CSC494 Project: Branching Program Complexity for the Height 4 Tree Evaluation Problem

Eric Bannatyne

May 24, 2016

1 Introduction

One of many major open problems in complexity theory consists in understanding the relationship between the complexity classes **L**, **NL**, and **P**. It is a well-known fact that $\mathbf{L} \subseteq \mathbf{NL} \subseteq \mathbf{P}$, however it is unknown whether any of these inclusions is proper. This has prompted numerous attempts to either prove that two of these classes are equal, or to provably separate them. This problem of separating **L** and **NL** from **P** motivates the study of the Tree Evaluation Problem (TEP), introduced by Cook et al. in [2]. The TEP is known to be solvable in polynomial time, using a standard black pebbling algorithm, however it is conjectured that the problem is neither in **L** nor in **NL**. In order to prove superlogarithmic space lower bounds for $FT^h_d(k)$ (using the notation from [2]), it suffices to prove a lower bound of $\Omega(k^{r(h)})$ for the number of states required by a branching program solving $FT^h_d(k)$, for some unbounded function $r(h)$.

Although the general problem has so far resisted proof, it has been possible to prove good lower bounds on the sizes of branching programs solving the TEP in a number of restricted models, such as thrifty branching programs [2] and read-once branching programs [3, 1]. For general branching programs, lower bounds are known for certain special cases of the TEP, including $FT^3_2(k)$, the height 3 TEP, as well as $\mathit{Children}^4_2(k)$, the problem of determining the values of the children of the root node in a height 4 tree. We hope to extend these results in order to prove good lower bounds for the height 4 TEP, by studying various restricted models and special cases of the general problem.

2 The Tree Evaluation Problem

The setup of the problem is as follows: Given $d, h \geq 2$, we write T^h_d to denote the perfect d-ary tree of height h, where the height of a tree refers to the number of levels in the tree. We focus mainly on the case when $d = 2$. The nodes v_i of T^h_d are numbered in heap style, so that the root node is v_1, and when $d = 2$, the left and right children of node v_i are v_{2i} and v_{2i+1}, respectively. The tree evaluation problem $FT^h_d(k)$ for the tree T^h_d is defined as follows:

Definition 1 (The Tree Evaluation Problem)**.** Let $d, h \geq 2$. An input I to $FT^h_d(k)$ encodes the following information:

1. A function $f^I_i : [k]^d \to [k]$ for every internal node v_i of T^h_d, and
2. an element of $[k]$ for every leaf of T^h_d.

We associate with every node v_i of T^h_d a value v^I_i. If v_i is a leaf labeled with the value $a \in [k]$, then $v^I_i = a$. If v_i is an internal node with children $v_{j_1}, \ldots, v_{j_d}$, then $v^I_i = f^I_i(v^I_{j_1}, \ldots, v^I_{j_d})$. The goal of $FT^h_d(k)$ on input I is to compute the value v^I_1 of the root.

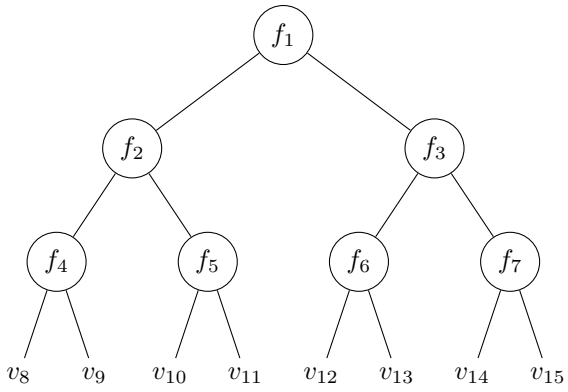

Figure 1: An illustration of the input to the height-4 TEP $FT_2^4(k)$.

Figure 1 shows an illustration of the TEP for the case $d = 2$ and $h = 4$. As the reader should verify, there is a straightforward polynomial-time algorithm to compute $FT_d^h(k)$.

