JOURNAL OF COMPUTER AND SYSTEM SCIENCES 53, 79–87 (1996)
ARTICLE NO. 0050

New Lower Bounds and Hierarchy Results for Restricted Branching Programs

Detlef Sieling*

FB Informatik, LS II, Universität Dortmund, 44221 Dortmund, Germany

Received August 31, 1993; revised August 14, 1995

In unrestricted branching programs all variables may be tested arbitrarily often on each path. But exponential lower bounds are only known if on each path the number of tests of each variable is bounded. We examine branching programs in which for each path the number of variables that are tested more than once is bounded by k but we do not bound the number of tests of those variables. Using a new lower bound method we can prove that such branching programs become more powerful by increasing k only by 1: For $k \leqslant (1-\epsilon)(n/3)^{(1/3)}/\log^{2/3} n$, where $\epsilon > 0$, we exhibit Boolean functions that can be represented in polynomial size if k variables may be tested more than once on each path, but only in exponential size if $k-1$ variables may be tested more than once on each path. Therefore, we obtain a tight hierarchy.

1. INTRODUCTION

Branching programs are a powerful representation of Boolean functions. We can derive branching programs for some function from non-uniform Turing machines for this function. Lower and upper bounds for branching programs imply lower and upper bounds for the space complexity of non-uniform Turing machines and of any other reasonable model of sequential computation. For this reason branching programs and lower bound methods for branching programs are extensively studied in complexity theory. Branching programs are also used as data structure for Boolean functions. For restricted variants of branching programs efficient algorithms for operations on Boolean functions represented by these branching programs are known. Such data structures are needed in logic synthesis, test pattern generation, verification of VLSI designs and analysis and synthesis of sequential circuits. The knowledge of lower and upper bounds for these variants of branching programs is useful for the estimation of the expressive power of the data structures.

A branching program is a directed acyclic graph with one source node. Sink nodes are labeled by a Boolean constant 0 or 1. Non-sink nodes, also called interior nodes, are labeled by Boolean variables and have two outgoing edges, one labeled by 0 and the other labeled by 1. Each input $a = (a_1, ..., a_n)$ defines a path from the source node to a sink node. In order to obtain this path for the input a we start at the source node. At an interior node labeled by x_i we follow the outgoing edge labeled by 0 if $a_i = 0$, or the outgoing edge labeled by 1 if $a_i = 1$. This is iterated until we reach a sink node. The label of this sink node is the value that the function represented by the branching program takes for the input a.

The best known lower bound for unrestricted branching programs can be obtained by methods due to Nečiporuk [11]. But this bound is only of size $\Omega(n^2 \log^{-2} n)$. Since we are interested in exponential lower bounds, we have to consider restrictions of general branching programs.

If we want to use branching programs as data structure for Boolean functions, we must make sure that as many as possible important Boolean functions can be represented in small size. Furthermore, it is necessary that the operations on Boolean functions can be performed efficiently on the data structure. The most important operations are evaluation, satisfiability, synthesis, and equality. An exhaustive list of operations and of applications is given in Wegener [14]. For the evaluation we have to compute for a data structure representing the function f and an input a the value $f(a)$. Satisfiability is the test whether there is some input a for which the function represented by the data structure takes the value 1. Synthesis is the problem to compute a data structure for $f_1 \circ f_2$, where data structures for f_1 and f_2 and a binary Boolean operation $\circ$ are given. For the equality test we have to decide whether two functions represented by data structures are equal.

For unrestricted branching programs satisfiability is NP-complete and equality is co-NP-complete. Therefore, only restrictions of general branching programs are usable as data structures for Boolean functions. In the following we survey the most important restrictions of branching programs. For ordered binary decision diagrams (OBDDs) an ordering of the variables must be fixed. On each path from the source node to a sink node the variables are tested according to this ordering. This also implies that each variable is tested at most once on each path. OBDDs

* Supported in part by DFG Grant We 1066/7-1.

0022-0000/96 $18.00

are the most popular data structure for Boolean functions because the operations can be performed efficiently on Boolean functions represented by OBDDs (Bryant [5, 6]). Exponential lower bounds for the size of OBDDs for integer multiplication and the hidden weighted bit function HWB (HWB is defined below) are also proved by Bryant [7].

In a read-once branching program (BP1) each variable may be tested at most once on each path from the source node to a sink node. Exponential lower bounds for the size of read-once branching programs can be obtained by cut-and-paste techniques due to Wegener [13] and Žák [15]. Read-once branching programs are not usable as data structure because synthesis is NP-hard. Graph driven binary decision diagrams (Sieling and Wegener [12], Gergov and Meinel [9]) are read-once branching programs for which a generalized variable ordering is given. Graph driven binary decision diagrams have the same expressive power as read-once branching programs and the most important operations can be performed efficiently.

Read-k-times branching programs may contain k tests of each variable on each path. In read-k-times branching programs as well as in unrestricted branching programs null-chains may occur, i.e., paths which are not chosen for any input. This happens if on some path at some node v the 0-edge leaving v is chosen and at some other node v' labeled by the same variable as v the 1-edge leaving v' is chosen. A path that is not a null-chain is called consistent. In non-syntactic read-k-times branching programs the number of tests of each variable is bounded by k only for consistent paths, while in syntactic read-k-times branching programs this number is bounded also for null-chains. An exponential lower bound for syntactic read-k-times branching programs is proved by Borodin, Razborov and Smolensky [4]. It is conjectured that for each $k \geqslant 2$ there are functions which can be represented by read-k-times branching programs of polynomial size but only by read-$(k-1)$-times branching programs of exponential size; this means that read-k-times branching programs of polynomial size form a hierarchy. But for $k \geqslant 3$ no such function is known so far. For non-syntactic read-k-times branching programs no exponential lower bound is known at all.

