + 2\frac{\rho_{k-2}}{a^2} \tag{3.8}$$

$$\rho_2 = \frac{3}{a^2} \tag{3.9}$$

$$\rho_1 = \frac{2}{a} \tag{3.10}$$

Then we claim that:

Lemma 2. *We have that*

$$\lim_{k\to\infty} \rho_k = \rho = \frac{20\sqrt{2}+4\sqrt{66}}{\sqrt{66}+33\sqrt{2}} \approx 1.10927\,.$$

PROOF. Define

$$\mathbf{r}_{k-1} = \begin{pmatrix} \rho_k \\ \rho_{k-1} \end{pmatrix} \qquad (k = 2,\ldots)$$

In extended state representation, recurrence (3.8)-(3.10) becomes $\mathbf{r}_k = A\mathbf{r}_{k-1}$, where

$$A = \begin{pmatrix} \frac{1}{2a} & \frac{2}{a^2} \\ 1 & 0 \end{pmatrix}.$$

It is immediate to see that the eigenvalues of A are $\lambda_1 = 1$ and $\lambda_2 = (1-\sqrt{33})/(1+\sqrt{33})$, and that $-1 < \lambda_2 < 0$. Therefore, $\mathbf{r}_k$ tends to the projection of $\mathbf{r}_1$ along an eigenvector $\mathbf{v}_1$ corresponding to λ_1. Furthermore, it can be verified that a unit eigenvector corresponding to λ_1 (λ_2) is:

$$\mathbf{v}_1 = \frac{\sqrt{2}}{2}\begin{pmatrix} 1 \\ 1 \end{pmatrix} \qquad \left(\mathbf{v}_2 = \frac{1}{\sqrt{\lambda_2^2+1}}\begin{pmatrix} \lambda_2 \\ 1 \end{pmatrix}\right)$$

The component r of $\mathbf{r}_1$ along $\mathbf{v}_1$ is given by solving

$$\begin{pmatrix} \mathbf{v}_1 & \mathbf{v}_2 \end{pmatrix}\begin{pmatrix} r \\ s \end{pmatrix} = \begin{pmatrix} \rho_2 \\ \rho_1 \end{pmatrix}$$

and is equal to $r = (40+8\sqrt{33})/(\sqrt{66}+33\sqrt{2})$. Since $\lim_{k\to\infty}\mathbf{r}_k = r\mathbf{v}_1$, by definition of $\mathbf{r}_k$ and $\mathbf{v}_1$ we have that

$$\lim_{k\to\infty} \rho_k = \frac{\sqrt{2}}{2}r = \frac{20\sqrt{2}+4\sqrt{66}}{\sqrt{66}+33\sqrt{2}}\,,$$

which proves the claim. □

As we have ρ_k as constant when k goes to infinity. We could claim:

Lemma 3. *$L(k,0)$ is bounded by $L(k,0) = \rho_{k+1}a^{k+1}/2$.*

PROOF. Define for convenience of notation $T(k) = L(k,0)$. By (3.3) and (3.10), we have that $T(0) = \rho_1 a/2$. Furthermore, by (3.2), (3.3), and (3.9), we have that $T(1) = 3/2 = \rho_2 a^2/2$. Substitute (3.1) in (3.2) to obtain: $T(k) = 2T(k-2) + T(k-1)/2$. By induction hypothesis and definition (3.8), $T(k) = \rho_{k-1}a^{k-1} + \rho_k a^k/4 = a^{k-1}\left(\rho_{k-1} + \rho_k a/4\right) = \rho_{k+1}a^{k+1}/2$, which completes the proof. □

Therefore, by combining $lemma2$ and (3.1), we could obtain the following tighter lower bound:

Theorem 4. *We have that $\lim_{k\to\infty} L(k,0)/a^{k+1} = \rho/2$ and $\lim_{k\to\infty} L(k,1)/a^k = \rho$.*

3.3 Comparison

The difference between the lower bound discussed in this thesis and the one in 'Randomized Algorithms'[?] is shown in the table below:

Comparison		
Item	This article	'Randomized Algorithms'
root	r=1	r= 0 with probability 1-p or 1 with probability p
Lower Bound	$L(k,1) \geq \rho_{k+1}\alpha^k$	$pL(k,1) + (1-p)L(k,0) \geq p\rho_{k+1}\alpha^k + (1-p)\rho_{k+1}(L_0)\frac{\alpha^{k+1}}{2}$

The Lower bound in 'Randomized Algorithms' equals to:

$$\rho_{k+1}\alpha^k[p + (1-p)\frac{\alpha}{2}] = L_0[p + (1-p)\frac{\alpha}{2}] \geq L(k,1)$$

with $\alpha = \frac{1+\sqrt{33}}{4}, p = \frac{3-\sqrt{5}}{2}$

With Calculation(Appendix: Detailed Calculation), we could see that the lower bound in this article is better than the previous one.