2.1 Branching Programs

Branching programs are a model of computation that can be thought of as a generalization of decision trees, in which the underlying graph is allowed to be a directed acyclic graph instead of a tree. A branching program contains a finite set of states, represented as nodes in a DAG. For our purposes, a state can either query the value of a leaf v_i of T_d^h, or it can query the value of the function $f_i(a_1, \ldots, a_d)$ for some fixed values $a_1, \ldots, a_d \in [k]$, or it can be an output state. For every query state γ, there are k edges from γ going to other states, corresponding to the possible results of the query made by γ. There are exactly k output states, one for each value in $[k]$, with no out-edges. Finally, there is a single starting state, with no edges entering that state.

Given a branching program and an input I to $FT_d^h(k)$, the computation $C(I)$ is a path in the graph of the branching program, defined as one might expect: Beginning at the start state, we follow the edge corresponding to the result of the current state's query, and we continue until we reach an output state.

For a more precise definition of the computational model, see [2]. Note that branching programs are a nonuniform model of computation, meaning that we can have a *different* branching program solving $FT_d^h(k)$ for every value of k. For any given degree d and height h, we are interested in the size that a branching program solving $FT_d^h(k)$ must have as k increases.

In [2], Cook et al. proved that any branching program (as defined above) solving $FT_2^3(k)$ requires at least $k^3/\log k$ states.

3 Restricted Lower Bounds for $FT_2^4(k)$

In this section, we present proofs of several lower bounds on the number of branching program states needed to solve the problem $FT_2^4(k)$ for height 4 trees, under various restrictions on the branching programs used to solve $FT_2^4(k)$. These restrictions can be viewed as placing constraints on how and when the branching program queries the root function. Indeed, the problem of proving lower bounds for the height 4 problem seems to be much difficult when the branching program's root queries are unconstrained, since it becomes much more difficult to reason about what information the branching program must learn in order to determine the correct output. However, we will see that if we make certain assumptions on the model it becomes possible

to prove the desired lower bound of $\Omega(k^4)$ states to solve $FT_2^4(k)$. Throughout this section, we assume that the root function f_1 is given as part of the input to $FT_2^4(k)$.

3.1 Root Function Queried Last

We first consider the restriction that the branching program used to solve $FT_2^4(k)$ makes all of its root function queries after all other (i.e. non-root) queries have been made. This corresponds to our intuition that querying the root function before knowing the values of its children should not enable the branching program to learn enough about the input in order to determine the correct output. In this setting, we can show that any branching program that solves $FT_2^4(k)$ must, at some point, learn the values of the children of the root, which we know from [2] to require $k^4/2$ states. The general method behind this lower bound proof is to use an adversarial path switching argument to show that any computation path on the branching program must have some state at which the children are known. From there, we can then convert the branching program solving $FT_2^4(k)$ to one that solves $Children_2^4(k)$, for which we already have a lower bound.

Theorem 1. *Let B be a deterministic k-way branching program solving $FT_2^4(k)$, such that for every input I, the computation $C(I)$ of B on I is a sequence of states $\gamma_1, \gamma_2, \ldots, \gamma_R, \ldots, \gamma_T$, where the states $\gamma_1, \ldots, \gamma_{R-1}$ do not query the root function, and the states $\gamma_R, \ldots, \gamma_{T-1}$ consist only of root queries, and γ_T is an output (sink) state. Then B has at least $\Omega(k^4)$ states.*

Proof. In order to prove this statement, we can describe a method for converting the branching program B, solving $FT_2^4(k)$, into a deterministic branching program B' that solves $Children_2^4(k)$. We can then use the known lower bound of $k^4/2$ for $Children_2^4(k)$ to derive a lower bound for $FT_2^4(k)$ in this restricted model.

For any input I to B, let γ_r^I denote the first state that queries the root function on the computation path $C(I)$ of B on input I. We shall argue that, when the computation reaches the state γ_r^I, the branching program must "know" the values v_2^I and v_3^I of the left and right children of the root, respectively. To show this, we make the following claim:

Claim 1. Let I and J be inputs to B such that $(v_2^I, v_3^I) \neq (v_2^J, v_3^J)$. Then $\gamma_r^I \neq \gamma_r^J$.

