Read-Once Branching Programs for Tree Evaluation Problems

Kazuo Iwama[1] **and Atsuki Nagao**[2]

1 **Graduate School of Informatics, Kyoto University, Kyoto, Japan**
iwama@kuis.kyoto-u.ac.jp

2 **Graduate School of Informatics, Kyoto University, Kyoto, Japan**
a-nagao@kuis.kyoto-u.ac.jp

Abstract

Toward the ultimate goal of separating **L** and **P**, Cook, McKenzie, Wehr, Braverman and Santhanam introduced the *tree evaluation problem* (*TEP*). For fixed $h, k > 0$, $FT_h(k)$ is given as a complete, rooted binary tree of height h, in which each internal node is associated with a function from $[k]^2$ to $[k]$, and each leaf node with a number in $[k]$. The value of an internal node v is defined naturally, i.e., if it has a function f and the values of its two child nodes are a and b, then the value of v is $f(a, b)$. Our task is to compute the value of the root node by sequentially executing this function evaluation in a bottom-up fashion. The problem is obviously in **P** and if we could prove that any branching program solving $FT_h(k)$ needs at least $k^{r(h)}$ states for any unbounded function r, then this problem is not in **L**, thus achieving our goal. The above authors introduced a restriction called *thrifty* against the structure of BP's (i,e., against the algorithm for solving the problem) and proved that any thrifty BP needs $\Omega(k^h)$ states. This paper proves a similar lower bound for *read-once* branching programs, which allows us to get rid of the restriction on the order of nodes read by the BP that is the nature of the thrifty restriction.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes

Keywords and phrases Lower bounds, Branching Programs, Read-Once Branching Programs, Space Complexity, Combinatorics

Digital Object Identifier 10.4230/LIPIcs.STACS.2014.409

1 Introduction

Settling the **P** vs. **NP** question is obviously the biggest goal of theoretical computer science, but the fact is that almost nothing is known for separation of other complexity classes, either. For example, separation of **L** (= Log space) and **P**, which has been much less popular than **P** vs. **NP**, should be equally important to make clear the whole view of complexity classes. To this end, Cook, McKenzie, Wehr, Braverman and Santhanam introduced a simple but very general problem called the *tree evaluation problem* (*TEP*) [3]. For fixed $h, k > 0$, $FT_h(k)$ is given as a complete, rooted binary tree of height h in which each internal node is associated with a function from $[k]^2$ to $[k]$, and each leaf node with a number in $[k]$. The value of an internal node v is defined naturally, i.e., if it has a function f and the values of its two child nodes are a and b, then the value of v is $f(a, b)$. Our task is to compute the value of the root node by sequentially executing this function evaluation in a bottom-up fashion. Note that the original definition in [3] is based on a d-ary tree. In this paper, we only consider a binary tree for our TEP.

Our computation model is branching programs (BP's) that are sometimes more useful to discuss complexity bounds rather than Turing machines (TM's) especially for problems having relatively low complexities like TEP. It is known that the size of a branching program (the number of its states) and the space of a TM are closely related, namely a lower bound $s(n)$ for BP's size implies a lower bound $\Theta(\log(s(n)))$ for TM's space. It then turns out that if we can prove that any BP solving $FT_h(k)$ needs at least $k^{r(h)}$ states for any unbounded function r, then this problem is not in $\mathbf{L}$. Since it is obviously in $\mathbf{P}$, we would be able to separate $\mathbf{L}$ and $\mathbf{P}$. For details of these observations, see [3].

It is not hard to construct a branching program that computes $FT_h(k)$ of size $O(k^h)$ (see Fig. 3 given later) and this construction strongly seems optimal. As mentioned above, we only need a much more moderate bound, $k^{r(h)}$, and that is the natural reason why we think this problem would fit our goal. In fact, [3] proves, by using the black pebbling game [10][2], that if our BP's satisfy a certain property, called the *thrifty* restriction, then we do need $\Omega(k^h)$ states. The thrifty restriction roughly means that when the BP reads an internal node v (actually reads its associated function), it has already read all the values of the v's subtree. Thus this algorithmic restriction strongly restricts the order of tree nodes that are read by the BP. (The thrifty restriction also applies to nondeterministic BP's, in which case its meaning is more subtle.) The authors claim that this restriction is "natural," but we can of course think of different kind of BP's that guess (read) function values first and then check the leaf values if they actually realize the function values. In fact our lower bound proof gets messy in this case.

Recall that we have another popular restriction type of BP's, namely the *read-once* restriction, where a read-once BP reads each input value at most once in any computation path. In fact the above $O(k^h)$ construction is not only thrifty but also read-once and [14] proves that if our BP is both thrifty and read-once, then this explicit construction with $(k+1)^h - k$ states is absolutely optimum. Now the natural question is what if we impose only the read-once restriction.

Our contribution. It is shown that if a read-once BP B solves $FT_h(k)$, then B needs $\Omega(k^h)$ states, thus proving a lower bound on the size of read-once BP's similar to that of thrifty BP's. Actually B needs to be read-once only for states reading leaf values, i.e., the result holds for even less restricted BP's such that in every computation path, if the last leaf-reading state s reads a leaf node v, any state appearing before s on the path does not read v. Note that there is no restriction at all on states reading internal nodes (associated with functions). Furthermore, since our main lemma bounds the number of only leaf-reading states, we do not have to care about the number of these non-leaf-reading states.