4 Analysis of Non-Directional Algorithm

As our lower bound is based on directional algorithms, in this section, we will discuss whether there exists a non-directional algorithm which is faster than the lower bound. Specifically, we claim that for the probability distribution defined in the previous section, there is a deterministic directional algorithm whose expected cost is lower than or equal to the expected cost of any deterministic non-directional algorithm.

4.1 Definition

Let σ be a non-directional leaf ordering, σ' be the corresponding canonical ordering, and σ'_h be the subsequence of σ containing only vertices at height h or lower. Let v be the first non-omitted vertex at which σ' differs from any directional algorithm. Let h be v's height and let t be v's logical time-stamp. Note that $t \geq 1$ since there is always a depth-first traversal that starts from any leaf. Let u be the last vertex that precedes v in σ', and let w be u's parent.

4.2 Analysis of three different cases of Non-Directional Algorithm

The proof proceeds by analysing the three cases in which a non-directional algorithm can fall into.

4.2.1 Non-Directional Algorithm Case 1

In this case, u and v are siblings and u is the last vertex that precedes v in σ'_h. Vertex w's value could be determined by vising u, non-directional algorithm 1 will go straight to visit v but not omitting it.:

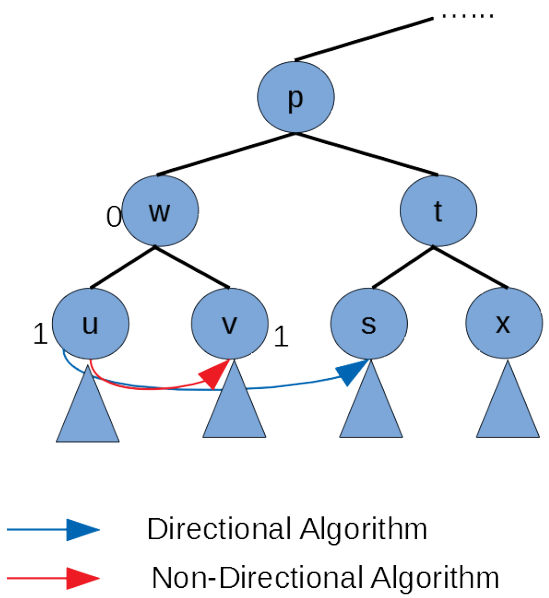


Figure 4.1. Non-Directional Algorithm case 1

PROOF. For case 1.If u and v are siblings, then the non-deterministic algorithm omits v because it can determine the value of w by visiting .Vertex w's value could be

determined only when u is true and w must be $false$. According to section 4, we already know that this means v's value could also been determined as false. Therefore, for any non-directional algorithm who evaluate v, the time for this process is wasted.

In conclusion, this case non-directional algorithm could not be faster than directional algorithm □

4.2.2 Non-Directional Algorithm Case 2

In this case. u and v are not siblings, but their parents are siblings. Vertex u could not determine the w's value, but instead of visiting its sibling, non-directional algorithm 2 jumps out to visit v, who is in another sub tree. After visiting vertex v, there are two small cases: either jump back to visit z, or continue by visiting y. The non-directional algorithm 2 follows the figure below:

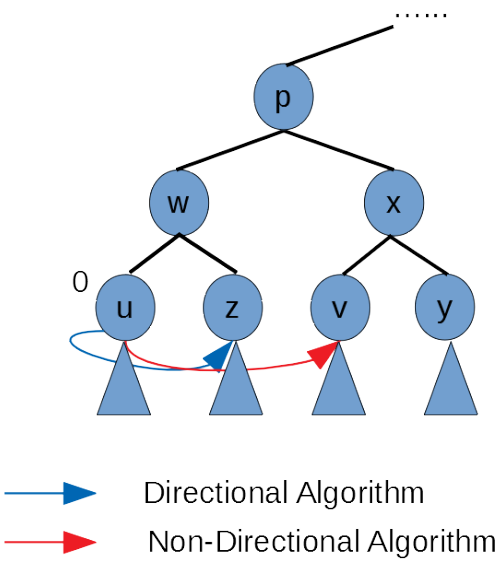


Figure 4.2. Non-Directional Algorithm case 2

If u and v are not siblings but at the same level, the non-directional algorithm 2 will visit v when w's value has not been determined. This non-directional algorithm will take advantage when only by visiting v and its sibling could determine p's value and z should be omitted. In this situation, u's value should be 0, and x's value should be 1. Therefore v and its sibling y's value should be 0, and p's value should also be 0.z should be 1, because if z is 0, then p's value could be determined by directional algorithm by visiting z after u, the visit of u in non-directional algorithm is a waste. Let's prove it is also faster for trees of height n by induction.