To prove this claim, assume for a contradiction that I and J are inputs to B such that $(v_2^I, v_3^I) \neq (v_2^J, v_3^J)$, but $\gamma_r^I = \gamma_r^J$. We can adversarially construct inputs I' and J' as follows:

1. $v_i^{I'} = v_i^I$ and $v_i^{J'} = v_i^J$ for $8 \leq i \leq 15$,
2. $f_i^{I'}(a,b) = f_i^I(a,b)$ and $f_i^{J'}(a,b) = f_i^J(a,b)$ for $2 \leq i \leq 7$ and $a, b \in [k]$,
3. $f_1^{I'}(v_2^{I'}, v_3^{I'}) = f_1^{J'}(v_2^{I'}, v_3^{I'}) = 0$, and $f_1^{J'}(v_2^{J'}, v_3^{J'}) = f_1^{I'}(v_2^{J'}, v_3^{J'}) = 1$,
4. $f_1^{I'}(a,b) = f_1^{J'}(a,b)$ for all $a, b \in [k]$.

The first two lines of this construction assert that the leaf values and non-root functions of I' and J' are identical to those of I and J, respectively. The last two lines ensure that I' and J' both have identical root functions, but have different root values. In particular, lines 1 and 2 imply that $v_i^{I'} = v_i^I$ and $v_i^{J'} = v_i^J$ for $j \in \{2, 3\}$, so $(v_2^{I'}, v_3^{I'}) \neq (v_2^{J'}, v_3^{J'})$, meaning that lines 2 and 3 do not conflict with one another.

The computation path $C(I')$ must be identical to $C(I)$ up until the state γ_r^I is reached, since I' agrees with I on all non-root values. Similarly, the computation of the input J' will follow the same path as that of J up until the computation reaches γ_r^J. Therefore $\gamma_r^{I'} = \gamma_r^I = \gamma_r^J = \gamma_r^{J'}$. Moreover, since $\gamma_r^{I'}$ makes a root query, all states following $\gamma_r^{I'}$ on any computation path must also only make root queries. However, because I' and J' have identical root functions, their computations must follow the same path, and in particular

they must reach the same final output state. Therefore B must output an incorrect answer on one of these inputs.

Using the above claim, we can convert B to a new deterministic k-way branching program B' that solves $\mathit{Children}_2^4(k)$: For every input I, simply replace the state γ_r^I with an output state corresponding to the pair (v_2^I, v_3^I). Then B' correctly solves $\mathit{Children}_2^4(k)$, since, by the claim, the state γ_r^I is only ever reached by inputs J such that $v_i^J = v_i^I$ for $i \in \{2, 3\}$. Therefore, we know that B' has at least $k^4/2$ states, and so it follows that B must also have at least $k^4/2$ states. □

3.2 All Root Queries Are Thrifty

For any internal node i of an input I to $FT_2^4(k)$, we say that a query of the form $f_i(a, b)$ for $a, b \in [k]$ is *thrifty* if $(a, b) = (v_{2i}^I, v_{2i+1}^I)$, the correct values of the children of node i. If we assume that all function queries made by the branching program are thrifty, then it is possible to prove that $\Omega(k^h)$ states are required for all heights h. However, in the case of $h = 4$, it suffices to assume that all root function queries are thrifty, via a simple application of the lower bound for $\mathit{Children}_2^4(k)$.

Theorem 2. *Let B be a deterministic k-way branching program solving $FT_2^4(k)$ such that, on every computation, all root queries are thrifty. Then B has at least $\Omega(k^4)$ states.*

Proof. Again, we will show that B can be converted to a branching program B' that solves $\mathit{Children}_2^4(k)$. To achieve this, observe that for any input I the computation $C(I)$ must include at least one root query. Moreover, if $C(I)$ visits a state querying $f_1^I(a, b)$ for some values $a, b \in [k]$, then it must be the case that $a = v_2^I$ and $b = v_3^I$, since all root queries are assumed to be thrifty. Therefore we can simply replace every such root query state with an output state whose value is the pair (a, b). The resulting deterministic k-way BP solves $\mathit{Children}_2^4(k)$, for which we can apply the existing lower bound, from which it follows that B has at least $k^4/2$ states. □

