ELSEVIER
Theoretical Computer Science 145 (1995) 45–69
Theoretical Computer Science

On the size of binary decision diagrams representing Boolean functions

Y. Breitbart[a,*,1], H. Hunt III[b,2], D. Rosenkrantz[b,3]

[a] *Computer Science Department, University of Kentucky, Lexington, KY 40506-0027, USA*
[b] *Computer Science Department, SUNY at Albany, 1400 Washington Ave., Albany, NY 12222, USA*

Received January 1994; revised June 1994
Communicated by M. Nivat

Abstract

We consider the size of the representation of Boolean functions by several classes of binary decision diagrams (BDDs) (also called *branching programs*), namely the classes of *arbitrary* BDDs of *real time* BDD (RBDD) (i.e. BDDs where each computation path is limited to the number of variables), of free BDDs (FBDDs) (also called *read-once-only* branching programs), of *ordered* BDDs (OBDDS) i.e. FBDDs where variables are tested in the same order along all paths), and *binary decision trees* (BDTs).

Using well-known techniques, we first establish asymptotically sharp bounds as a function of n on the minimum size of arbitrary BDDs representing almost all Boolean functions of n variables and provide asymptotic lower and upper bounds, differing only by a factor of two, on the minimum size OBDDs representing almost all Boolean functions of n variables.

We then, using a method to obtain exponential lower bounds on complexity of computation of Boolean functions by RBDD, FBDD and OBDD that originated in (Breitbart, 1968), present the highest such bounds to date and also present improved bounds on the relative economy of description of particular Boolean functions by the above classes of BDDs. For each nontrivial pair of BDD classes considered, we exhibit infinite families of Boolean functions representable much more concisely by BDDs in one class than by BDDs in the other.

1. Introduction

Binary decision diagrams (BDDs) play an important role in several application areas including VLSI design, pattern recognition, decision table programming, and

* Corresponding author.
[1] This material is based in part upon work sponsored by the Center for Manufacturing and Robotics of the University of Kentucky and by the National Science Foundation under grant No. IRI-9221947.
[2] Research supported by NSF Grant CCR 89-03319.
[3] Research supported by NSF Grant CCR 90-06396

SSDI 0304-3975(94)00181-2

Computer aided design (CAD). Thus they have attracted a great deal of attention both from theoreticians [3, 6–11, 15, 16, 19, 24, 29, 30, 32, 34–38] and from applied researchers [1, 2, 4, 5, 12–14, 18, 20, 27, 28, 31].

Theoreticians used restricted classes of branching programs for proving exponential lower bounds on computational complexity of different problems [29, 30, 32, 34–38, 23, 21]. It appears that it is easier to develop lower bound techniques for branching programs than for Turing machines and yet both these models have an equivalent computational power. Recently, it was shown that lower bound techniques developed for branching programs can be extended to obtain nonlinear lower bounds for VLSI circuit complexity [21].

Engineering community has advocated using BDDs and OBDDs as a CAD specification language for Boolean functions [4, 12]. The use of OBDDs, rather than arbitrary BDDs, facilitates the solution of various analysis and synthesis problems. However, there has been concern in the CAD community with the size of OBDDs representing particular Boolean functions.

These considerations have generated considerable interest in the comparative complexity of a Boolean function description by different BDD models.

Formally, a BDD is a labeled directed acyclic graph with the following properties:

1. There is exactly one source node.
2. Each nonsink node has outdegree 2, is labeled by a Boolean variable, and is called an *internal node.*
3. Each internal node has one outgoing edge labeled by 0 and the other labeled by 1.
4. Each sink node is labeled by 0 or 1, and is called a *terminal* node.

By the *size* of BDD A, denoted by $L(A)$, we mean the number of internal nodes of A. Let n ($n > 1$) be a number of Boolean variables that are used to label BDD nodes. For each assignment to the variables labeling a given BDD, there is a unique path from the source node to a terminal node. A Boolean function $f(x_1, \ldots, x_n)$ (where $n \geqslant 0$) is *computed* (or *represented*) by a given BDD if for each sequence of n Boolean values $\alpha = \alpha_1, \ldots, \alpha_n$, the path corresponding to α ends with a terminal node whose label is $f(\alpha_1, \ldots, \alpha_n)$.

A *real-time* BDD (RBDD) is a BDD for which the length of each path does not exceed n, where n is the length of the input word. A *free* BDD (FBDD) is a BDD for which each path has at most one occurrence of each variable. An *ordered* BDD (OBDD) is a FBDD such that the order of occurrences of variables on all its path is consistent with some linear order, i.e. if variable x occurs before variable y on some path, then there is no path where y occurs before x. A *binary decision tree* (BDT) is a BDD for which each node has indegree at most one.