A decision tree is a branching program for which the underlying graph is a tree; this means each node except the source node has indegree one. We may assume that a decision tree does not contain null-chains: If in a decision tree some variable is tested more than once on some path, all tests but the first one are redundant and can be eliminated. It is well known that decision trees have exponential size even for simple functions like parity. Therefore, decision trees cannot be used as data structure for Boolean functions. Methods for the estimate of the decision tree complexity of Boolean functions are presented in Wegener [13].

We call a branching program leveled if the node set can be partitioned into levels so that each edge leaving a node at level i leads to a node at level $i+1$. If at each level the same variable is tested, the branching program is called oblivious. Exponential lower bounds for oblivious branching programs of linear depth were first proved by Alon and Maass [1].

There are functions that can be represented in a natural way by branching programs in which on each path each variable is tested at most once with the only exception that at the end of each path one variable may be tested for a second time. An example for such a function is the hidden weighted bit function HWB due to Bryant [7]. This function is defined by

$$\mathrm{HWB}_n(x_1, \ldots, x_n) = x_s, \qquad \text{where} \quad s = \sum_{i=1}^{n} x_i.$$

Here we assume that $x_0 = 0$. It is easy to construct a branching program for HWB. This branching program consists of two parts: The top part has $n+1$ sinks numbered from 0 to n and the jth sink is reached if $\sum_{i=1}^{n} x_i = j$. The interior nodes are arranged in n levels numbered from 1 to n. The jth level contains j nodes labeled x_j. The 0-successor of the ith node in level j is the ith node in level $j+1$, the 1-successor is the $(i+1)$th node in level $j+1$. Hence, in this part the variables are tested on each path in the order $x_1, \ldots, x_n$. We obtain the bottom part if we replace the ith sink by a test of x_i and the sink with number 0 by a 0-sink.

This branching program for HWB is neither an OBDD nor a read-once branching program because in the bottom part variables are tested for a second time. It is a read-twice branching program but we do not need the full expressive power of read-twice branching programs because on each path at most one variable is tested twice. It is even possible to construct a read-once branching program of polynomial size for HWB (Sieling and Wegener [12]) but the branching program constructed above resembles much more the definition of HWB. Since there is no OBDD of polynomial size for HWB (Bryant [7]), we see that OBDDs can be made more powerful by allowing one variable to be tested for a second time at the end of each computation path. This leads to the question whether OBDDs in which one variable may be tested for a second time at the end of each path can be made more powerful by allowing two repeated tests at the end of each path. We have to prove a lower bound for OBDDs with one repeated test at the end of each path. Our lower bound technique does not work only for OBDDs with k repeated tests at the end of each path (OBDD_{+k}) but also for read-once branching programs with repeated tests. We can even omit the properties that the repeated tests are performed at the end of each computation path and that the number of repetitions is bounded by two.

Now we define branching programs with k repeated tests ($\mathrm{BP1}_{+k}$). On each path (consistent or not) only k variables may be tested more than once, while all other variables may

be tested at most once. On different paths the sets of variables which may be tested more than once may be different. Since the number of variables which may be tested more than once is also bounded for null-chains, we have a syntactic restriction.

We exhibit functions $f^k = (f_n^k)$ that are variants of the hidden weighted bit function. For these functions we prove an exponential lower bound for the size of each $\mathrm{BP1}_{+(k-1)}$. On the other hand, these functions can be represented in polynomial size if k repeated tests are allowed. Therefore, polynomial size branching programs with k repeated tests form a tight hierarchy. This hierarchy result even holds if k is a function that depends on the input length n and $k \leqslant (1-\varepsilon)(n/3)^{1/3}/\log^{2/3} n$ for some $\varepsilon > 0$. If $k = O(\log^{1-\varepsilon} n)$ for some $\varepsilon > 0$, we also obtain a hierarchy of polynomial size OBDDs with k repeated tests.

Branching programs with k repeated tests do not appear to be usable as data structure because synthesis is NP-hard even for read-once branching programs. But the lower and upper bounds show how OBDDs have to be extended in order to make OBDDs more powerful; it is necessary to test on each path some variables more than once. But then we have to deal with null-chains. Read-k-times ordered binary decision diagrams (kOBDDs) are a variant of branching programs with null-chains. Bollig, Sauerhoff, Sieling, and Wegener [2] have shown that there are efficient algorithms for the operations on Boolean functions represented by kOBDDs. In kOBDDs each path can be partitioned into k parts so that in each part the variables are tested at most once and according to a given ordering. The branching program for HWB is obviously a 2OBDD. Bollig, Sauerhoff, Sieling, and Wegener [3] prove that kOBDDs and kIBDDs of polynomial size form proper hierarchies. In kIBDDs the variable orderings in the k layers may be different. No other hierarchy results for restricted branching programs are known so far.

Our main results are:

- We prove exponential lower bounds for new restrictions of branching programs.
- These lower bounds are obtained by a new lower bound method.
- We get tight hierarchies of functions that can be represented by polynomial size OBDDs and BP1s with k repeated tests.
- The new method allows the proof of lower bounds close to the corresponding upper bounds which has not been possible so far by other lower bound methods.

2. THE CONSIDERED FUNCTIONS AND THE UPPER BOUNDS

The function $f_n^k: \{0,1\}^n \to \{0,1\}$ is defined on the set of variables $X = \{x_0, ..., x_{n-1}\}$. Let m be the largest number where $mk\lceil \log n \rceil \leqslant n$. We partition the set of variables into k groups $X^1, ..., X^k$ each consisting of m numbers of bit length $\lceil \log n \rceil$. Let $s(j)$ be the sum (mod n if n is odd, and mod$(n-1)$ else) of the numbers of the jth group. Then

$$f_n^k(x_0, ..., x_{n-1}) = x_{s(1)} \oplus \cdots \oplus x_{s(k)}.$$

Since in a $\mathrm{BP1}_{+k}$ repeated tests are allowed not only at the end of each computation path, we get different upper bounds for the size of OBDDs and BP1s with k repeated tests for f^k. For simplicity we assume throughout this paper that n is an odd number.