Since there are no restrictions on the order of nodes visited by the BP any longer, there is no obvious way of directly using the pebbling game for lower bound proof. Instead, we use a similar notion from a slightly different angle, namely we use what we call a *cut configuration*, a set of the values of $h-1$ nodes that "cut" paths between leaf nodes and the root of the given $FT_h(k)$. The key lemma (Lemma 5) is that if a last leaf-reading state accepts two or more inputs having different cut configurations, then the function part in the inputs is severely restricted, which means the number of different inputs whose paths go through this state is very small. Thus there must be a lot of inputs whose function part does not have this restriction, and we can imply that those inputs have only one cut configuration for any of the last leaf-reading states. For such a fixed function part, the number of inputs having that cut configuration for the last leaf-reading state is easily bounded from above. Thus follows the lower bound for the number of such states. Of course there should still exist a big gap between this class of BP's and general ones, but at least we can get rid of the issue of node orders visited by BP's, which was quite annoying for the attempt of generalising our lower bound proofs.

Our proof depends on another important lemma (Lemma 3) that relates the number of leaf-reading states and the number of states reading second-leaves (the leaves located at height $h-1$). This lemma holds for general BP's and gives us a by-product. Namely, as shown in Section 4, we can obtain a k^3 lower bound for general BP's for the height-3 TEP, which is the same as [3] but with a simpler proof.

Related Work. Other than the lower bounds for thrifty BP's, [3] includes several important results, for instance, it gives a lower bound, k^3, for *unrestricted* BP's solving $FT_3(k)$, which is tight up to the constant factor. This is still the best lower bound for general BP's solving TEP. [7] studies mainly nondeterministic BP's for TEP. Its main result is that "bitwise-independent" thrifty nondeterministic BP's for TEP have at least $\frac{1}{2}k^{h/2}$ states, which is tight against the upper bounds shown in [3]. Their main technique is so-called the entropy method developed in [6]. See [3] for several other attempts trying to separate relatively low complexity classes. For instance [4] studies the complexity of BP's solving GEN (known to be P-complete) that asks a certain kind of reachability to a target element repeatedly using a binary operation.

Studies on branching programs have been quite popular since their introduction by Masek [8], and there is a large literature if it is restricted to studies on their size lower bounds (the following is only a small fraction): The best general deterministic lower bound is still $\Omega(n^2/(\log n)^2)$, which was proved almost half a century ago by Nečiporuk [9]. Note that the above lower bound for $FT_3(k)$ is $\Omega(n^{3/2}/(\log n)^{5/2})$ in terms of the binary input length. (For a general d-ary TEP, [3] obtains a stronger $\Omega(n^2/(\log(n))^2)$ lower bound applying the Nečiporuk method.) Against read-once branching programs, we have much better lower bounds. In 1984, Žák [15] first obtained a super-polynomial lower bound, $\Omega(2^{\sqrt{n}-\log n})$, for the half-clique function, which was improved to more than $2^{n/3-o(n)}$ by Wegener [13]. For the triangle parity function, Ajtai [1] gave a 2^{cn} lower bound and the value of c was later improved by Simon(1993) [12]. Jukna [5] relaxed the read-once restriction to the k-read-once restriction (i.e., all variables except k ones are read-once). He obtained a lower bound of $2^{\Omega((\frac{n}{k})^{1/2})}$ for $k = O(n/\log n)$ and this is extended by Žak [11] into a hierarchy theorem based on this value k.

2 Preliminaries

For the Tree Evaluation Problem (TEP), $FT_h(k)$, we are given a complete binary tree T^h of height h with nodes 1 through 2^h-1 (see Fig. 1 for $h = 3$). Each internal node $1 \le i \le 2^{h-1}-1$ is associated with some explicit function $f_i : [k]^2 \mapsto [k]$, where $[k] = \{1, 2, \dots, k\}$. Each leaf node j ($2^{h-1} \le j \le 2^h - 1$) is associated with a number in $[k]$. Our task is to compute the value of the function f_1 at the root node in the natural way: Suppose that we have inputs $f_1, f_2, f_3, a_4, a_5, a_6, a_7$ for the tree of Fig. 1. Then the value we want to obtain is

$$f_1(f_2(a_4, a_5), f_3(a_6, a_7)).$$

Note that each f_i is given as an explicit sequence of values, e.g., $f_i(1,1)$, $f_i(1,2)$, $f_i(1,3)$, $f_i(2,1)$, $f_i(2,2)$, $f_i(2,3)$, $f_i(3,1)$, $f_i(3,2)$, $f_i(3,3)$ for $k = 3$. In some cases, it is convenient to use a $k \times k$ matrix instead of the above sequence. For instance Fig. 2 shows an example of f_1,f_2,f_3 for $h = 3$. Now if (a_4, a_5, a_6, a_7)=$(3, 3, 1, 2)$, then the solution for this inputs is $f_1(f_2(3,3), f_3(1,2))$= $f_1(3,2)$= 1. In our lower bound proof, the nodes located at height $h-1$ (parents of leaves) play an important role. We call them *second-leaves.*

Our computation model is a (deterministic) *branching program* (*BP*) B, which is a directed, rooted, acyclic graph. Its vertices are called *states* including a unique *initial state* and k *sink*

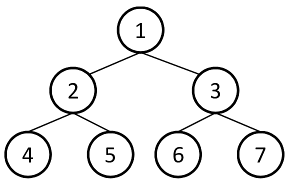

Figure 1 $FT_3(3)$.