Assumption: $E(c_N(x))$ depends only on the height of the tree rooted at x and on the value of vertex x, but it does not depend on the state of the algorithm (for example, it is immaterial whether x is the first or second child visited in the tree rooted at x's parent).

E.g. $E(c_N(x)|w=1,x=0)$ actually does not include $E(c_N(u)|w=1,x=0)$, it just includes expectation of node v and node y. This means though at this state of algorithm, of which we must evaluate vertex u first before the evaluation process of vertex x, the expectation of evaluation cost of x is not affected by the expectation of u. It should just determined by its own sub-trees' expectation. Otherwise we could say that, the expectation of a vertex is just determined by its own structure, but not by algorithm.

PROOF. Assuming that directional algorithm is faster than or equal to non-directional algorithm for trees of height 1,2,...,n-1. For nor-tree of height n: Let $E(c(s))$ be the expected cost of an vertex s in directional algorithm and $E(c_N(s))$ be the expected cost of an vertex s in non-directional algorithm. Let P,X,W be situation when $w=0,x=0$, $w=0,x=1$ and $w=1,x=0$ The expected cost of directional algorithm is:

$$\begin{aligned} E(c(p)) = Pr[P]E[c(w)|P] + Pr[P]E[c(x)|P] + \\ Pr[X]E[c(w)|X] + Pr[X]E[c(x)|X] + Pr[W]E[c(w)|W] \end{aligned} \tag{4.1}$$

For the non-directional algorithm, there are two sub-cases. If the algorithm continues to visit vertex y before going back to z, then:

$$E(c_N(p)) = Pr[P]E[c_N(w)|P] + Pr[P]E[c_N(x)|P] + Pr[X]E[c_N(u)|X] + Pr[X]E[c_N(x)|X] + Pr[W]E[c_N(w)|W] + Pr[W]E[c_N(x)|W] \tag{4.2}$$

For jumping back to z's case:

$$E(c_N(p)) = Pr[P]E[c_N(w)|P] + Pr[P]E[c_N(x)|P] + Pr[X]E[c_N(w)|X] + Pr[X]E[c_N(x)|X] + Pr[W]E[c_N(w)|W] + Pr[W]E[c_N(v)|W] \tag{4.3}$$

For the algorithm continues to visit vertex $y$ïijŽ

$$E(c_N(p)) - E(c(p)) \geq Pr[W]E[c_N(x)|W] - Pr[X]E[c(w)|X] + Pr[X]E[c_N(u)|X] \tag{4.4}$$

Based on the assumption: $E[c_N(x)|W] = E[c_N(w)|X]$, which means that the expectation of x under W is equal to the expectation of w under X, because the condition and value of these two are the same in these two different situation. For the algorithm jump back to visit z:

$$E(c_N(p)) - E(c(p)) \geq Pr[W]E[c_N(v)|W] \tag{4.5}$$

Thus we could find that directional algorithm is faster; □

4.2.3 Non-Directional Algorithm Case 3

In this case. u and v are not siblings, and they may not in the same height. Their parents are not siblings; Vertex u could not determine the w's value, but instead of visiting its sibling, non-directional algorithm 3 will jump out to visit v, who is in another sub tree and in different height. The non-directional algorithm 3 follows the figure below:

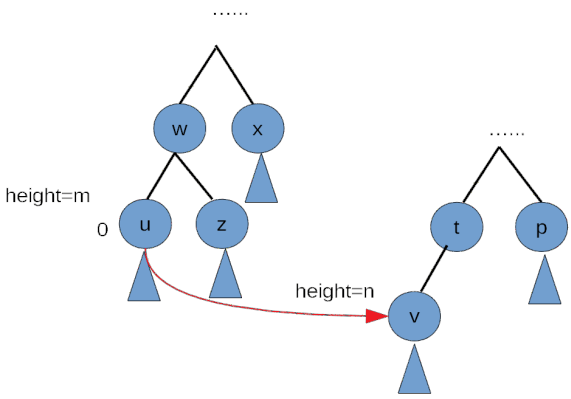


Figure 4.3. Non-Directional Algorithm case 3

In this case, let$p$ be the nearest common parent of u and v and w and x be its child nodes.just as the figure below shows:

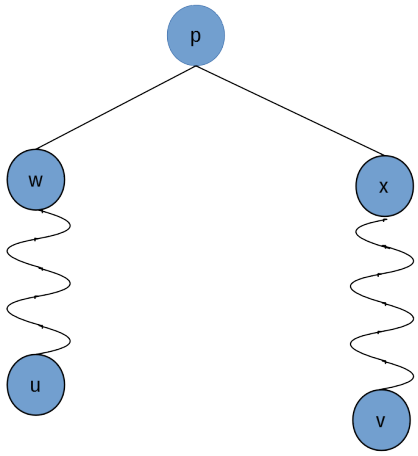


Figure 4.4. Non-Directional Algorithm case 3 transform

Similar to case 2's proof, we will prove this by induction. Assuming that directional algorithm is faster than or equal to non-directional algorithm for trees of height 1,2,...,n-1. For nor-tree of height n, Here are several small cases here:

(1) The non-directional algorithm continue visit the sub tree containing v;

(2) The non-directional algorithm jump back to u to visit that sub tree.