THEOREM 1. (a) *The function $f^k = (f_n^k)$ can be represented by an OBDD_{+k} of size $O(n^{k+1})$.*

(b) *The function $f^k = (f_n^k)$ can be represented by a $\mathrm{BP1}_{+k}$ of size $O(n^2)$.*

Proof. First we describe an OBDD P_l that computes for X^l the value $s(l)$. This OBDD has n sinks numbered from 0 to $n-1$ and the ith sink is reached if $s(l) = i$. The depth of P_l is bounded by $m\lceil \log n \rceil$ since $s(l)$ depends essentially on $m\lceil \log n \rceil$ variables. The contribution of some variable x_r to $s(l)$ is $x_r 2^{\mathrm{pos}(r)} \bmod n$, where pos$(r)$ denotes the position of x_r in its binary number. Hence, width n is sufficient for each level to store the partial sum mod n of the contributions of the variables tested before. The nodes at each level are numbered beginning with 0, and the source is the zeroth node at level 0. The 0-successor of the jth node of the level where x_r is tested is the jth node of the following level; the 1-successor is the $[(j + 2^{\mathrm{pos}(r)}) \bmod n]$th node of the following level.

The jth node of some level is only reached if the sum mod n of contributions of the variables tested before equals j. This also holds for the sink nodes and, hence, P_l computes the desired function. The number of nodes can be estimated by $O(nm\lceil \log n \rceil) = O(n^2/k)$.

The OBDD_{+k} for f_n^k consists of two parts. The top part is an OBDD with n^k sinks that computes the value of the vector $(s(1), ..., s(k))$. This part consists of copies of P_l which are arranged in a complete n-ary tree of depth k. At the ith level of this tree the variables contained in the block X^i are tested by copies of P_i and the value $s(i)$ is computed. The value of $(s(1), ..., s(i))$ is stored in the branching program because paths with different values for $(s(1), ..., s(i))$ are never joined. The number of copies of P_l in the tree is $\sum_{j=0}^{k-1} n^j = O(n^{k-1})$. Therefore, the number of nodes in the top part is bounded by $O(n^{k-1} \cdot n^2/k) = O(n^{k+1}/k)$. We obtain the bottom part if we replace each sink of the top part which is reached if $(s(1), ..., s(k)) = (s^*(1), ..., s^*(k))$ by a branching program of depth k that computes $x_{s^*(1)} \oplus \cdots \oplus x_{s^*(k)}$. Hence, the depth of the bottom part is k and we really get an OBDD_{+k}. The number of nodes in the bottom part is bounded by $O(k \cdot n^k)$ because there are n^k different values of the vector $(s(1), ..., s(k))$.

In a BP1$_{+k}$ we may perform the test of $x_{s(l)}$ immediately after the computation of $s(l)$. We replace the ith sink of P_l by a test of x_i and obtain a branching program P_l^* that computes for the lth block the value $x_{s(l)}$. Then P_1^* and two copies of each P_l^*, $l>1$, are sufficient for the computation of $x_{s(1)} \oplus \cdots \oplus x_{s(k)}$. ▮

The OBDD$_{+k}$ is of polynomial size only if k is a constant. At the end of Section 3 we modify the function f^k by adding dummy variables. For this new function $\tilde{f}^k$ we obtain the upper bound $O(n^2)$ for the size of an OBDD$_{+k}$ also for non-constant k. A superpolynomial lower bound for the size of each OBDD$_{+(k-1)}$ for $\tilde{f}^k$ can be shown if $k = O(\log^{1-\varepsilon} n)$ for some $\varepsilon > 0$.

3. THE LOWER BOUND

First we prove the following property of the function f^k: Even if we replace a large number of input bits by arbitrary constants, it is possible to obtain each value in $\{0, ..., n-1\}^k$ for $(s(1), ..., s(k))$ by choosing a suitable assignment to the remaining bits (Lemma 2). Then we consider some node v in a given BP1$_{+(k-1)}$ for f^k. Using Lemma 2 we show that the sets of variables tested on different consistent paths from the source node to v cannot differ too much if the number of variables tested on some consistent path to v is not too large (Lemma 3, Lemma 4, Lemma 5). Then we can rearrange the given BP1$_{+(k-1)}$ and estimate the number of consistent paths leading from the source node to v (Lemma 6, Lemma 7). In the proof of Theorem 8 we define a set of marked nodes in the BP1$_{+(k-1)}$ and prove a lower bound for the number of consistent paths leading from the source node to all marked nodes. Together with the upper bound of Lemma 6 we obtain the desired exponential lower bound for the size of each BP1$_{+(k-1)}$ for f^k. The hierarchy results are stated in Theorem 9.

LEMMA 2. *Let $t(1), ..., t(k) \in \{0, ..., n-1\}$. If in the input $X = \{x_0, ..., x_{n-1}\}$ at most $m-1$ bits are replaced by arbitrary constants, there is an assignment to the remaining bits so that $s(l) = t(l)$ for all $l \in \{1, ..., k\}$.*

Proof. If at most $m-1$ bits are replaced by constants, there is in each group some binary number in which no bit has been replaced. For each group we can replace all bits outside this number by arbitrary constants and then we can choose a suitable value for this number in order to get $s(l) = t(l)$. ▮

Next we want to show that in each BP1$_{+(k-1)}$ for f^k the numbers of variables tested on different consistent paths from the source node to some node v cannot differ too much if these numbers of variables are not too large. We consider the situation depicted in Fig. 1. On the path P from the source node to v the variables $x_{i(1)}, ..., x_{i(u)}$ are tested, this means we run through P if appropriate values are assigned to $x_{i(1)}, ..., x_{i(u)}$.