$f_1(x,y)$

$y \backslash x$	1	2	3
1	3	1	2
2	2	2	1
3	3	3	3

$f_2(x,y)$

$y \backslash x$	1	2	3
1	2	1	2
2	3	2	1
3	1	2	3

$f_3(x,y)$

$y \backslash x$	1	2	3
1	2	1	3
2	2	2	3
3	1	1	1

Figure 2 Example of f_1, f_2, f_3.

states. Each non-sink state (or simply a state if no confusion would arise) has k outgoing edges labelled by 1 through k, while each sink state has no outgoing edges. Each state has a *label* of the form (i_1, i_2, i_3) or j, where $1 \le i_1 \le 2^{h-1}-1$, $1 \le i_2, i_3 \le k$ and $2^{h-1} \le j \le 2^h - 1$. Each sink state has a label l where $1 \le l \le k$. A BP B computes the solution of TEP in the following way. Suppose that our input is $I = (f_1(1,1), f_1(1,2), \ldots, f_{2^{h-1}-1}(k,k), a_{2^{h-1}}, \ldots, a_{2^h-1})$. Then its *computation path*, P, for input I is defined as follows. P starts from the initial state. If P is now at a state with label (i_1, i_2, i_3), then P is extended by the edge labelled by $f_{i_i}(i_2, i_3)$. If P is at a state with label j, then it is extended by the edge labelled by a_j. We often say that B "reads" the input attached to the node i_1 (a non-leaf node) or j (a leaf) and branches due to its value between 1 and k. P ends with some sink state; if its label is l, then the outcome of the computation is l. If this outcome is equal to the correct solution for all possible inputs I, then we say B *solves* $FT_h(k)$.

Fig. 3 shows an example of a BP that solves $FT_3(3)$. The computation path for the input previously given (f_1, f_2, f_3 in Fig. 2, and (a_4, a_5, a_6, a_7) =$(3,3,1,2)$) is given by a thick line. A BP is called *read-once* if all paths from the root to sinks do not have two or more same labels. The BP in Fig. 3 is read-once.

Our lower bound proof is based on the following simple idea: Suppose that A ($|A| = m_1$) is a carefully selected subset of all the possible inputs for $FT_h(k)$. Let B be any (read-once) BP that solves $FT_h(k)$. Then our proof says that we can always select a set S of states such that each computation path corresponding to each input in A goes through some state in S and any state in S accepts computation paths of at most m_2 inputs in A, concluding that $|S|$ is at least m_1/m_2. To introduce such an input set A, we consider the following constraint for functions f_i: Suppose that

$$X = \begin{bmatrix} \alpha_{11} & \alpha_{12} & \cdots & \alpha_{1k} \\ \alpha_{21} & \ddots & \cdots & \alpha_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_{k1} & \alpha_{12} & \cdots & \alpha_{kk} \end{bmatrix}$$

is the matrix representation of f_i. Then it has to satisfy the following three constraints: (i) $\alpha_{11} \ldots \alpha_{1k}$ (= the first row) is a permutation of $(1, \ldots, k)$ (ii) $\alpha_{11} \ldots \alpha_{k1}$ (= the first column) is a permutation of $(1, \ldots, k)$ (iii) For $\forall j \ge 2$, $\alpha_{j1} \ldots \alpha_{jk}$ is a permutation that can be written as $\delta^l(\alpha_{11} \ldots \alpha_{1k})$ for some $1 \le l \le k$ where δ is the cyclic permutation

$$\delta = \begin{pmatrix} 1 & 2 & \cdots & k-1 & k \\ 2 & 3 & \cdots & k & 1 \end{pmatrix}$$

and δ^l is a composition of l δ's. Thus each row is a permutation, and it is not hard to see that each column is also a permutation. X is fixed by determining its first row and the first column, and hence there are $k!(k-1)!$ different f_i's. Let F be the class of functions satisfying these constraints. In this paper, we assume that our function f_i is always selected

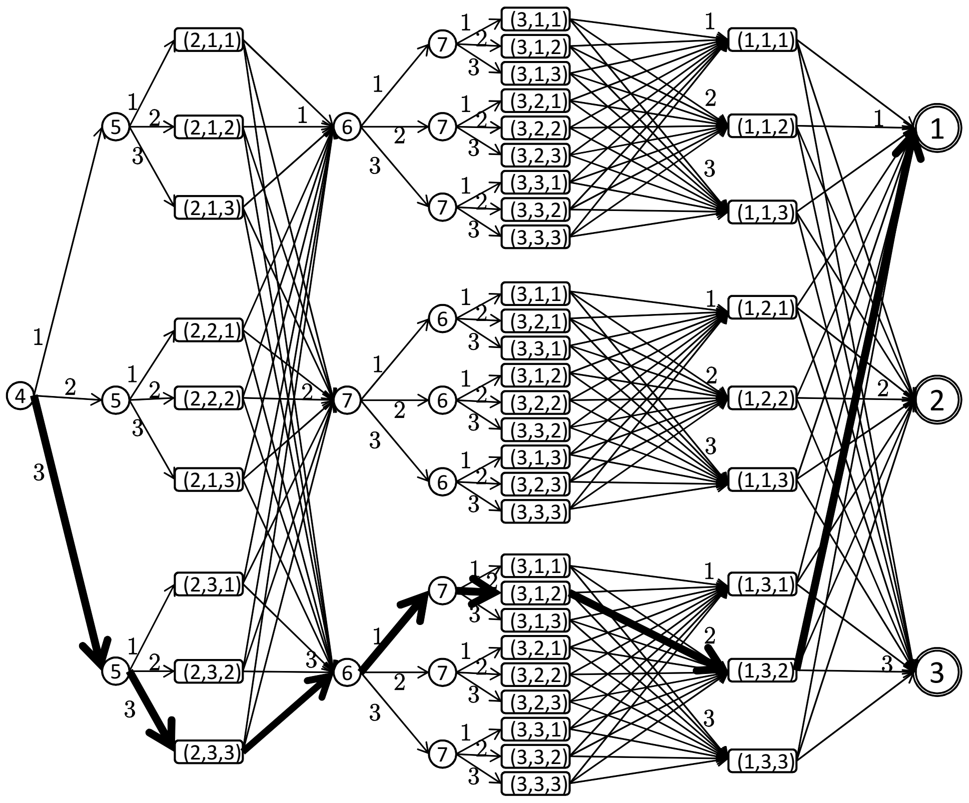

Figure 3 An example of a read-once branching program solving $FT_3(3)$.