(3) The non-directional algorithm jump to different subtrees randomly.

PROOF. Let P, X, W be situation when $w = 0, x = 0$, $w = 0, x = 1$ and $w = 1, x = 0$.et $E(c(s))$ be the expected cost of an vertex s in directional algorithm and $E(c_N(s))$ be the expected cost of an vertex s in non-directional algorithm.

The expected cost of directional algorithm is:

$$\begin{aligned} E(c(p)) = Pr[P]E[c(w)|P] + Pr[P]E[c(x)|P] + Pr[X]E[c(w)|X] \\ + Pr[X]E[c(x)|X] + Pr[W]E[c(w)|W] \end{aligned} \tag{4.6}$$

For non-directional algorithm continue visit the sub tree containing v:

$$\begin{aligned} E(c_N(p)) = Pr[P]E[c_N(w)|P] + Pr[P]E[c_N(x)|P] + Pr[X]E[c_N(u)|X] \\ +Pr[X]E[c_N(x)|X] + Pr[W]E[c_N(w)|W] + Pr[W]E[c_N(x)|W] \end{aligned} \tag{4.7}$$

For non-directional algorithm jump back to u to visit that sub tree:

$$\begin{aligned} E(c_N(p)) = Pr[P]E[c_N(w)|P] + Pr[P]E[c_N(x)|P] + Pr[X]E[c_N(w)|X] \\ +Pr[X]E[c_N(x)|X] + Pr[W]E[c_N(w)|W] + Pr[W]E[c_N(v)|W] \end{aligned} \tag{4.8}$$

For the algorithm continue visit the sub tree containing v:

$$E(c_N(p)) - E(c(p)) \geq Pr[W]E[c_N(x)|W] - Pr[X]E[c(w)|X] + Pr[X]E[c_N(u)|X] \tag{4.9}$$

Based on the assumption: $E[c_N(x)|W] = E[c_N(w)|X]$, the algorithm jump back to u to visit that sub tree:

$$E(c_N(p)) - E(c(p)) \geq Pr[W]E[c_N(v)|W]$$

Therefore we could conclude that directional algorithm is faster than the non-directional one in this case. □

5 Experiment and Analysis for Algorithm's Versatility

From previous chapters, we have proved the algorithm in uniform binary tree. We wonder whether this evaluation process could still be useful in other types of binary trees. Therefore, in this section, we will analyze the game tree evaluation algorithm through two different types binary game tree, Fibonacci tree and skew-F tree. These experiments could help us to understand the Versatility for this algorithm.

5.1 Fibonacci Tree

5.1.1 Definition

Fibonacci tree[16] is defined as a set of binary trees representing the recursive call structure of the Fibonacci computation process. It has n-1 internal nodes and n leaves. The root's value of Fibonacci tree is equal to the nth Fibonacci number. The left sub-tree should represent the computation process of $n-1$th Fibonacci number, and the right sub-tree of root should represent the computation process of the $n-2$th Fibonacci. An Example of Fibonacci tree with height 4 is shown below

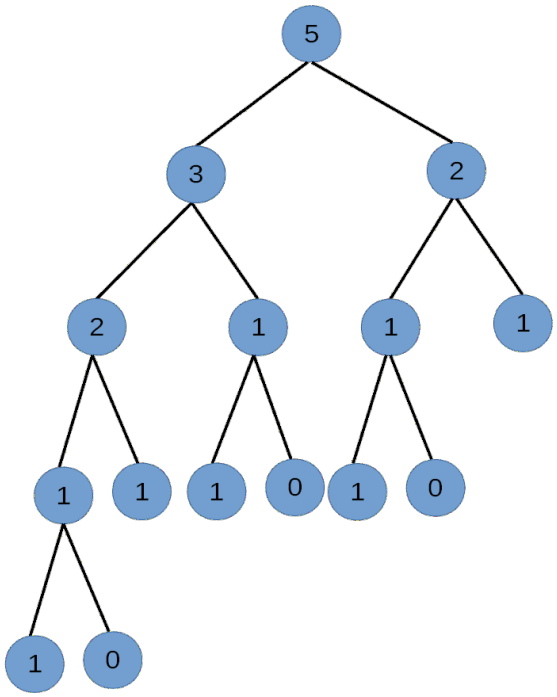


Figure 5.1. Fibonacci Tree

5.1.2 Fibonacci Tree Evaluation

We will define a reluctant input, and let $L(k,i)$ $(k = 0,1,2,\ldots;\ i = 0,1)$ be the expected number of leaves visited by a deterministic depth-first pruned algorithm for a Fibonacci tree of height k which evaluates to true or false if $i = 1$ or 0 respectively. We first have that