3.3 Restricted Thrifty Root Queries

In principle, it may intuitively seem reasonable to assume that all queries made by a branching program are thrifty queries. Any branching program implementing the black pebbling algorithm is a thrifty branching program, and is conjectured to be optimal for solving $FT_d^h(k)$ in general. In practice, though, this may not be a reasonable assumption to make, because if a branching program is non-thrifty, that does not necessarily rule out the possibility that it can learn complex relationships between the values of certain nodes without directly computing the values of their children. However, it is easy to show that any branching program solving $FT_d^h(k)$ must have at least *one* thrifty root query on every computation path (otherwise an adversary could change the value of the root, and the branching program would be none the wiser). In this case, rather than assuming that all root queries are thrifty, we allow some root queries to be non-thrifty, however we restrict the position of the thrifty root query on a given computation path, namely, we assume that the thrifty root query occurs after all non-root queries have been made. This restriction alone has not proven to be sufficient to prove an $\Omega(k^4)$ lower bound. However, if we combine this restriction with the assumption that each possible value of the root node is read-once, then we can prove the following lower bound.

Theorem 3. *Let B be a deterministic k-way branching program solving $FT_2^4(k)$, such that for every input I, and $a, b \in [k]$, the computation path $C(I)$ queries the value of $f_1(a, b)$ at most once, and the thrifty query to $f_1^I(v_2^I, v_3^I)$ may only occur after all non-root queries have been made. Then B has at least $\Omega(k^4)$ states.*

Proof. Let I be an input to $FT_2^4(k)$, and let γ be the state making the thrifty root query (i.e. γ queries $f_1^I(v_2^I, v_3^I)$) on the computation path $C(I)$, and let J be some input such that $C(J)$ also visits γ. We claim that $v_2^I = v_2^J$ and $v_3^I = v_3^J$. We will assume that this is not the case, in hopes of obtaining a contradiction.

First, we observe that every state that is a descendant of γ in the underlying DAG of B must query the root function. Since the root function is read-once, this means that any possible sequence of states following γ must be consistent with any computation path leading up to γ. In particular, we can construct new inputs I' and J' that will follow the same paths as I and J up to γ, which will be forced to follow the same path after γ.

Let Q be the set of all pairs $(a, b) \in [k]^2$ such that the value $f_1(a, b)$ is queried by γ or by some state that is a descendant of γ. Now define inputs I' and J' so that I' is identical to I, and J' is identical to J, with the exception that $f_1^{I'}(v_2^J, v_3^J) = f_1^{J'}(v_2^J, v_3^J) = 1$, and $f_1^{I'}(a, b) = f_1^{J'}(a, b) = 0$ for all $(a, b) \in Q - \{(v_2^J, v_3^J)\}$.

Since I' and J' agree on all inputs to the root function in Q, their computations must follow the same path after γ, and they must ultimately reach the same output state. However, since $(v_2^I, v_3^I) \in Q - \{(v_2^J, v_3^J)\}$, and we also have $(v_2^{I'}, v_3^{I'}) = (v_2^I, v_3^I)$ and $(v_2^{J'}, v_3^{J'}) = (v_2^J, v_3^J)$, it follows that our new inputs have root values $v_1^{I'} = 0$ and $v_1^{J'} = 1$. Therefore B must output an incorrect answer on either I' or J'. So it must be the case that $v_2^I = v_2^J$ and $v_3^I = v_3^J$.

Therefore, for any input I, there must be some state γ (i.e. when the thrifty root query is made) at which point the computation "knows" the values of the children v_2^I and v_2^I. We can therefore convert B to a branching program B' solving the problem $Children_2^4(k)$ by replacing γ with an output state labeled (v_2^I, v_3^I). Thus B' is obtained by repeating this process for every possible input I. It then follows that the existing $k^4/2$ lower bound for the number of branching program states needed to solve $Children_2^4(k)$ also applies to the number of states in B. □

So far all of our lower bound proofs have consisted of showing that a branching program with certain restrictions solving $FT_2^4(k)$ must do so by first solving the $Children_2^4(k)$ problem, and then applying the known lower bound for that problem. However it is possible to approach this problem from another perspective, by examining the number of *leaf queries* required for solving $FT_2^3(k)$. In the next section, we will show that a lower bound of $\Omega(k^2)$ on the number of states querying leaves required to solve $FT_2^3(k)$ would imply that solving $FT_2^4(k)$ requires $\Omega(k^4)$ states. However, this next lower bound is included in this section because it assumes the same set of restrictions on the branching program as in Theorem 3. In fact, the following result implies Theorem 3, as we shall see later. Moreover, it is interesting to note that the proof of this result differs from the lower bound proofs we have seen so far, in that we use a tag argument to show that the branching program requires at least $k^2/2$ states, as opposed to converting the branching program to one solving $Children_2^4(k)$.