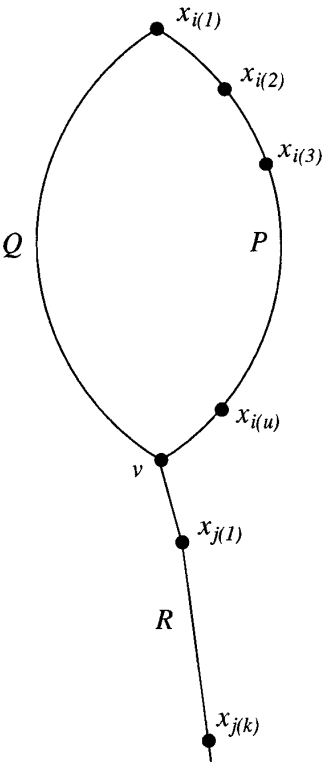

FIGURE 1

In the following we show that it may be necessary to test arbitrarily chosen variables $x_{j(1)}, ..., x_{j(k)} \notin \{x_{i(1)}, ..., x_{i(u)}\}$ on some path R starting at v. This implies that on each path Q leading from the source node to v at most $k-1$ variables of $x_{j(1)}, ..., x_{j(k)}$ and, therefore, at most $k-1$ variables not tested on P may be tested; otherwise the number of variables tested more than once on the path QR would exceed $k-1$.

LEMMA 3. *Let $u \leqslant u^* := m-2k-2$. Let X_1, X_2, and X_3 be a partition of X, where $|X_1| = u$, $|X_2| = k$, and $|X_3| = n-u-k$. For each assignment to the variables in X_1 there is an assignment to the variables in X_3 so that the resulting subfunction of f_n^k cannot be computed by a decision tree of depth less than k.*

Proof. Let an assignment to the variables in X_1 be given. We have to compute a suitable assignment to the variables in X_3. The variables in X_3 are called free until they are fixed to a constant.

The possible contribution of each variable to its binary number is a power of 2. Let $r(1), ..., r(k)$ be the possible contributions of the variables in X_2.

We claim that for some $R \in \{0, ..., n-1\}$ the numbers R and $(R + r(i)) \bmod n$, $1 \leqslant i \leqslant k$, are indices of free variables.

Each of the $u+k \leqslant u^*+k$ variables that is not free excludes at most $k+1$ numbers R. Hence, it is sufficient to prove $(u^*+k)(k+1)<n$. This follows from the inequality

$$\frac{n}{\log n}\frac{k+1}{k}-k^2-3k-2<n,$$

which holds for every n and k.

Since n is odd and the numbers $r(i)$ are powers of 2, we have $r(i) \not\equiv 0 \bmod n$ and, therefore, $R \not\equiv (R+r(i)) \bmod n$ for all $i \in \{1, ..., k\}$. Hence, it is possible to assign 0 to x_R and 1 to $x_{(R+r(1)) \bmod n}, ..., x_{(R+r(k)) \bmod n}$.

Now the variables $x_R, x_{(R+r(1)) \bmod n}, ..., x_{(R+r(k)) \bmod n}$ are no longer free. The number of variables that are not free is still bounded by $u+2k+1 \leqslant m-1$.

Hence, by Lemma 2, there is an assignment to the free variables such that $s(1)=\cdots=s(k)=R$ if the variables in X_2 have value 0. By this assignment we obtain a subfunction f^* depending only on the variables in X_2. In order to prove that the depth of each decision tree for f^* is k we compute the critical complexity of f^*.

The critical complexity $c(g, a)$ of a Boolean function $g \in B_k$ for an input $a \in \{0,1\}^k$ is the number of inputs a' which differ from a in exactly one bit and for which $g(a') \neq g(a)$ holds. The critical complexity $c(g)$ is defined as the maximum of $c(g, a)$ for all $a \in \{0,1\}^k$. It is proved by Bublitz, Schürfeld, Voigt, and Wegener [8] that the depth of each decision tree for g is at least $c(g)$.

We compute $c(f^*, (0, ..., 0))$. If $x=(0, ..., 0)$, we have $s(1)=\cdots=s(k)=R$ and $f^*(0, ..., 0)=0$ because $x_R=0$. Now we consider an input x where exactly one bit x^* is equal to 1. Let x^* be contained in the jth group. Then we have $x_{s(j)}=1$ and $x_{s(l)}=0$ for all $l \neq j$. Therefore $f^*(x)=1$.

Since there are k inputs x with exactly one bit equal to 1, the critical complexity of f^* and, therefore, the depth of each decision tree for f^* is k. ∎

LEMMA 4. *Let a* $\mathrm{BP1}_{+(k-1)}$ *G for f_n^k be given and let $u^*=m-2k-2$. Let $u \leqslant u^*$ and let v be a node in the branching program that is reachable from the source node via a consistent path P on which u variables $x_{i(1)}, ..., x_{i(u)}$ are tested. Then on each other path Q from the source node to v at most $k-1$ variables not contained in $\{x_{i(1)}, ..., x_{i(u)}\}$ are tested.*

Proof. We assign those values to $x_{i(1)}, ..., x_{i(u)}$ for which the path P is chosen (see Fig. 1). For each choice of k variables $x_{j(1)}, ..., x_{j(k)} \notin \{x_{i(1)}, ..., x_{i(u)}\}$ we can apply Lemma 3 for $X_1=\{x_{i(1)}, ..., x_{i(u)}\}$ and $X_2=\{x_{j(1)}, ..., x_{j(k)}\}$. We obtain a subfunction $f^*: X_2 \to \{0,1\}$ which is not computable by a decision tree of depth less than k. This implies that each decision tree and also each branching program for f^* contains a path on which all the variables $x_{j(1)}, ..., x_{j(k)}$ are tested.