from F unless otherwise stated (but of course, our BP's must give correct solutions for all inputs). Now here are easy but important lemmas.

▶ **Lemma 1.** *Suppose that two inputs I and I' (their function parts satisfy the constraint) are exactly the same except only one leaf value at node j. Then the final value of $FT_h(k)$ is different between I and I'.*

Proof. Suppose that the final value is the same and consider the path from the root to j. Since the root value is the same and the leaf value is different, there must be a node i on the path such that the value of i is the same but the value of i's next node i' on the path is different, say, a in I and a' in I'. Let i'' be the sibling of i' (both i' and i'' have i as their parent). Then the value of i'' is the same, say b, in both I and I'. Thus we have $f_i(a,b) = f_i(a',b)$ for $a \neq a'$, which contradicts that $f_i \in F$. ◀

▶ **Lemma 2.** *Suppose that a BP B solves $FT_h(k)$. Then (1) for any internal node i of $FT_h(k)$ and for any $a, b \in [k]$, there must be a state whose label is (i, a, b) in B. (2) If P is a legal computation path, then for any leaf node j, P includes a state that reads j.*

Proof. For (2), suppose that P corresponds to input I and it does not read j. Then consider another input I' which is different from I only in j. Then B obviously outputs the same value for I and I', contradicting the previous lemma. (1) is proved similarly by considering two inputs I and I' that differ only in $f_i(a,b)$ and such that both inputs actually use $f_i(a,b)$ (meaning the values of i's two children are a and b under I and I'). Note that if I satisfies

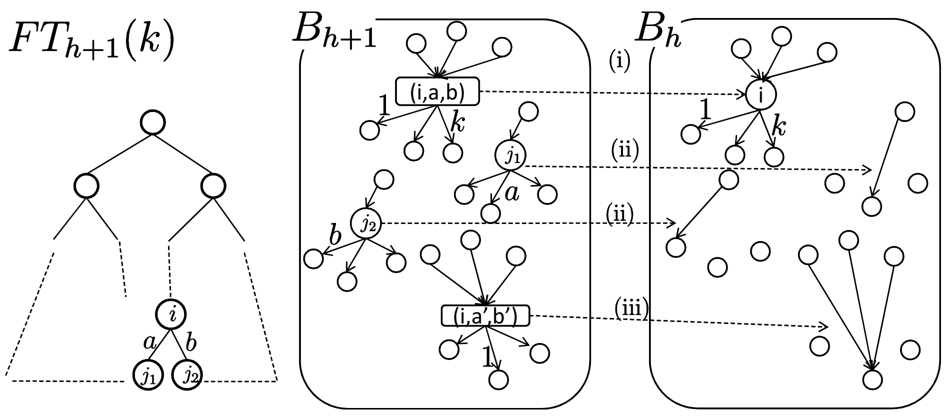

Figure 4 Modification rules(i), (ii) and (iii).

the restriction, then I' does not. Now one can see, exactly as in the proof of the previous lemma, that the final value is different between I and I', but B outputs the same value, a contradiction. ◀

The next lemma (hinted by Th. 5.8 and Th. 5.9 of [3]) relates the number of states reading leaf nodes and the number of state reading second-leaf nodes. By this lemma, we can increase the degree of k by one in the lower bound given in the next section. Note that this lemma holds for general BP's (and see Sec. 4 for its by-product).

▶ **Lemma 3.** *For $h \geq 1$, if there is a BP B_{h+1} solving $FT_{h+1}(k)$ such that the number of states that read second-leaf nodes is n, then there is a BP B_h solving $FT_h(k)$ such that the number of states that read leaf nodes is at most n/k^2. Furthermore, if B_{h+1} is read-once, so is B_h, also.*

Proof. we construct B_h from the given B_{h+1} as follows. Let i be a second-leaf node of $FT_{h+1}(k)$ and (a, b) is a pair of inputs to f_i such that the number of states in B_{h+1} that read $f_i(a, b)$ is less than or equal to the number of states reading $f_i(a', b')$ for any (a', b'). Let m be the number of such state s reading $f_i(a, b)$. By Lemma 2, there is at least one state that reads $f_i(a, b)$ for any $(a, b) \in [k] \times [k]$. So, m is at most $(1/k^2)\times$(the number of states that read f_i). Now we make the following modification against B_{h+1} (see Fig. 4). The basic idea is that we fix the values of the two child (leaf) nodes of i to a and b. Then i looks like a leaf node of $FT_h(k)$ and among the states in B_{h+1} that read i, only $1/k^2$ ones survive by the following construction. This holds for any i and hence the lemma holds. (i) Change the label of the above m states from (i, a, b) to i. (Namely this state reads a leaf node of $FT_h(k)$.) (ii) Suppose that j_1 and j_2 are the two leaf nodes whose parent is i. Then we remove all the states q of B_{h+1} that read j_1 (j_2, respectively) by connecting q's incoming edges to the state to which the edge from q labelled by a (b, respectively) goes. (iii) We remove all the state q of B_{h+1} that read $f_i(a', b')$, $((a', b') \neq (a, b))$, by connecting q's incoming edges to the state to which the edge from q labelled by 1 goes (this "1" is not important or it may be any number in $[k]$).