Theorem 4. *Let B be a deterministic k-way branching program solving $FT_2^3(k)$, such that for every input I, and $a, b \in [k]$, the computation path $C(I)$ queries the value of $f_1(a, b)$ at most once, and the thrifty query to $f_1^I(v_2^I, v_3^I)$ may only occur after all non-root queries have been made. Then B has at least $k^2/2$ states that query leaf values.*

Proof. Let E be the set of all inputs I to $FT_2^3(k)$ such that $f_2^I = f_3^I = +_k$ (addition mod k). Let Γ be the set of all states in B that query leaf values, and assume for a contradiction that $|\Gamma| < k^2/2$.

For every input $I \in E$, let $T(I) = (\gamma^I, v_i^I)$, where γ^I is the last state on the computation $C(I)$ that queries a leaf, and $i \in \{4, 5, 6, 7\}$ is the node queried by γ^I. Moreover, we define a second tagging function U such that for any input I,

$$U(I) = \begin{cases} (v_4^I, v_5^I, v_3^I) & \text{if } \gamma^I \text{ queries node 4 or 5} \\ (v_6^I, v_7^I, v_2^I) & \text{otherwise.} \end{cases}$$

Next, we can show that the image of E under U must have at least $k^3/2$ distinct triples. To prove this claim, let $E' \subseteq E$ be the set of all inputs I where nodes 4, 5, 6 and 7 have the form a, b, a, c for $a, b, c \in [k]$.

Then if $I \in E'$, either $U(I) = (a, b, a +_k c)$ or $U(I) = (a, c, a +_k b)$. There are a total of k^3 distinct triples of either formr, and at least half of the triples must be from one of the two forms, so $|U(E)| \geq |U(E')| \geq k^3/2$.

However, by the assumption that $|\Gamma| < k^2/2$, there are strictly fewer than $k^3/2$ possible values for $T(I)$ over all inputs $I \in E$. It follows that there must be inputs $I, J \in E$ such that $U(I) \neq U(J)$ but $T(I) = T(J)$, i.e. $(\gamma^I, v_i^I) = (\gamma^J, v_j^J)$, and thus $i = j$. This implies that either $v_2^I \neq v_2^J$ or $v_3^I \neq v_3^J$. This is because if we fix either node 2 or 3 to some value, then fixing the values of one of its children determines the value of the other child. Thus I and J have different children values, that is, $(v_2^I, v_3^I) \neq (v_2^J, v_3^J)$.

Since I and J both reach the state γ^I on their last leaf query and follow the same edge out of γ^I, it is possible to choose values of the root functions (using the fact that the root is read once, and I and J have identical functions at nodes 2 and 3) so that $C(J)$ must visit the state on $C(I)$ at which the thrifty root query occurs for I. However, using an argument identical to that of Theorem 3, it is possible to choose the values of I and J so that $v_1^I \neq v_1^J$, but $C(I)$ and $C(J)$ reach the same final output state, yielding a contradiction. Therefore B must have at least $k^2/2$ states that query leaf values. □

4 Lower Bounds on Leaf Queries With Fixed Functions

The next two results, originally from [1], show that it is possible to prove a lower bound of $\Omega(k^e)$ for the number of branching program states required to solve the height h TEP $FT_2^h(k)$ by proving a lower bound of $\Omega(k^{e-2})$ on the number of states querying *leaves* required to solve the height $h-1$ problem $FT_2^{h-1}(k)$.

Theorem 5. *Let $h \geq 3$ and let B be a deterministic branching program with s states solving $FT_2^h(k)$. Then B can be transformed into a deterministic branching program B' solving $FT_2^{h-1}(k)$ in which the number of states querying leaves is at most s/k^2.*

Proof. Let B be a branching program with s states that solves $FT_2^h(k)$. Then there exists some $r, r' \in [k]$ such that at most s/k^2 states that make queries of the form $f_j(r, r')$, where j is a node at level 2 of the tree (i.e. the children of node j are leaves). We can construct a branching program B' solving $FT_2^{h-1}(k)$ with at most s/k^2 states that query leaves. Intuitively, this is achieved by simulating B for the case in which, for each node j at level 2, the function f_j is such that $f_j(r, r')$ is the value of the leaf j in T_2^{h-1} and $f_j(a, b) = 1$ for all $(a, b) \neq (r, r')$, and the children of node j have the values $(v_{2j}, v_{2j+1}) = (r, r')$.