It is easy to obtain a branching program for $f_{|x_i=c}$ from a branching program for f: We redirect all edges leading to a node w labeled by x_i to the c-successor of w. If the source node is labeled by x_i, we define its c-successor as new source node. By this procedure we obtain a branching program G^* for f^* starting from the branching program G for f_n^k. By the definition of f^* it follows that G^* contains only nodes of the part of G with source v. Since the branching program for f^* contains a path on which $x_{j(1)}, ..., x_{j(k)}$ are tested, there is also such a path in G starting at v. At most $k-1$ tests may be repeated, therefore, on each path Q leading from the source node to v at most $k-1$ of the variables $x_{j(1)}, ..., x_{j(k)}$ are tested. This holds for all choices of $x_{j(1)}, ..., x_{j(k)} \notin \{x_{i(1)}, ..., x_{i(u)}\}$ and the claim follows. ∎

Let v be a node in a branching program. For each consistent path leading from the source node to v we count how many variables are tested on this path before v is reached. We denote the largest of these numbers by $L(v)$ and the smallest by $S(v)$. If some path contains several tests of some variable, this variable is counted only once. Since we consider only consistent paths, null-chains can affect neither $S(v)$ nor $L(v)$.

LEMMA 5. *Let v be a node in a* $\mathrm{BP1}_{+(k-1)}$ *for f_n^k and let $u^*=m-2k-2$. If $S(v) \leqslant u^*$, then $S(v) \geqslant L(v)-k+1$.*

Proof. We assume $S(v)<L(v)-k+1$ or equivalently $S(v) \leqslant L(v)-k$. This implies that on some path related to $L(v)$ at least k variables are tested which are not tested on some path related to $S(v)$ in contradiction to Lemma 4. ∎

In order to compute a lower bound for the number of nodes in a branching program with $k-1$ repeated tests for f^k we mark a set of nodes in a given $\mathrm{BP1}_{+(k-1)}$. We prove a lower bound for the number of consistent paths leading from the source node to all marked nodes. Together with an upper bound for the number of consistent paths leading to a single marked node we obtain the desired lower bound. In the following lemma we prove the upper bound for the number of consistent paths leading to a single marked node.

LEMMA 6. *Let v be a node in a* $\mathrm{BP1}_{+(k-1)}$ *for f_n^k and let $u^*=m-2k-2$. If $L(v) \leqslant u^*$, then the number of consistent paths leading from the source node to v is bounded by $O(n^{3k-2}2^{u^*(k-1)/k})$.*

Proof. Let T be the decision tree for f_n^k such that for all inputs the sequence of tested variables is the same as in the given $\mathrm{BP1}_{+(k-1)}$. Let V^* be the set of nodes in T representing the given node v and being reached in T on a consistent path. We partition the set of paths leading from the source node of T to some node $v^* \in V^*$ into sets $P(j)$, $1 \leqslant j \leqslant A_v$, of paths on which exactly the same variables are tested. A_v denotes the number of such sets. Lemma 6 follows from the upper bound $O(n^{2k-2})$ for A_v and from the upper bound $n^k 2^{u^*(k-1)/k}$ for the size of the sets $P(j)$.

1. *An Upper Bound for A_v*

Let V_P denote the set of variables tested on the (consistent) path P from the source node to $v^* \in V^*$. We do not include the variable tested at v^* in V_P. Select for P a path that maximizes $|V_P|$. Let Q be some other (consistent) path to v^*. Due to Lemma 4 we can obtain V_Q from V_P if we remove k^* variables from V_P, where $k^* \leqslant k-1$, and add at most k^* other variables. Then the number of possible sets V_Q is bounded by

$$\sum_{k^*=0}^{k-1} \left[\binom{|V_P|}{k^*} \cdot \sum_{j=0}^{k^*} \binom{n}{j} \right] = O(n^{2k-2}).$$

This is the desired upper bound for A_v.

2. *An Upper Bound for the Size of $P(j)$*

Let us consider some set $P(j)$ and let $U = \{x_{i(1)}, ..., x_{i(u)}\}$ be the set of variables tested on the paths in $P(j)$. We know that $u \leqslant u^*$ since $L(v) \leqslant u^*$. Since the subtrees whose sources are contained in V^* are isomorphic, it is possible to merge all nodes in V^* which belong to paths in $P(j)$. Let v^* be the resulting node and let T^* be the decision tree whose source is v^*. On each path in T^* at most $k-1$ variables contained in U are tested. We rearrange T^* in such a way that the U-variables are tested at the end of each path. Let $Y := X - U$. Perform on the decision tree successively the following operations for each $x^* \in Y$:

- Create a new source node labeled by x^*. The successors of this node are two copies of the previous decision tree.
- Eliminate redundant tests and nonreachable nodes and edges.

In the second step all nodes labeled by x^*, except the new source node, are removed. The new decision tree computes the same function as the old one. Before the rearrangement on each path in the decision tree at most $k-1$ of the variables $x_{i(1)}, ..., x_{i(u)}$ are tested. The same holds afterwards because only tests of $x^* \in Y$ are inserted. Now the tests of $x_{i(1)}, ..., x_{i(u)}$ are the last tests on each path. These tests are arranged in small decision trees of depth $k-1$ in the bottom of the decision tree with root v^* (see Fig. 2). In the following we examine which functions have to be computed by these small decision trees.

Each path from the source node to v^* defines an assignment to $x_{i(1)}, ..., x_{i(u)}$. We call $(s^*(1), ..., s^*(k))$ the value of this partial assignment if after assigning 0 to all variables in Y we get $s^*(i)$ as the sum mod n of the numbers in X^i for all $i \in \{1, ..., k\}$. We derive an upper bound for the number of those paths leading from the source node to v^* for which the values of the partial assignments are equal to a fixed vector $(s^*(1), ..., s^*(k))$. We multiply this upper bound by n^k in order to obtain the upper bound for the number of all paths leading to v^*.