We repeat this change for all second-leaf nodes of $FT_{h+1}(k)$, obtaining B_h. We omit the proof that this construction is correct, since it is almost obvious from the construction. ◀

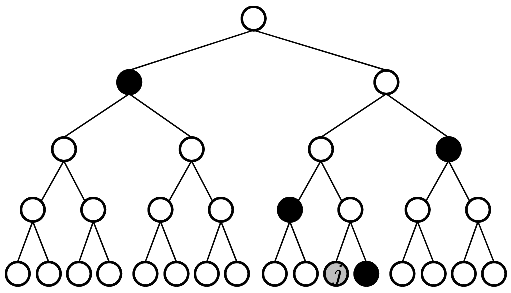

Figure 5 Involved nodes in $CC(I, j)$.

3 Lower Bounds

In this section we obtain a lower bound for the number of states that read leaf nodes of $FT_h(k)$. Then combining it with Lemma 3, we obtain a better lower bound for the number of states that read second-leaf nodes of $FT_h(k)$. Recall that our input satisfies the constraint (its functions belong to F) and all BPs in this section are read-once. Let B be a BP that solves $FT_h(k)$ and P be its arbitrary computation path. (To avoid confusion, we sometimes say that P is a *legal* computation path to emphasise that P is based on an input whose function part satisfies the constraint.) Then by Lemma 2, P reads all leaf values (for any leaf j, there is a state in P that reads j). Let q be the last state on P that reads a leaf value, i.e., there is no state after q on P that reads a leaf. Since B is read-once, q is also the last leaf-reading state on any other legal computation path that includes q. Thus, as far as we are looking at only legal computation paths, we can define a *last leaf-reading state* without specifying a computation path.

Now we define our key tool in the proof in this section. Suppose that $I = (f_1, \dots, f_{2^{h-1}-1}, a_{2^{h-1}}, \dots, a_{2^h-1})$ is currently associated with $FT_h(k)$ and let j be a leaf. Then the *cut configuration* (CC) for I with respect j, denoted as $CC(I, j)$, is defined as follows.

$$CC(I, j) = (a_1, a_2, \dots, a_{h-1})$$

where, (i) a_1 is the value of j's sibling and (ii) if a_i, $1 \le i \le h-2$, is the value of node x, a_{i+1} is the value of the sibling of x's parent (see Fig. 5). Suppose that we know functions f_1 to $f_{2^{h-1}-1}$. Then if we further know these $h-1$ values as well as the value of j, then we can compute the solution (= the value of node 1). In fact, it is well-known that we can compute the solution in such a way that we need at most $(h-1)\lceil \log k \rceil$ memory space at any stage of its computation (by recursively obtaining the values of a_1, a_2, and so on, in this order first, then the associated function values from bottom to top). What will be done in the rest of this section is to count the number of legal inputs with a certain restriction on its CC that go through a last leaf-reading node. Our first lemma is an upper bound on the number of inputs having a single CC.

▶ **Lemma 4.** *Fix functions $f_1, \dots, f_{2^{h-1}-1}$, an arbitrary leaf node, j, and an arbitrary $(a_1, \dots, a_{h-1})$, $a_i \in [k]$. Then the number of leaf values whose CC with respect to j is $(a_1, \dots, a_{h-1})$ is at most $k^{2^{h-1}-h+1}$*

Proof. Let T_v be a subtree of $FT_h(k)$ whose root is a node v at height i of $FT_h(k)$. Let $v_1, \dots, v_{2^{i-1}}$ (we used a simplified numbering) be the leaf nodes of T_v, and $g(v_1, \dots, v_{2^{i-1}})$ be the value of v. We first calculate the number of different leaf values $(b_1, \dots, b_{2^{i-1}})$ such that $g(b_1,\ldots,b_{2^{i-1}}) = a$ for a fixed $a \in [k]$. For fixed $b_1,\ldots,b_{2^{i-1}-1}$, $g(b_1,\ldots,b_{2^{i-1}-1},x)$ is a function from $[k]$ to $[k]$ (denoted by $g'(x)$). By lemma 1, $g'(x_1) \neq g'(x_2)$ if $x_1 \neq x_2$, in other words, g' is a bijection. Hence, for any $a \in [k]$, value $b \in [k]$ such that $g'(b) = a$ is fixed. Since this holds for any $b_1,\ldots,b_{2^{i-1}-1}$, the number of leaf values $b_1,\ldots,b_{2^{i-1}}$ such that $g(b_1,\ldots,b_{2^{i-1}}) = a$ is $k^{2^{i-1}-1}$.