$$L(k,1) = L(k-1,0) + L(k-2,0) \tag{5.1}$$

$$L(0,x) = 1(x = 1,0) \tag{5.2}$$

Let p_k be defined recursively by $p_1 = 1/2$ and

$$\frac{p_k}{1-p_k} = \frac{L(k-1,0)}{L(k-2,0)}$$

. The reluctant input is defined as follows. If we wish that the k-th Fibonacci tree ($k > 0$) evaluates to 0, then the input will make sure that at least one of the subtrees evaluates to 0. Specifically, the left subtree evaluates to 0 with probability p_k and to 1 with probability $1-p_k$. Consequently, the right subtree evaluates to 0 with probability $1-p_k$ and to 1 with probability p_k.

Consider first a directional algorithm that chooses to descend to the left subtree. Its expected cost is: $p_k L(k-1,1) + (1-p_k)[L(k-1,0) + L(k-2,1)]$. If instead the directional algorithm chooses to descend to the right subtree, then its expected cost is $(1-p_k)L(k-2,1) + p_k[L(k-2,0) + L(k-1,1)]$, which by the choice of p_k is the same as the cost of the algorithm that descended to the left.

Hence, by using (5.1), we have that

$$L(k,0) = p_k L(k-2,0) + (1-p_k)L(k-1,0) + L(k-3,0) + (1-p_k)L(k-4,0) \tag{5.3}$$

Define the sequence ρ_k as the solution to the $L(k,0)$ recurrence:

$$\rho_k = \frac{\rho_{k-1}}{a^2+a} + \frac{\rho_{k-2}}{a^2+a} + \frac{\rho_{k-3}}{a^3} + \frac{\rho_{k-4}}{a^5+a^4} \tag{5.4}$$

$$\rho_4 = \frac{6.558171076}{a^4} \tag{5.5}$$

$$\rho_3 = \frac{4.512036108}{a^3} \tag{5.6}$$

$$\rho_2 = \frac{3.189189189}{a^2} \tag{5.7}$$

$$\rho_1 = \frac{2.2}{a} \tag{5.8}$$

Define

$$L(k,0) = \rho_k a^k \tag{5.9}$$

and r_k as the recurrence vector of ρ_k

$$\mathbf{r}_k = \begin{pmatrix} \rho_{k+3} \\ \rho_{k+2} \\ \rho_{k+1} \\ \rho_k \end{pmatrix} \qquad (k = 1, \ldots)$$

Then we have $\mathbf{r}_k = A\mathbf{r}_{k-1}$, where

$$A = \begin{pmatrix} \frac{1}{a+a^2} & \frac{1}{a+a^2} & \frac{1}{a^3} & \frac{1}{a^4+a^5} \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{5.10}$$

The characteristic equation is:

$$\det(A - \lambda I) = \det\left(\begin{pmatrix} \frac{1}{a+a^2} - \lambda & \frac{1}{a+a^2} & \frac{1}{a^3} & \frac{1}{a^4+a^5} \\ 1 & -\lambda & 0 & 0 \\ 0 & 1 & -\lambda & 0 \\ 0 & 0 & 1 & -\lambda \end{pmatrix}\right) = 0 \tag{5.11}$$

which solved for λ gets the claimed eigenvalues with

$$\lambda^4 - \frac{\lambda^3}{a+a^2} - \frac{\lambda^2}{a+a^2} - \frac{\lambda}{a^3} - \frac{1}{a^4+a^5} = 0 \tag{5.12}$$

For a, with:

$$A\mathbf{v}_1 = \mathbf{v}_1 \tag{5.13}$$

where $\mathbf{v}_1$ is[1]

$$\begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

We could get:

$$a^5 + a^4 - 2a^3 - a^2 - a - 1 = 0$$

Let a = 1.439650182. We could get:

$$\lambda^4 - 0.284718354857539\lambda^3 - 0.095421125796929 - 0.335142164770774\lambda - 0.284718354857539\lambda^2 = 0$$

whose root is:

$$\begin{pmatrix} 1.000000000 \\ -0.313238282 \\ -0.201021681 + 0.514021596i \\ -0.201021681 - 0.514021596i \end{pmatrix}$$