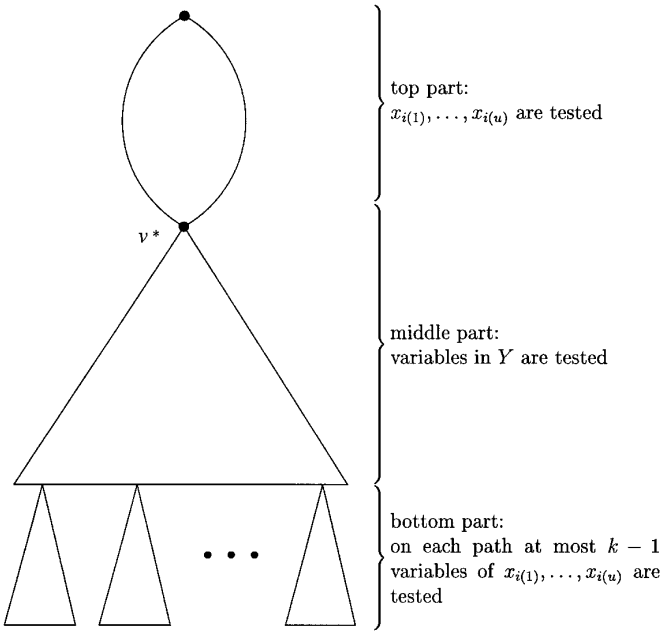

FIGURE 2

Now we fix $(s^*(1), ..., s^*(k))$ and consider only assignments to $x_{i(1)}, ..., x_{i(u)}$ with value $(s^*(1), ..., s^*(k))$. Lemma 2 implies that we can choose for $(s(1), ..., s(k))$ every value in $\{0, ..., n-1\}^k$ and can assign suitable values to the variables in Y in order to obtain that for $i \in \{1, ..., k\}$ the sum mod n of the numbers in X^i is equal to $s(i)$. On the other hand, this assignment to the variables in Y determines a path starting at v^* and leading to one of the small decision trees in the bottom part.

We assign values to the variables in Y so that $s(1) = i(1)$, $s(2) = i(2)$, ..., $s(k) = i(k)$. Then the value of f_n^k is $x_{i(1)} \oplus \cdots \oplus x_{i(k)}$. According to the assignments of the variables in Y we reach one of the small decision trees in the bottom part which computes a function $g_1(x_{i(1)}, ..., x_{i(u)})$. This is also the value that the branching program computes. Since $g_1(x_{i(1)}, ..., x_{i(u)})$ is computed by a decision tree of depth $k-1$, it is different from $x_{i(1)} \oplus \cdots \oplus x_{i(k)}$. Among the assignments to $x_{i(1)}, ..., x_{i(u)}$ with value $(s^*(1), ..., s^*(k))$ only those may define paths leading to v^* for which the equation

$$x_{i(1)} \oplus \cdots \oplus x_{i(k)} \oplus g_1(x_{i(1)}, ..., x_{i(u)}) = 0$$

holds.

We can derive more equations by choosing assignments to the variables in Y for which

$$s(1) = i(k+1), \quad s(2) = i(k+2), ..., \quad s(k) = i(2k),$$

or

$$s(1) = i(2k+1), \quad s(2) = i(2k+2), ..., \quad s(k) = i(3k),$$

and so on. Therefore, all of the following equations have to be satisfied by assignments to $x_{i(1)}, ..., x_{i(u)}$ with value $(s^*(1), ..., s^*(k))$ which define paths leading to v^*:

$$\begin{aligned} x_{i(1)} \oplus \cdots \oplus x_{i(k)} \oplus g_1(x_{i(1)}, ..., x_{i(u)}) &= 0 \\ &\vdots \\ x_{i((t-1)k+1)} \oplus \cdots \oplus x_{i(tk)} \oplus g_t(x_{i(1)}, ..., x_{i(u)}) &= 0. \end{aligned} \tag{1}$$

Here we assume w.l.o.g. that $t := u/k$ is an integer. The function g_j is the function computed by the decision tree of depth $k-1$ which is reached for the corresponding assignment to the variables in Y.

The number of solutions of the system of Eq. (1) is an upper bound for the number of paths with value $(s^*(1), ..., s^*(k))$ leading to v^*. In the following we show that the number of solutions is $2^{u(k-1)/k}$. Since there are n^k possible values for $(s^*(1), ..., s^*(k))$, the number of paths leading to v^* is bounded by $n^k 2^{u(k-1)/k}$. This implies the desired upper bound $n^k 2^{u^*(k-1)/k}$ because $u \leqslant u^*$.

Let $G_j(x_{i(1)}, ..., x_{i(tk)})$ denote the left-hand side of the jth equation of (1), i.e.,

$$\begin{aligned} &G_j(x_{i(1)}, ..., x_{i(tk)}) \\ &\quad := x_{i((j-1)k+1)} \oplus \cdots \oplus x_{i(jk)} \oplus g_j(x_{i(1)}, ..., x_{i(u)}). \end{aligned}$$

For the calculation of the number of solutions of (1) we prove the following lemma.

LEMMA 7. *For each $J \subseteq \{1, ..., t\}$, $J \neq \varnothing$, there are exactly $2^{tk}/2$ assignments to $x_{i(1)}, ..., x_{i(tk)}$ so that*

$$\bigoplus_{j \in J} G_j(x_{i(1)}, ..., x_{i(tk)}) = 0.$$

Proof. Let $J \subseteq \{1, ..., t\}$, $J \neq \varnothing$ be given. We partition the set of all assignments to $x_{i(1)}, ..., x_{i(tk)}$ into classes consisting of two elements. Then we show that for the assignments in each class $\bigoplus_{j \in J} G_j(x_{i(1)}, ..., x_{i(tk)})$ takes different values. Therefore, the numbers of assignments with $\bigoplus_{j \in J} G_j(x_{i(1)}, ..., x_{i(tk)}) = 0$ and $\bigoplus_{j \in J} G_j(x_{i(1)}, ..., x_{i(tk)}) = 1$ are equal. Since there are 2^{tk} assignments to $x_{i(1)}, ..., x_{i(tk)}$ the claim follows.