Now we calculate the number N of leaf values such that their CC with respect to leaf j is $(a_1,\ldots,a_{h-1})$. Let the node taking value a_i be v_i. See Fig. 5 again. First, note that node j can take any of the k values. Next, the value of node v_1 is fixed to a_1; it takes only one value. Since node v_2 is a top node of a subtree with height 2, its leaf nodes can take $k^{2^{2-1}-1} = k$ different values by the above fact. Similarly, v_3's leaf nodes can take $k^{2^{3-1}-1} = k^3$ different values and so on. Therefore,

$$\begin{aligned}N &= k \cdot 1 \cdot k \cdot k^3 \cdot \cdots \cdot k^{2^{h-2}-1}\\ &= k \cdot k^{2^1+2^2+\cdots+2^{h-2}-(h-2)}\\ &= k \cdot k^{2^{h-1}-2-(h-2)} = k^{2^{h-1}-(h-1)}\end{aligned}$$

◀

Now we are ready to prove our main lemma. We divide an input $(f_1, \ldots, f_{2^{h-1}-1}, a_1, \ldots, a_{2^{h-1}})$ into two parts, the function part (f-part) $\mathbf{f} = (f_1,\ldots,f_{2^{h-1}-1})$ and the leaf value part (l-part) $\mathbf{l} = (a_1,\ldots,a_{2^{h-1}})$. Let B be any (read-once) BP solving $FT_h(k)$ and s be any last leaf-reading state, reading a leaf j. Let c_1 and c_2 be two different CC's with respect to j whose inputs have the same f-part, and $G(c_1,c_2,s)$ be the set of such f-parts (i.e., if $\mathbf{f} \in G(c_1,c_2,s)$, then there are two l-parts $\mathbf{a}_1$ and $\mathbf{a}_2$ such that $(\mathbf{f},\mathbf{a}_1)$ and $(\mathbf{f},\mathbf{a}_2)$ have CC's c_1 and c_2, respectively). Note that there are $(k!(k-1)!)^{2^{h-1}-1}$ different f-parts in total and we denote this number by N_0.

▶ **Lemma 5.** *Suppose that k is a prime number. Then for any c_1, c_2, s, $|G(c_1,c_2,s)| \leq N_0\frac{k}{k!}$*

Proof. Let $c_1 = (a_1,\ldots,a_{h-1})$, $c_2 = (b_1,\ldots,b_{h-1})$, and let $v_1,\ldots,v_{h-1}$ be the nodes providing these two CC values. Also let $g_1,\ldots.g_{h-1}$ be the functions associated with $v_1,\ldots,v_{h-1}$ producing both c_1 and c_2 (for different leaf values). Recall that s is a last leaf-reading state (reading node j) and our BP is read-once. Hence, the value of j is first read at s, which means two computation paths realizing c_1 and c_2 are not affected by the value of j until the state s. Furthermore, these paths must go to the same sink-node for each fixed value a of the node j, because after the state s our BP reads only function values being the same for c_1 and c_2.

$$g_1(a_1, g_2(a_2,\ldots,g_{h-1}(a_{h-1},a)\ldots)) = g_1(b_1, g_2(b_2,\ldots,g_{h-1}(b_{h-1},a)\ldots)) \tag{1}$$

Note that for a fixed a_{h-1}, $g_{h-1}(a_{h-1},a)$ is a bijection form $[k]$ to $[k]$ and can be represented as a permutation

$$\delta_{h-1} = \begin{pmatrix} 1 & 2 & \cdots & k\\ \alpha_1 & \alpha_2 & \cdots & \alpha_k\end{pmatrix}$$

where $\alpha_1 \ldots \alpha_k$ are the (a_{h-1})th row of the matrix of g_{h-1}. Using similar representations for g_1 to g_{h-2}, (1) can be written as

$$\delta_1\delta_2\ldots\delta_{h-1} = \delta'_1\delta'_2\ldots\delta'_{h-1} \tag{2}$$

where δ_i is the (a_i)th row of (the matrix of) g_i and δ'_i is the (b_i)th row of g_i. Due to our constraint for g_i, we can write $\delta'_i = \delta^{l_i}\delta_i$ for some $0 \le l_i \le k-1$ (recall that δ is the cyclic permutation).

Now (2) can be rewritten as

$$\delta_1\delta_2 \dots \delta_{h-1} = \delta^{l_1}\delta_1\delta^{l_2}\delta_2 \dots \delta^{l_{h-1}}\delta_{h-1}$$

Suppose that a_i and b_i are the first different values in the two CC's (i,e., $a_1 = b_1, \dots a_{i-1} = b_{i-1}$). Then $l_1 = l_2 = \cdots = l_{i-1} = 0$ and hence

$$\begin{aligned}
& \delta_1\delta_2 \dots \delta_{h-2}\delta_{h-1} = \delta^{l_1}\delta_1\delta^{l_2}\delta_2 \dots \delta_{h-2}\delta^{l_{h-1}}\delta_{h-1} \\
\Leftrightarrow \quad & \delta_i\delta_{i+1} \dots \delta_{h-2} = \delta^{l_i}\delta_i\delta^{l_{i+1}}\delta_{i+1} \dots \delta^{l_{h-1}} \\
\Leftrightarrow \quad & \delta_i = \delta^{l_i}\delta_i\delta^{l_{i+1}}\delta_{i+1} \dots \delta^{l_{h-1}}\delta^{-1}_{h-2} \dots \delta^{-1}_{i+1} \\
\Leftrightarrow \quad & \delta^* = \delta_i^{-1}\delta^c\delta_i \qquad (3)
\end{aligned}$$

where $\delta^* = (\delta^{l_{i+1}} \dots \delta^{-1}_{i+1})^{-1}$ and $\delta^c = \delta^{l_i}$.