We could see that the eigenvalues of A are $\lambda_1 = 1$ and other eigenvalues$(\lambda_2, \lambda_3, \lambda_4)$ with real part less than 0. Therefore $\mathbf{r}_k$ tends to the projection of $\mathbf{r}_1$ along an eigenvector $\mathbf{v}_1$ corresponding to λ_1. By using the corresponding eigenvectors to find the component r of $\mathbf{r}_1$ along $\mathbf{v}_1$, we could prove the claim. We could get $\mathbf{r}_1$ is:

$$\mathbf{r}_1 = \begin{pmatrix} \rho_4 \\ \rho_3 \\ \rho_2 \\ \rho_1 \end{pmatrix}$$

[1] This is not the vector that you use later on

By relationship:

$$\begin{pmatrix} \mathbf{v}_1 & \mathbf{v}_2 & \mathbf{v}_3 & \mathbf{v}_4 \end{pmatrix} \begin{pmatrix} r \\ s \\ t \\ u \end{pmatrix} = \begin{pmatrix} \rho_1 \\ \rho_2 \\ \rho_3 \\ \rho_4 \end{pmatrix}$$

We could get $r = -3.052885190463789$. Since $lim_{k\to\infty} r_k = r\mathbf{v}_1$ We have that

$$lim_{k\to\infty}\rho_k = -0.5r = 1.52644259523189 \tag{5.14}$$

Lemma 4. *$L(k,0)$ is bounded by $L(k,0) = \rho_k a^k$*

PROOF. Define for convenience of notation $T(k) = L(k,0)$. By (5.9) , we have that $T(1) = \rho_1 a, T(2) = \rho_2 a^2, T(3) = \rho_3 a^3, T(4) = \rho_4 a^4$. By induction process, (5.3), and definition (5.4) we have $T(k) = \frac{a^{k-1}}{a+1}\rho_{k-1} + \frac{a^{k-1}}{a+1}\rho_{k-2} + a^{k-3}\rho_{k-3} + \frac{a^{k-4}}{a+1}\rho_{k-4}$, which completes the proof. □

By combining the lemma and recurrence relationship we obtain the following tighter lower bound:

Theorem 5. *We have that* $\lim_{k\to\infty} L(k,0)/a^k = \rho$ *and* $\lim_{k\to\infty} L(k,1)/a^{k-2} = (a+1)\rho$.

5.2 skew F-tree

5.2.1 Definition

A skew F-tree $S^k$is a NOR tree defined inductively: S^k is a single vertex, and for every specific vertex S^n in higher level will have left sub-tree as a single vertex, and right sub tree S^{n-1}.[17]

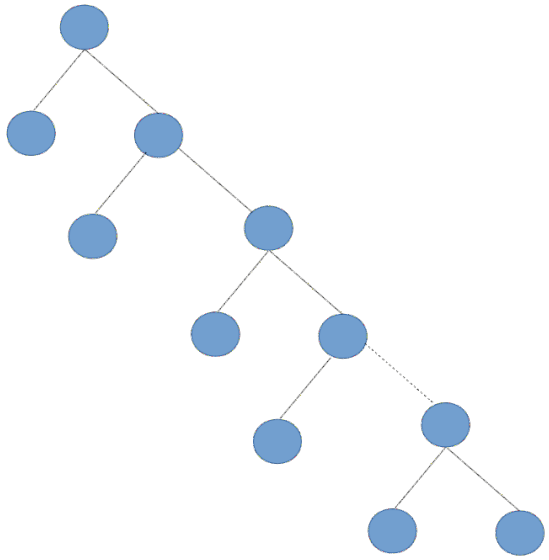


Figure 5.2. skew-F tree Example

5.2.2 Evaluation of skew F-tree

From the definition in the last section, it is clear that:

$$L(k,1) = L(k-1,0)+1 \tag{5.15}$$

$$L(0,i) = 1 \quad (i = 0,1) \tag{5.16}$$

and also that $L(k,1) \leq k+1$. The reluctant input is defined as follows. If we wish that the k-th Skew-F tree ($k > 0$) evaluates to 0, then the input will make sure that at least one of the subtrees evaluates to 0. Specifically, the left subtree evaluates to 0 with probability p_k and to 1 with probability $1-p_k$. Consequently, the right subtree evaluates to 0 with probability $1-p_k$ and to 1 with probability p_k.

$$L(k,0) = p_k + (1-p_k)(L(k-1,1)+1) \tag{5.17}$$

$$L(k,0) = p_k(L(k-1,0)+1) + (1-p_k)L(k-1,1) \tag{5.18}$$

Therefore we could have:

$$p_k = \frac{1}{1+L(k-1,0)} = \frac{1}{L(k,1)} \tag{5.19}$$

Consider first a directional algorithm that chooses to descend to the left subtree. Its expected cost is (5.17) If instead the directional algorithm chooses to descend to the right subtree, then its expected cost is (5.18) which by the choice of p_k is the same as the cost of the algorithm that descended to the left.