The relative succinctness of two BDD classes can be captured by results showing that there are functions succinctly representable via BDDs in the first class, but for which a much larger size is required if the representing BDD must be in the second class. The strongest such result would say that BDD in the second class is exponentially larger than the BDD in the first class. For instance, it is well known [36] that there are Boolean functions representable by OBDDs of size linear in the number of variables n, but for which any BDT has size exponential in n. Here we consider the reverse case, and exhibit a family of Boolean functions $\{f_n \mid n \geqslant 1\}$ for which we show that they can be computed by BDT of size $\Theta(n^2/\log n)$ but require at least $\Omega(2^{n/\log n})$ nodes to be computed by any OBDD. This result says that there are BDTs of size m such that any equivalent OBDD must be of size roughly $2^{\sqrt{m/\log m}}$.

Complexity of computation of Boolean functions by FBDD were studies by several researchers. Wegener [35] exhibited a family of Boolean functions $\{f_n \mid n \geqslant 1\}$ for which any FBDD computing f must have at least $\Omega(2^{cn^{1/3}})$ size. Zak [38] exhibited a family $\{f_n \mid n \geqslant 1\}$ such that any FBDD computing f must have size of at least $\Omega(2^{c\sqrt{n}}/n)$. Similar results were obtained by Dunne [17] and Krause [22]

These results were improved by Wegener [37] and by Ajtai et al. [3, 6]. In [6] a family $\{f_n \mid n \geqslant 1\}$ is exhibited such that each function requires at least $\Omega(2^{cn})$ nodes to be computed by FBDD. Their constant c, however, is very small, namely $c < 10^{-13}$.

Clearly, each FBDD is also RBDD but not vice versa. Kriegel and Waack [24] exhibit a family $\{f_n \mid n \geqslant 1\}$ of Boolean functions each of which requires RBDD of $\Omega(2^{n/48})$ size. In contrast, we exhibit a family $\{f_n \mid n \geqslant 1\}$ such that each such function can be computed by BDD of size $\Theta(n^2)$ and yet each RBDD computing the function must have at least $2^{n/3}$ nodes. In addition, we exhibit another family $\{f_n \mid n \geqslant 1\}$ such that each such function can be computed by RBDD of $\Theta(n^2)$ size and yet each FBDD computing the function requires at least $c2^{n/6}$ nodes.

It appears that the above result provides the largest lower bound for RBDD (and, by implication, for FBDD and OBDD) achieved so far. Our construction uses methods developed in [8–11].

The relationship between FBDDs and OBDDs was considered by Fortune et al. [19]. They exhibited a family $\{f_n \mid n \geqslant 1\}$ such that each such function can be computed by FBDD of $\Theta(n^2)$ size and yet each OBDD computing the function requires at least $\Theta(2^{\sqrt{n}-o(n)})$ size. In contrast, we exhibit here a family $\{f_n \mid n \geqslant 1\}$ such that each such function can be computed by FBDD of $\Theta(n^2)$ size and yet each OBDD computing the function requires at least $2^{n/9}$ nodes.

The rest of the paper is organized as follows. Section 2 contains basic definitions and outlines our method for proving exponential lower bounds for computations on BDDs. In Section 3, we consider the asymptotic size of general BDDs and OBDDs representing Boolean functions. It contains asymptotically sharp bounds as a function of n on the minimum size of BDDs representing almost all Boolean functions of n variables. Our asymptotic lower and upper bounds as a function of n on the minimum size OBDDs representing almost all Boolean functions of n variables are asymptotically differ only by a factor or two. We have been informed that a similar result concerning BDDs appears in [25], which seems to be part of a series of technical reports. However, we have been unable to obtain it, and have not found citations to it in the literature on BDDs. The asymptotic result for OBDD is obtained by a straightforward analysis of a well known general OBDD construction. Both these results are included here for the sake of completeness of our presentation and a lack of single place where asymptotic bounds are presented. Our main results are contained in the next three sections.

In Section 4, we consider the relative succinctness of OBDDs and BDTs. In Section 5, we define a family of Boolean functions $\{f_n \mid n \geqslant 1\}$ for which we show that any RBDD computing a function from our class requires $2^{n/3}$ size. In Section 6 we construct Boolean functions that can be easily computed by BDD (RBDD, FBDD, respectively) and require exponential size for RBDD (FBDD, OBDD, respectively). Section 7 concludes the paper.