Now let

$$\delta_i = \begin{pmatrix} \alpha_1 & \alpha_2 & \cdots & \alpha_k \\ 1 & 2 & \cdots & k \end{pmatrix} \text{ and } \delta^* = \begin{pmatrix} 1 & 2 & \cdots & k \\ \beta_1 & \beta_2 & \cdots & \beta_k \end{pmatrix},$$

Then (3) can be written as

$$\begin{aligned}
& \begin{pmatrix} 1 & 2 & \cdots & k \\ \beta_1 & \beta_2 & \cdots & \beta_k \end{pmatrix} \\
= & \begin{pmatrix} 1 & 2 & \cdots & k \\ \alpha_1 & \alpha_2 & \cdots & \alpha_k \end{pmatrix} \begin{pmatrix} 1 & 2 & \cdots & k \\ 1+c & 2+c & \cdots & k+c \end{pmatrix} \begin{pmatrix} \alpha_1 & \alpha_2 & \cdots & \alpha_k \\ 1 & 2 & \cdots & k \end{pmatrix} \\
= & \begin{pmatrix} \alpha_1 & \alpha_2 & \cdots & \alpha_k \\ \alpha_{1+c} & \alpha_{2+c} & \cdots & \alpha_{k+c} \end{pmatrix}
\end{aligned}$$

where $1+c, \dots, k+c$ are all MOD k. Note that δ^* and δ^c are conjugate and therefore their cycle structures are the same. Since δ is the cyclic permutation, δ^c has a single cycle and therefore δ^* also has a single cycle.

It then turns out that if we fix α_1 to $d \in [k]$, then by the left hand side, d should be mapped to β_d, meaning $\alpha_{1+c} = \beta_d$. Then again by the left hand side, β_d should be mapped to β_{β_d}, meaning $\alpha_{1+2c} = \beta_{\beta_d}$, and so on. Namely once α_1 is fixed, all the other α_i's are sequentially fixed one after another or δ_i itself is fixed. Since k is prime, this sequence of value transfer does not end in the middle. Thus we have at most k different possibilities for δ_i (due to k different values for α_1). Recall that δ_i can take $k!$ different permutation in general. (The whole matrix is determined by fixing the first row and the first column, but it should be noted that it is also determined by fixing any row and then any column). But now there are only k possibilities as shown above. So the number of different g_i is at most $N_0\frac{k}{k!}$ for each combination of other $h-2$ functions. Note that s may have another CC, say c_3, other than c_1 and c_2. Then we have another restriction for functions, which results in even a smaller number of possible functions. Thus it is enough to consider only the case that the state s has two different CC's, for the upper bound of the lemma. ◀

Now we imply a contradiction if the number of leaf-reading states is less than k^{h-1}, through the following two lemmas.

▶ **Lemma 6.** *Suppose that $s_1, s_2, \ldots, s_{k^{h-1}-1}$ are $k^{h-1}-1$ different last leaf-reading states of the BP B. Then there is an f-part $\mathbf{f}_0$ such that for any s_i $(1 \le i \le k^{h-1}-1)$, if inputs $(\mathbf{f}_0, \mathbf{a}_1)$ and $(\mathbf{f}_0, \mathbf{a}_2)$ go through s_i, their CC's are the same.*

Proof. We count the number of f-parts $\mathbf{f}$ that do not satisfy the condition of the lemma. By Lemma 5, there are $N_0 \cdot \frac{k}{k!}$ such $\mathbf{f}$ for each combination of state s_i, CC c_1 and CC c_2. Note that we have $k^{h-1}-1$ s_i's and there are at most k^{h-1} different CC's in general. Therefore the number of such $\mathbf{f}$'s is at most

$$N_0 \cdot \frac{k}{k!} \cdot (k^{h-1}-1) \cdot (k^{h-1})^2 \le N_0 k^{3h-2}/k!,$$

which is strictly less than N_0 for a large (prime) k. Thus an $\mathbf{f}_0$ of the lemma must exists. ◀

▶ **Lemma 7.** *B needs at least k^{h-1} last leaf-reading states.*

Proof. Suppose B has at most $k^{h-1}-1$ last leaf-reading states. Then by Lemma 6, there is an f-part $\mathbf{f}_0$ such that inputs having this $\mathbf{f}_0$ as their f-part show at most one CC for any of these last leaf-reading states. However, Lemma 4 shows each state accepts at most $k^{2^{h-1}-h+1}$ l-parts, meaning these $k^{h-1}-1$ states accept at most $k^{2^{h-1}-h+1} \cdot (k^{h-1}-1) < k^{2^{h-1}}$ l-values in total. Since each of the all $k^{2^{h-1}}$ l-values must be accepted by some last leaf-reading state, this is a contradiction. ◀

▶ **Theorem 8.** *Any read-once BP B_h solving $FT_h(k)$ needs at least k^h states.*

Proof. The contraposition of Lemma 4 claims that if the number of leaf-reading states of B_h is at least m, then the number of second-leaf-reading states of B_{h+1} is at least $k^2 m$. Now the theorem is immediate from Lemma 7. ◀

4 General Branching Programs for Height-3 TEP

Recall that Lemma 3 holds for general BPs. Also it turns out that the TEP of height two is somewhat special. Thus we can obtain the following general lower bound for BPs for the height-3 TEP with a simpler proof than that of [3].

▶ **Theorem 9.** *Any (general) BP solving $FT_3(k)$ needs at least k^3 states.*

Proof. Due to Lemma 3, it suffices to show that any BP solving $FT_2(k)$ needs at least k leaf-reading states. In the following, we show it needs at least $k+1$ leaf-reading states, which is optimal by a construction similar to that of Fig. 3. Recall that $FT_2(k)$ has three nodes, 1, 2 and 3, where node 1 is associated with a function f_1 and nodes 2 and 3 are leaf nodes. Suppose that we have a BP B that solves $FT_2(k)$ and that has at most k leaf-reading states. We fix f_1 to an arbitrary function in F and then B can be modified to the BP that reads only leaf nodes; we also denote this BP by B.