Therefore, by using (5.19) and (5.15), we could get:

$$L(k,0) = \frac{L(k-1,0)L(k-2,0)+2L(k-1,0)+1}{1+L(k-1,0)} \tag{5.20}$$

$$L(k,1) = L(k-2,1) - \frac{L(k-2,1)}{L(k-1,1)} + 2 \tag{5.21}$$

Then we will show that $L(k,1) \geq \frac{k}{2}$ by induction:

PROOF. We start an induction with:

$$L(0,1) = 1 \tag{5.22}$$

$$L(1,1) = 2 \tag{5.23}$$

$$L(2,1) = 2.5 \tag{5.24}$$

$$L(3,1) = 3.25 \tag{5.25}$$

Ee could see that $L(k,1) \geq \frac{k}{2}$ stands in base condition. With (5.19), we know that when $L(k,1)$ becomes large enough, we could say $pk = 0$. Considering $L(k,0)$ when k is even and large enough so we could consider p_k as 0, we could have:

$$L(k,0) = L(k-1,1) \tag{5.26}$$

$$L(k-1,1) = 1 + L(k-2,0) \tag{5.27}$$

Then we could have $L(k,0) = \frac{k+1}{2}$ and $L(k,1) \geq \frac{k}{2} + 1$. □

In conclusion, for skew-F tree, $\frac{k}{2} + 1 \leq L(k,1) \leq k+1$.

5.2.3 conclusion

From these two experiments, we have showed that this game tree evaluation algorithm could also be used for evaluating other types game tree for getting their lower bounds. Many other types of binary game tree could also be tested, but as the whole method is fixed, and we could prove it by Yao's principle. Therefore we could say that this game tree evaluation analysis process has versatility.

6 Conclusion and Future Work

In this chapter, we will discuss some of the strengths and weaknesses of the new game tree evaluation process, and also describe some methods for future work.

All in all, we believe that we have proposed an algorithm for evaluating uniform binary game tree with comparison in Chapter 4 and Chapter 5. We have made very detailed calculation for both directional and Non-directional algorithm's lower bound and prove it with iteration. We have shown this method could also be used for evaluating some more complex type binary game trees but not only uniform tree, including Fibonacci tree, skew-F tree.

One important strength of our analysis is trying to find out the best probability distribution at first and tighter the lower bound by including conditional probability. This helps us get the better lower bound than "Randomized Algorithm".

In short, we have constructed an algorithm with better lower bound compared with classical game tree evaluation steps which could be used for many cases of binary game tree evaluation.

One important weakness of this algorithm is from the analysis it self. We made some limitation in analysis, which may lead to some loose for the final lower bound result. A tighter lower bound still could be found. Also this algorithm only could be applied for

binary game tree. It is questionable its usage in ternary and other types of game tree. In addition, although we already know it is more efficient in binary game tree, we don't know whether this is still good enough in other cases.

Therefore it is clear that there are some future work needed to be done for this algorithm. First of all, this evaluation process only been demonstrated in binary game tree. It is clear that there are many other types of a game trees, like ternary tree, quad tree, or some realistic game tree like the tic-tac-toe game tree, chess board game tree. A more general algorithm may be needed to propose for game tree evaluation based on the previous method, which may includes the number of nodes in each level, different type of boolean functions and so on. Then the algorithm could be applied for many different cases.

Then it is clear that most of the experiments and analysis are for theatrical field. We could not figure out how this promotion of algorithm could be used for some practical environment. this also could be an important part for the future work. Combined with the a more general evaluation algorithm, some cool things maybe done. For example maybe we could use this analysis method for realistic chess game cases, and calculate the probability to win for both of the two players.

Last but not least, although our algorithm is good enough for game tree evaluation, most of its method is based and inspired by the methods that already known. Obviously, if we could find some new approach that is totally unknown and different from the past will be a very cool thing.

In conclusion. A more efficient approach for binary game tree evaluation has been proposed by us. It has better lower bound and could be used for evaluates different types of binary game tree. Considering its weakness of using for non-binary game tree,

the future work should focus on improve this method for more types of game tree. If possible try its usage in some practical filed like board game will be good. In theoretical aspect, to discover a totally new approach different from previous will be a good method.

Appendix A
Detailed Calculation

0.1 Lemma 2

The characteristic equation is:

$$A-\lambda I=\begin{pmatrix}\frac{1}{2a}-\lambda & \frac{2}{a^2}\\ 1 & -\lambda\end{pmatrix}=\lambda^2-\frac{1}{2a}\lambda-\frac{2}{a^2}=0 \tag{A.1}$$

which solved for λ gets the claimed eigenvalues.