We describe the partition by a procedure that computes for each assignment ρ the other member $\hat{\rho}$ of the class ρ belongs to. Let us look at the paths that are chosen in the decision trees for the functions $g_j, j \in J$, if we assign values to $x_{i(1)}, ..., x_{i(tk)}$ according to ρ. Since the depth of these decision trees is bounded by $k-1$, there are at most $|J|(k-1)$ variables on the paths selected for this input. The $\oplus$-sum $\bigoplus_{j \in J} G_j(x_{i(1)}, ..., x_{i(tk)})$ consists of the $\oplus$-sum $\bigoplus_{j \in J} g_j(x_{i(1)}, ..., x_{i(tk)})$ and the $\oplus$-sum of $|J|k$ single variables. Therefore, some of the single variables are not tested on any path selected by ρ. Among these variables we choose as x^* the variable with the smallest index. We obtain the assignment $\hat{\rho}$ from ρ by negating the value of x^*. In the decision trees the same paths are selected for ρ and for $\hat{\rho}$ because x^* is not tested on any of these paths. This implies $\hat{\hat{\rho}} = \rho$ and, therefore, this procedure really gives a partition of the set of assignments.

For both ρ and $\hat{\rho}$ the $\oplus$-sum $\bigoplus_{j \in J} g_j(x_{i(1)}, ..., x_{i(tk)})$ takes the same value because in the decision trees the same paths are chosen. But the $\oplus$-sum of single variables takes different values because x^* is different for ρ and $\hat{\rho}$. Therefore, also $\bigoplus_{j \in J} G_j(x_{i(1)}, ..., x_{i(tk)})$ takes different values for ρ and $\hat{\rho}$. ∎

Let N_w, $w \in \{0, 1\}^t$, denote the number of assignments to $x_{i(1)}, ..., x_{i(tk)}$ for which $(G_1(x_{i(1)}, ..., x_{i(tk)}), ..., G_t(x_{i(1)}, ..., x_{i(tk)})) = w$. The number of assignments satisfying all equations in (1) is $N_{(0, ..., 0)}$. We show $N_w = 2^{tk-t} = 2^{u((k-1)/k)}$ not only for $w = (0, ..., 0)$ but even for all $w \in \{0, 1\}^t$.

Since there are 2^{tk} assignments to $x_{i(1)}, ..., x_{i(tk)}$, we get the following equation:

$$\sum_{w \in \{0, 1\}^t} N_w = 2^{tk}. \tag{2}$$

Now fix $J \subseteq \{1, ..., t\}$, $J \neq \varnothing$. The number of assignments to $x_{i(1)}, ..., x_{i(tk)}$ for which $\bigoplus_{j \in J} G_j(x_{i(1)}, ..., x_{i(tk)}) = c$, where $c \in \{0, 1\}$, can be written as

$$\sum_{w | (\bigoplus_{j \in J} w_j) = c} N_w.$$

Lemma 7 implies that this sum takes the same value for $c = 0$ and $c = 1$. This leads to

$$\sum_{w | (\bigoplus_{j \in J} w_j) = 0} N_w - \sum_{w | (\bigoplus_{j \in J} w_j) = 1} N_w = 0. \tag{3}$$

Since there are $2^t - 1$ choices for the set J, we get $2^t - 1$ equations of the form of (3). Together with Eq. (2) we obtain a system of 2^t linear equations with 2^t variables N_w, $w \in \{0, 1\}^t$.

It is easy to check that $N_w = 2^{tk-t}$ for all $w \in \{0, 1\}^t$ satisfies all linear equations. Therefore, it suffices to prove that this is the unique solution. We show that the rank of the matrix of coefficients is 2^t.

We index the columns of this matrix M by vectors $w \in \{0, 1\}^t$ and the rows of M by sets $J \subseteq \{1, ..., t\}$. The row indexed by the empty set belongs to Eq. (2) and the rows indexed by $J \neq \varnothing$ correspond to the equations of the form (3). The entry of M at position (J, w) is

$$M(J, w) = \begin{cases} +1 & \text{if } \bigoplus_{j \in J} w_j = 0 \\ -1 & \text{if } \bigoplus_{j \in J} w_j = 1. \end{cases}$$

We see that M is a Sylvester matrix. The rank of Sylvester matrices is maximal (see, e.g., MacWilliams and Sloane [10]). This completes the proof of Lemma 6. ▮

Now we are ready to prove the lower bound.

THEOREM 8. *The number of nodes in each* $\mathrm{OBDD}_{+(k-1)}$ *and each* $\mathrm{BP1}_{+(k-1)}$ *for* $f^k = (f_n^k)$ *is* $2^{\Omega(n/(k^2 \log n) - 3k \log n - 2k)}$.

If k is a constant, we get the lower bound $2^{\Omega(n/\log n)}$. But the proof works also for non-constant k. If $k \leqslant (1-\varepsilon)(n/3)^{1/3}/\log^{2/3} n$ for some $\varepsilon > 0$, we get the lower bound $2^{\Omega(n^{1/3})}$.

Proof. Let a $\mathrm{BP1}_{+(k-1)}$ for f_n^k be given and let $u^* = m - 2k - 2$. In the given branching program we mark all nodes v for which $L(v) \leqslant u^*$ and $S(v) \geqslant u^* - 2k + 2$. We claim that each consistent path from the source node to a sink node contains at least one marked node. First we know that on each such path more than u^* variables are tested because Lemma 3 implies that the subfunction of f_n^k obtained by assigning constants to u^* variables is not a constant function. Now we search on such a path for that node v where the $(u^* - k + 2)$th variable is tested. Therefore, $S(v) \leqslant u^* - k + 1$ and $L(v) \geqslant u^* - k + 1$. Using Lemma 5 we conclude

$$L(v) \leqslant S(v) + k - 1 \leqslant (u^* - k + 1) + k - 1 = u^*$$

$$S(v) \geqslant L(v) - k + 1 \geqslant (u^* - k + 1) - k + 1 = u^* - 2k + 2.$$

This implies that v is a marked node and that there is at least one marked node on each consistent path from the source node to a sink node. We also know that on each consistent path at least to $u^* - 2k + 2$ variables are tested before a marked node is reached. Therefore, there are at least $2^{u^* - 2k + 2}$ consistent paths from the source node to all marked nodes. If we select a single marked node v, we know because of Lemma 6 that the number of consistent paths from the source node to v is bounded by $O(n^{3k-2} 2^{u^*((k-1)/k)})$. Hence, the number of marked nodes is at least

$$\Omega\left(\frac{2^{u^* - 2k + 2}}{n^{3k-2} 2^{u^*((k-1)/k)}}\right) = 2^{\Omega(n/(k^2 \log n) - 3k \log n - 2k)}. \quad ▮$$