We give a new label (in addition to the original label of B), a set of pairs (a,b), $1 \le a, b \le k$, to each state and each edge of B by the following rule: (i) The initial node of B has label $\{(a,b) \mid 1 \le a, b \le k\}$, i.e., the set of all possible pairs. (ii) Suppose that a state s has a label S and an edge e from s reads node 2 to get value i. Then the label to the edge e is $\{(i,b) \mid 1 \le b \le k\} \cap S$, namely the (possibly empty) set of pairs in S whose first element is i. Similarly for the case that s reads node 3 (we do the same thing with the second element of the pair). (iii) Suppose that all the edges entering state s already have labels. Then the label of s is the union of the labels of those incoming edges. Now it is easy to see that such labels "describe" an execution of B in the following sense: Suppose that the label of an edge

e includes a pair (a, b). Then the computation path of B goes through this edge e if and only if the values of nodes 2 and 3 are a and b, respectively (and f_1 is the current fixed function).

Now suppose for contradiction that B has at most k states other than k sink states and we look at edges that go to these sink states. Note that the number of all edges is k^2 since each of the k states has k edges. Also note that B has to read both of the two leaf states in its computation path, so at least one edge goes to non-sink states. Consequently the number of the above (going to sink states) edges is at most $k^2 - 1$. Since the total number of pairs is k^2, it is impossible to map all those pairs to the edges in a one-to-one fashion, or one of the following two cases must happen:

1. Some pair (a, b) does not appear in any label of these edges. B obviously does not do a correct computation when the values of nodes 2 and 3 are a and b, respectively.
2. Some edge has two (or more) pairs, say (a, b) and (a', b'). Notice that if the state this edge outgoes from reads node 2, then we have $a = a'$. Then the computation of B is not correct again since the output would be the same if the values of node 2 is the same and the values of node 3 are different (recall that our f_1 is in F). Similarly for the case that the state reads node 3 (then $b = b'$).

Thus we can conclude that such B is not a correct BP. ◀

5 Concluding Remarks

The obvious future work is to remove the read-once restriction. Since our main lemma (Lemma 5) heavily depends on the read-once restriction, we do not have any specific approaches to this ultimate goal at this moment. There are a couple of more reasonable sub-goals: One is to prove that if a BP B is thrifty, then B can be converted to a read-once BP without increasing the number of leaf-reading states drastically, or equivalently, to prove that reading a same leaf node twice or more do not help much in thrifty BP's. Another possibility is to attack the case for $h = 4$. This seems more tractable since we can restrict ourselves to the number of leaf-reading states of BP's for $FT_3(k)$ that have several specific properties as shown in [3]. Also this lower bound will outperform the longstanding one by Nečiporuk [9].

References

1 M. Ajtai, L. Babai, P. Hajnal, J. Komlós, P. Pudlák, V. R odl, E. Szemerédi, and G. Turán. Two lower bounds for branching programs. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pages 30–38. ACM, 1986.

2 S. Cook. An observation on time-storage trade off. *Journal of Computer and System Sciences*, 9(3):308–316, 1974.

3 S. Cook, P. McKenzie, D. Wehr, M. Braverman, and R. Santhanam. Pebbles and branching programs for tree evaluation. *ACM Transactions on Computation Theory (TOCT)*, 3(2):4, 2012.

4 A. Gál and P. McKenzie M. Koucký and. Incremental branching programs. *Theory of Computing Systems*, 43(2):159–184, 2008.

5 S. Jukna and A. Razborov. Neither reading few bits twice nor reading illegally helps much. *Discrete Applied Mathematics*, 85(3):223–238, 1998.

6 S. Jukna and S. Žák. On uncertainty versus size in branching programs. *Theoretical computer science*, 290(3):1851–1867, 2003.

7 B. Komarath and J. Sarma. Pebbling, entropy and branching program size lower bounds. In *30th International Symposium on Theoretical Aspects of Computer Science (STACS)*, page 622, 2013.

8 W. Masek. *A fast algorithm for the string editing problem and decision graph complexity.* PhD thesis, Massachusetts Institute of Technology, 1976.

9 È. Nečiporuk. A boolean function. In *Doklady of the Academy of the USSR*, volume 169, pages 765–766, 1998. English translation in Soviet Mathematics Doklady 7:4, pp.999–1000.

10 M. Paterson and C. Hewitt. Comparative schematology. In *Record of the Project MAC Conference on Concurrent Systems and Parallel Computation*, pages 119–127. ACM, 1970.

11 P. Savický and S. Žák. A read-once lower bound and a (1,+ k)-hierarchy for branching programs. *Theoretical Computer Science*, 238(1):347–362, 2000.

12 J. Simon and M. Szegedy. A new lower bound theorem for read-only-once branching programs and its applications. *Advances in Computational Complexity Theory*, 13:183–193, 1993.

13 I. Wegener. On the complexity of branching programs and decision trees for clique functions. *Journal of the ACM (JACM)*, 35(2):461–471, 1988.

14 D. Wehr. Exact size of smallest minimum-depth deterministic bps solving the tree evaluation problem. Unpublished. http://www.cs.toronto.edu/~wehr/.

15 S. Žák. An exponential lower bound for one-time-only branching programs. In *Mathematical Foundations of Computer Science 1984*, pages 562–566. Springer, 1984.