We know that $\mathbf{v}_1 = A\mathbf{v}_1$, which is:

$$\begin{pmatrix}x\\ y\end{pmatrix}=\begin{pmatrix}\frac{1}{2a} & \frac{2}{a^2}\\ 1 & 0\end{pmatrix}\begin{pmatrix}x\\ y\end{pmatrix} \tag{A.2}$$

By calculation:

$$x=\frac{1}{2a}x+\frac{2}{a^2}y \tag{A.3}$$

$$y=x \tag{A.4}$$

By (A.3), we could find x could be any value. We already know that $\mathbf{v}_1$ is a unit vector.

Because $A\mathbf{v}_2 = \lambda_2\mathbf{v}_2$. Thus we could find:

$$\lambda_2\begin{pmatrix}x\\ y\end{pmatrix}=\begin{pmatrix}\frac{1}{2a} & \frac{2}{a^2}\\ 1 & 0\end{pmatrix}\begin{pmatrix}x\\ y\end{pmatrix} \tag{A.5}$$

By calculation,

$$\lambda_2 x=\frac{1}{2a}x+\frac{2}{a^2}y \tag{A.6}$$

$$\lambda_2 y=x \tag{A.7}$$

Therefore we could get:

$$\mathbf{v}_2 = \frac{1}{\sqrt{\lambda_2^2+1}} \begin{pmatrix} \lambda_2 \\ 1 \end{pmatrix}$$

The system of equation becomes:

$$\frac{\sqrt{2}}{2}\rho + \frac{\lambda_2}{\sqrt{1+\lambda_2^2}}\alpha = \rho_2$$

$$\frac{\sqrt{2}}{2}\rho\lambda_2 + \frac{\lambda_2}{\sqrt{1+\lambda_2^2}}\alpha = \rho_1\lambda_2$$

and subtracting the second equation from the first:

$$\rho = \frac{\rho_1\lambda_2 - \rho_2}{\frac{\sqrt{2}}{2}(\lambda_2 - 1)} = \frac{40+8\sqrt{33}}{\sqrt{66}+33\sqrt{2}} \approx 1.5687$$

0.2 Lower Bound Comparison

Since

$$\rho = \frac{20\sqrt{2}+4\sqrt{66}}{\sqrt{66}+33\sqrt{2}}\,,$$

$$a = (1+\sqrt{33})/4\,,$$

and the previous lower bound is $(2-p)^k$ where $p = (3-\sqrt{5})/2$. Then,

$$\lim_{k\to\infty} \frac{\rho\alpha^k}{(2-p)^k} = \infty\,,$$

which implies that the new lower bound is strictly tighter than the previous one.

Complete References

[1] Man-tak Shing Te Chiang Hu. Combinatorial algorithms. Combinatorial Algorithms, 2002.

[2] Daniele Gardy Bernhard Gittenberger Michael Drmota, Philippe Flajolet. Mathematics and Computer Science III Algorithms, Trees, Combinatorics and Probabilities. 2000.

[3] Victor Allis. Searching for solutions in games and artificial intelligence. University of Limburg, Maastricht, The Netherlands, 1994.

[4] Daniel Roche. Randomness in computing, game tree unit. United States Academy, 2013.

[5] Prabhakar Raghavan Rajeev Motwani. Randomized algorithm. pages 28–42, 1995.

[6] Andrew Yao. Probabilistic computations: Toward a unified measure of complexity. Proceedings of the 18th IEEE Symposium on Foundations of Computer Science (FOCS), pages 222–227, 1977.

[7] Shmuel Zamir Michael Maschler, Eilon Solan. Game Theory. 2013.

[8] Ann. Contributions to the theory of games. 1950.

[9] Lynn H Loomis. On a theorem of von neumann. National Academy of Sciences, 1946.

[10] F. Meyer auf der Heide. Nondeterministic versus probabilistic linear search algorithms. Proc.26th Annual Symposium on Foundations of Computer Science, pages 65–73, 1985.

[11] Steven D. Galbraith. Mathematics of public key cryptography. Mathematics of Public Key Cryptography, pages 16–17, 2012.

[12] M Snir. Lower bounds for probabilistic linear decision trees. Theoretical Computer Science, 1985.

[13] U. Manber and M. Tompa. The complexity of problems on probabilistic nondeterministic and alternating decision trees. JACM, 1985.

[14] Andrew Yao. Probabilistic computations: towards a unified measure of complexity. Proc. 18th Annual Symposium on Foundations of Computer Science, 1977.

[15] Ran Borodin Allan, El-Yaniv. Online computation and competitive analysis. 2005.

[16] Yasuichi Horibe. "notes on fibonacci trees and their optimality". 1982.

[17] Avi Widerson Michael Saks. Probabilistic boolean decision trees and the complexity of evaluating game trees. IEEE, 1986.