2. Basic definitions and lower bound technique

Let A be a BDD, τ be a class of BDDs, f be a Boolean function, and $n \geqslant 1$ be an integer. We define the τ-size of f, denoted by $L_\tau(f)$, as

$$L_\tau(f) = \min\{L(A) \mid A \text{ represents } f \text{ and } A \in \tau\}.$$

We define $L_\tau(n)$ as

$$L_\tau(n) = \max\{L_\tau(f) \mid g \text{ is an } n \text{ variable Boolean function}\}.$$

Since BDD $\supseteq$ RBDD $\supseteq$ FBDD $\supseteq$ OBDD and BDD $\supseteq$ RBDD $\supseteq$ FBDD $\supseteq$ BDT, the following relationships hold trivially for all Boolean functions f:

$$L_{\mathrm{BDD}}(f) \leqslant L_{\mathrm{RBDD}}(f) \leqslant L_{\mathrm{FBDD}}(f) \leqslant \min\{L_{\mathrm{OBDD}}(f), L_{\mathrm{BDT}}(f)\}.$$

Let $f, g : \mathbf{N} \to \mathbf{R}$ be such that $f(n), g(n) > 0$. We write $f(n) \sim g(n)$, if $\lim_{n\to\infty} f(n)/g(n) = 1$.

Let $X = (x_1, \ldots, x_n)$ and $X \supseteq Y = (x_{i_1}, \ldots, x_{i_k})$, where $k > 1$. If $f(X)$ is a Boolean function of n variables and $\alpha = \alpha_{i_1}, \ldots, \alpha_{i_k}$ are values for variables from Y, then we denote f^α the Boolean function obtained from f by substitution of $\alpha_{i_1}, \ldots, \alpha_{i_k}$ for variables from Y. It is well known that every Boolean function has a unique ring sum representation [33].

We say that a Boolean function $f(X)$ is *linear* in variables from Y if every product in its ring sum representation contains at most one variable from Y. If a Boolean function $f(X)$ is linear in all of its variables, then f is called *linear*. We say that a Boolean function $f(X)$ is *symmetric* in variables from Y, if $f(X)$ does not change under any permutation of variables from Y. If a function is symmetric in all of its variables, then it called *symmetric*. A Boolean function $f(x_1, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_n)$ *depends essentially on* x_i iff $f(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n) \neq f(x_i, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n)$. From the above definitions, it can be easily derived that if $f(X)$ is symmetric in Y, then either $f(X)$ depends essentially on all variables from Y, or it does not depend essentially on any variable from Y. Hence, any symmetric and nonconstant Boolean function depends essentially on all of its variables.

Lemma 1. *If $f(X)$ is symmetric and linear in Y, then*

$$f(X) = \Phi_1(X, Y) + \Phi_2(X, Y) \cdot \sum_{x' \in Y} x', \tag{1}$$

where $+$ is taken by modulo two.

Proof. Two cases are possible: (1) either f does not depend essentially on any variable from Y, or (2) it depends essentially on all variables from Y. In the first case, assuming $\Phi_2(X, Y) = 0$, we derive (1). In the second case, consider those products from ring sum representation of f that contain variables from Y. Since f is linear in Y, each such product contains exactly one variable from Y. Let P_j $(1 \leqslant j \leqslant k)$ be a sum of all products that contain variable x_{i_j} from Y. Since f is symmetric in Y, we obtain that all P_j are the same, and, thus, (1) holds. □

Corollary 1. *If $f(X)$ is symmetric and linear and not a constant, then $f(X) = x_1 + x_2 + \cdots + x_n + \alpha$, where α is a constant from $\{0, 1\}$* (i.e. *$f(X)$ is linear in all of its arguments*).

Proof. Since f is linear, functions Φ_1 and Φ_2 in (1) are constants. Since f is not a constant, we obtain that Φ_2 is equal to 1. □

Let $S_n^i(X)$ be the symmetric Boolean function that is equal to one iff exactly $0 \leqslant i \leqslant n$ of its variables are equal to one. $S_n^i(X)$ is called an elementary symmetric function. By definition, $S_n^i \equiv 0$ if $i > n$ or $i < 0$. It is well known [33] that if k-tuple α contains exactly s ones among its k values, then

$$S_n^{i,\alpha} = S_{n-k}^{i-s}. \tag{1a}$$

Lemma 2. *Let $X = (x_1, \ldots, x_n)$ and $Y = (y_1, \ldots, y_n)$ be disjoint sets of variables. Let $f^1(X, Y) = \sum_{i=0}^n L_i^1(X) S_n^i(Y)$ and $f^2(X, Y) = \sum_{i=0}^n L_i^2(X) S_n^i(Y)$, where L_i^1 and L_i^2 are arbitrary Boolean functions. Then*