To construct B', replace every state of B that queries $f_j(r, r')$, where j is a level-2 node of T_2^h, with one that makes the leaf query v_j. Then remove every state γ querying $f_j(a, b)$ for $(a, b) \neq (r, r')$, and reroute all edges into γ to the destination of the edge labeled 1 leaving γ. Finally, remove every state δ querying a leaf of T_2^h, and replace every edge into δ by sending it to the destination of the outedge of δ labeled r or r' depending on whether the leaf is a left or right child, respectively. □

Theorem 6. *Let $h \geq 3$ and let B be a deterministic branching program solving $FT_2^{h-1}(k)$ with s states that query leaves and $O(sk^2)$ states in total. Then B can be converted to a deterministic branching program B' which solves $FT_2^h(k)$ with $O(sk^2)$ states.*

Proof. Given B, one can construct B' by simply replacing every state of B making a leaf query with a subprogram that evaluates the corresponding level-2 node in T_2^h. Each such subprogram requires $O(k^2)$ states, giving $O(sk^2)$ states overall. □

Treating the functions at the internal nodes of the tree T_2^h as fixed enables us to study branching programs that only make leaf queries. This approach was used by James Cook and Siu Man Chan in [1] to prove a lower bound of $\Omega(k^h)$ states for deterministic read-once branching programs solving $FT_2^h(k)$. In their lower bound proof, they consider the internal functions of T_2^h to be polynomials over the finite field $GF(k)$ for $k = 2^d > 3^{h-1}$, where d is an odd positive integer. The internal nodes are fixed to be the Siu Man polynomial

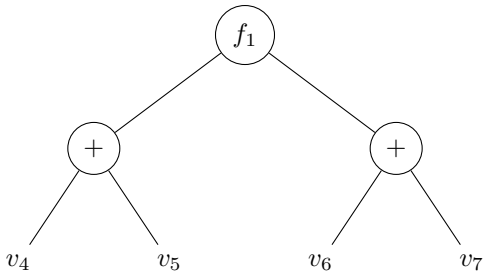

Figure 2: The tree T_2^3 with the root node fixed to be the Siu Man polynomial $f_1(a,b) = a^3 + b^3$ over $GF(k)$, and the children of the root fixed to be addition.

$f_i(a,b) = a^3 + b^3$ over $GF(k)$. The Siu Man polynomial has a number of useful properties. First, if either argument is fixed, it becomes a permutation polynomial in one variable. Moreover, we can consider the last-leaf function $f(x)$, which outputs the value of the root depending x, the value of the last leaf j queried by a branching program, after all of the other leaves have been queried. The crucial property of the Siu Man polynomial is that every possible setting of the tuple $(s_1, \ldots, s_{h-1})$ yields a different function f, where $s_1, \ldots, s_{h-1}$ are the siblings of the nodes along the path from the leaf j to the root. Then it follows that the branching program must know the values of $s_1, \ldots, s_{h-1}$ in order to determine the correct output.

In the rest of this section, we focus our attention on trying to prove an $\Omega(k^2)$ lower bound on the number of leaf queries for the height 3 case $FT_2^3(k)$, with the root function fixed to be the Siu Man polynomial $f_1(a,b) = a^3 + b^3$, and $f_1(a,b) = f_2(a,b) = a + b$ over $GF(k)$, where $k = 2^d > 3^{h-1}$ for some odd positive integer d, as illustrated in Figure 2. Since the functions are all fixed, the input consists simply of the quadruple (v_4, v_5, v_6, v_7) of leaf values, and any branching program can only make leaf queries.