We obtain a tight hierarchy of polynomial size branching programs with k repeated tests if $k \leqslant (1-\varepsilon)(n/3)^{1/3}/\log^{2/3} n$ for some $\varepsilon > 0$. Since the upper bound of Theorem 1 for OBDDs with k repeated tests becomes superpolynomial for non-constant k, we get a hierarchy of polynomial size OBDDs with k repeated tests only for constant k. By adding dummy variables we obtain an upper bound of polynomial size for a function $\tilde{f}^k = (\tilde{f}_n^k)$. Let $\tilde{n} := \lfloor n^{1/k} \rfloor$. The function $\tilde{f}_n^k : \{0,1\}^n \to \{0,1\}$ depends essentially only on the variables $x_0, ..., x_{\tilde{n}-1}$. It is defined by

$$\tilde{f}_n^k(x_0, ..., x_{n-1}) := f_{\tilde{n}}^k(x_0, ..., x_{\tilde{n}-1}).$$

Due to Theorem 1 we get for the size of OBDDs with k repeated tests for $\tilde{f}^k$ the upper bound $O(\tilde{n}^{k+1}) = O(n^{1+1/k})$.

If $k = O(\log^{1-\varepsilon} n)$ for some $\varepsilon > 0$, we can apply Theorem 8 and get the superpolynomial lower bound $2^{\Omega(\log^2 n)}$ for the size of each $\mathrm{OBDD}_{+(k-1)}$ for $\tilde{f}^k$.

Let $P(\mathrm{BP1}_{+k})$ and $P(\mathrm{OBDD}_{+k})$ denote the sets of Boolean functions that can be represented by a polynomial size $\mathrm{BP1}_{+k}$ and OBDD_{+k}, respectively. We have proved:

THEOREM 9. (a) $P(\mathrm{BP1}_{+(k-1)}) \subsetneq P(\mathrm{BP1}_{+k})$ *if* $k \leqslant (1-\varepsilon)(n/3)^{1/3}/\log^{2/3} n$ *for some* $\varepsilon > 0$.

(b) $P(\mathrm{OBDD}_{+(k-1)}) \subsetneq P(\mathrm{OBDD}_{+k})$ *if* $k = O(\log^{1-\varepsilon} n)$ *for some* $\varepsilon > 0$.

Since the classes of both hierarchies are separated by the same functions, we also have

$$P(\mathrm{OBDD}_{+k}) \not\subseteq P(\mathrm{BP1}_{+(k-1)})$$
$$\text{if } k = O(\log^{1-\varepsilon} n) \text{ for some } \varepsilon > 0.$$

ACKNOWLEDGMENT

I thank Ingo Wegener for many helpful remarks on earlier versions of this paper.

REFERENCES

1. N. Alon and W. Maass, Meanders and their applications in lower bound arguments, *J. Comput. System Sci.* **37** (1988), 118–129.
2. B. Bollig, M. Sauerhoff, D. Sieling, and I. Wegener, Read k times ordered binary decision diagrams—Efficient algorithms in the presence of null-chains, Tech. Report, Universität Dortmund, 1993.
3. B. Bollig, M. Sauerhoff, D. Sieling, and I. Wegener, On the power of different types of restricted branching programs, submitted.
4. A. Borodin, A. Razborov, and R. Smolensky, On lower bounds for read-k-times branching programs, *Comput. Complexity* **3** (1993), 1–18.
5. R. E. Bryant, Symbolic manipulation of Boolean functions using a graphical representation, *in* "Proceedings, 22nd Design Automation Conference, 1985," pp. 688–694.
6. R. E. Bryant, Graph-based algorithms for Boolean function manipulation, *IEEE Trans. Comput.* **35** (1986), 677–691.
7. R. E. Bryant, On the complexity of VLSI implementations and graph representations of Boolean functions with application to integer multiplication, *IEEE Trans. Comput.* **40** (1991), 205–213.
8. S. Bublitz, U. Schürfeld, B. Voigt, and I. Wegener, Properties of complexity measures for PRAMs and WRAMs, *Theoret. Comput. Sci.* **48** (1986), 53–73.
9. J. Gergov and C. Meinel, Frontiers of feasible and probabilistic feasible Boolean manipulation with branching programs, *in* "Proceedings, 10th Symposium on Theoretical Aspects of Computer Science 1993," pp. 576–585.

10. F. J. MacWilliams and N. J. A. Sloane, "The Theory of Error-Correcting Codes," North-Holland, Amsterdam, 1977.
11. È. I. Nečiporuk, A Boolean function, *Soviet Math. Dokl.* **7** (1966), 999–1000.
12. D. Sieling and I. Wegener, Graph driven BDDs—A new data structure for Boolean functions, *Theoret. Comput. Sci.* **141** (1995), 283–310.
13. I. Wegener, On the complexity of branching programs and decision trees for clique functions, *J. Assoc. Comput. Mach.* **35** (1988), 461–471.
14. I. Wegener, Efficient data structures for Boolean functions, *Discrete Math.* **136** (1994), 347–372.
15. S. Žák, An exponential lower bound for one-time-only branching programs, *in* "Proceedings, 11th Symposium on Mathematical Foundations of Computer Science, 1984," pp. 562–566.