4.1 Branching Program Tables

One strategy proposed by David Liu for proving lower bounds for the tree evaluation problem involves associating with every state and every edge of a branching program solving $FT_d^h(k)$ a table that encodes the possible values of certain "critical" nodes. For now we will restrict our attention to branching programs B solving the problem $FT_2^3(k)$, under the following restrictions:

1. $k = 2^d$, where $d \geq 5$ is an odd positive integer,
2. $f_1(a,b) = a^3 + b^3$ (the Siu Man polynomial) and $f_2(a,b) = f_3(a,b) = a + b$ for all $a, b \in [k]$, where the operations are considered to be over the field $F = GF(k)$,
3. For every input I to B, v_7 is the last leaf queried on $C(I)$.

Let B be a deterministic k-way branching program solving $FT_2^3(k)$ with the above restrictions, and let γ be any state of B.

Definition 2. The table of state γ is a function $T_\gamma : [k]^3 \to [k] \cup \{\varnothing\}$ such that, for all $(v_2, v_6, v_7) \in [k]^3$,

- If there is some input I with $(v_2^I, v_6^I, v_7^I) = (v_2, v_6, v_7)$ such that $C(I)$ reaches γ, then

$$T_\gamma(v_2, v_6; v_7) = f_1(v_2, f_3(v_6, v_7)) = v_2^3 + (v_6 + v_7)^3.$$

- Otherwise, $T_\gamma(v_2, v_6; v_7) = \varnothing$ (null, to indicate absence of a value).

Definition 3. The table of an edge labeled $c \in [k]$ leaving state γ is a function $T_{\gamma,c} : [k]^3 \to [k] \cup \{\varnothing\}$ such that, for all $(v_2, v_6, v_7) \in [k]^3$,

- If there is some input I with $(v_2^I, v_6^I, v_7^I) = (v_2, v_6, v_7)$ such that $C(I)$ reaches γ and $v_i^I = c$, where i is the node queried by γ, then

$$T_{\gamma,c}(v_2, v_6; v_7) = f_1(v_2, f_3(v_6, v_7)) = v_2^3 + (v_6 + v_7)^3.$$

- Otherwise, $T_{\gamma,c}(v_2, v_6; v_7) = \varnothing$.

A table in this context can be viewed as a 2-dimensional matrix with rows indexed by pairs (v_2, v_6) and columns indexed by values of v_7. At any given state (resp. edge), the table associated with that state (resp. edge) provides a tool for visualizing the possible node values of inputs that reach a given state (resp. edge) as well as the possible root values. The intention is to be able to prove lower bounds on the number of leaf queries needed for $FT_2^3(k)$ by observing how the tables associated with a branching program transform on a local scale, which enables us to model information that must be "known" by a portion of a branching program.

In order to compute the values in the table at a given node, we can apply some simple rules to inductively compute a node's table based on local information. As a simple base case, if γ is the initial state of B, then T_γ is simply a full table, i.e. with no null entries. Note that knowing the values of v_2, v_6 and v_7 is sufficient to compute the value of the root.

On the other hand, if γ is any other state, then consider all states γ' and $c \in [k]$ such that there is an edge from γ' to γ with the label c. Let $v_2, v_6, v_7 \in [k]$. If $T_{\gamma',c}(v_2, v_6; v_7) = \varnothing$ for all γ' and c, then $T_\gamma(v_2, v_6; v_7) = \varnothing$. Otherwise, $T_\gamma(v_2, v_6; v_7) = T_{\gamma',c}(v_2, v_6; v_7)$, which will be the same regardless of which γ' and c are chosen.

We can similarly define rules for determining the tables associated with edges. Let γ be a state of B and let $c \in [k]$. If γ queries v_7, then for all values $v_2, v_6, v_7 \in [k]$,

$$T_{\gamma,c}(v_2, v_6; v_7) = \begin{cases} T_\gamma(v_2, v_6; v_7) & \text{if } v_7 = c \\ \varnothing & \text{if } v_7 \neq c. \end{cases}$$

That is, if we take an edge labeled c exiting a state querying v_7, then every column must be filled with nulls, with the exception of column c. Similarly, if γ queries v_6, then we have

$$T_{\gamma,c}(v_2, v_6; v_7) = \begin{cases} T_\gamma(v_2, v_6; v_7) & \text{if } v_6 = c \\ \varnothing & \text{if } v_6 \neq c. \end{cases}$$

However, if γ queries v_4 or v_5, then simply knowing the table associated with γ is not sufficient to determine how to update the tables of edges exiting γ, as these tables do not distinguish between different values of v_4 and v_5. This is because, for now, we intend to treat v_7 as a "bottleneck node" of sorts, in which case the actual values of v_4 and v_5 are, it seems, not significant. In order to derive similar update rules, it would be necessary to store additional information at each node and edge, encoding the possible values of v_4 and v_5 that can reach that node or edge. However, this will not be necessary to prove the following result.

Theorem 7. *Let B be a deterministic k-way branching program solving the restricted version of $FT_2^3(k)$, as above. Moreover, assume that v_7 is the last leaf to be queried on any computation $C(I)$, and v_7 is queried at most once on every computation. Then B must have $\Omega(k^2)$ states that query leaves.*

Proof. Let I be some input to B, and let γ be the final state on $C(I)$ querying a leaf (i.e. assume that γ queries v_7). Since v_7 has not been queried before, it must be free to take on any value in $[k]$ at γ. This means that, for all $v_2, v_6 \in [k]$, if $T_\gamma(v_2, v_6; v_7) = \varnothing$ for some $v_7 \in [k]$, then $T_\gamma(v_2, v_6; v_7) = \varnothing$ for all $v_7 \in [k]$. Likewise, if $T_\gamma(v_2, v_6; v_7) \neq \varnothing$ for some $v_7 \in [k]$, then $T_\gamma(v_2, v_6; v_7) \neq \varnothing$ for all $v_7 \in [k]$. In other words, every row of T_γ either consists entirely of nulls or does not contain any null entries.

Moreover, and this is the crucial property of the Siu Man polynomial that is used in this proof, since every pair $(v_2, v_6) \in [k]^2$ induces a distinct root function of v_7, then every non-null row of T_γ must be distinct. For every $c \in [k]$, the edge labeled c out of γ leads to an output state, all non-null entries in $T_{\gamma,c}$ must be the same. But in $T_{\gamma,c}$, we have $T_{\gamma,c}(v_2, v_6; d) = \varnothing$ for all $d \neq c$. Suppose, for a contradiction, that (v_2, v_6) and (v_2', v_6') are pairs corresponding to two different non-null rows in T_γ. Then

$$T_{\gamma,c}(v_2, v_6; c) = T_{\gamma,c}(v_2', v_6'; c)$$

which implies that

$$T_\gamma(v_2, v_6; c) = T_\gamma(v_2', v_6'; c).$$

However, since this must be true for every c, it must follow that $T_\gamma(v_2, v_6; c) = T_\gamma(v_2', v_6'; c)$ for all $c \in [k]$, that is, rows (v_2, v_6) and (v_2', v_6') in T_γ are identical, which is a contradiction. Therefore T_γ can only contain one non-null row. In other words, only k^2 inputs can possibly reach a given state γ querying v_7. Since there are k^4 possible inputs to B, it follows that B must contain at least k^2 states that query v_7. □

5 Conclusion

The approach to separating **L** and **NL** from **P** by proving branching program size lower bounds for the Tree Evaluation Problem has seen some partial success, in the case of trees of height 3, however it has proven to be a difficult problem to extend these results to trees of height 4 and beyond. Nevertheless, we have been able to prove $\Omega(k^4)$ branching program size lower bounds for $FT_2^4(k)$ in various restricted models, and examining a variety of different proof techniques that may prove to be useful in attacking more general problems surrounding the TEP. One of the next steps in this line of research may be to extend these lower bounds to general branching programs solving the height 4 TEP. This would be a major achievement in itself, and may provide some insight into how one might extend these results further to prove strong space lower bounds for the general Tree Evaluation Problem for trees of arbitrary height, bringing us a step closer to settling the question of whether **L** equals **P**.

References

[1] S. M. Chan, J. Cook, S. Cook, P. Nguyen, and D. Wehr. New results for tree evaluation. Work in progress. Dec. 2010.

[2] S. Cook, P. McKenzie, D. Wehr, M. Braverman, and R. Santhanam. Pebbles and branching programs for tree evaluation. *ACM Trans. Comput. Theory*, 3(2):4:1–4:43, Jan. 2012. ISSN: 1942-3454. DOI: 10.1145/2077336.2077337.

[3] D. Liu. Pebbling arguments for tree evaluation. *CoRR*, abs/1311.0293, 2013.