$$f^1 = f^2 \text{ iff for all } 0 \leqslant i \leqslant n,\ L_i^1 = L_i^2.$$

Proof. Let us assume to the contrary, $f^1 = f^2$, but there is a k such that $L_k^1 \neq L_k^2$. Then

$$f^1 + f^2 = 0 = \sum_{i=0}^{n} (L_i^1(X) + L_i^2(X)) S_n^i(Y) \tag{2}$$

Since $L_k^1 + L_k^2 \neq 0$, there is an α such that $L_k^1(\alpha) + L_k^2(\alpha) = 1$. Let β be an n-tuple that contains exactly k ones. Then $S_n^k(\beta) = 1$, and $S_n^i(\beta) = 0$ for each $i \neq k$. Thus, $f^1(\alpha\beta) + f^2(\alpha\beta) = 1$, which contradicts (2). □

Using Lemma 2 straightforwardly, we derive the following lemma.

Lemma 3. *If $f = \sum_{i=0}^n L_i(X) S_n^i(Y)$ (where again X and Y are disjoint sets of n variables), then f depends essentially on x_k from X if and only if there is a j such that L_j depends* *essentially on x_k. Moreover, if f depends essentially on at least one variable from X, then it depends essentially on all variables from Y.*

Our lower bounds estimates are based on the following lemma.

Lemma 4. *Let $f(X)$ be a Boolean function and k be an integer such that $0 \leqslant k < n$. Suppose that for any k variables $x_{i_1} \cdots x_{i_k}$, the functions obtained by substituting constants for these variables are all different and depend essentially on all the remaining variables. They any* RBDD *for f contains at least 2^k internal nodes.*

Proof. Let A be a RBDD computing f. Any path in A to a terminal node must contain more than k edges. Therefore there are 2^k paths of length k, each reaching an internal node.

Consider any pair of such paths. If the two paths involve different sets of k variables, then the Boolean functions of the remaining variables after traversing the two paths are different, since these functions depend essentially on different sets of variables. If the two paths involve the same set of k variables, then by the hypothesis, the functions of the remaining variables are different. Therefore, each of the 2^k paths of length k must reach a distinct internal node of A. □

3. Representation of Boolean functions by BDDs

Since it is very difficult to find a specific Boolean function whose representation by a BDD requires exponential size, researchers have attempted to find such a function using an asymptotic approach. Such methods allow us to find a uniform method that builds minimal size BDDs for the most complex Boolean functions.

Although it is well known that $L_{\mathrm{BDD}}(n)$ is, roughly, exponential in n, the exact size complexity does not seem to be well known. For instance, in his widely cited book [36], Wegener presents the following bounds on $L_{\mathrm{BDD}}(n)$:

$$\frac{2^n}{3n} \leqslant L_{\mathrm{BDD}}(n) \quad \text{and} \quad L_{\mathrm{BDD}}(n) \text{ is } \mathrm{O}\left(\frac{2^n}{n}\right).$$

However, the original results on $L_{\mathrm{BDD}}(n)$ in [26] are stronger, namely

$$\frac{2^{n-1}}{n} < L_{\mathrm{BDD}}(n) < \frac{2^{n+2}}{n} - 1.$$

Our Theorem 1 provides an asymptotically *exact* estimate on $L_{\mathrm{BDD}}(n)$.

Theorem 1. *For all $n \geqslant 1$, $(2^n/n)(1 - \varepsilon_n) \leqslant L_{\mathrm{BDD}}(n) \leqslant (2^n/n)(1 + \varepsilon_n)$, where $\varepsilon_n \to 0$ as $n \to \infty$. Thus, $L_{\mathrm{BDD}}(n) \sim 2^n/n$.*

Proof. We prove a stronger statement, namely, that "almost all" Boolean function of n variables require $(2^n/n)\,(1-\varepsilon_n)$ internal nodes for their computation by BDDs, where $\varepsilon_n \to 0$ as $n \to \infty$.

Upper bound: $L_{\mathrm{BDD}}(n) \leqslant (2^n/n)(1+\varepsilon_n)$, *where* $\varepsilon_n \to 0$ *as* $n \to \infty$. To prove the upper bound we design a computation method that for any Boolean function f of n arguments constructs a BDD computing f that has no more than $(2^n/n)\,(1+\varepsilon_n)$ internal nodes, where $\varepsilon_n \to 0$, as $n \to \infty$. The computation utilizes a representation of Boolean functions described in [33].

Let $f(X)$ be a Boolean function of n variables. We choose a parameter k, where $1 \leqslant k \leqslant n$. Then $f(X)$ can be represented by a $2^k \times 2^{n-k}$ matrix, whose rows are numbered by k-tuples of values for $x_1,\dots,x_k$ and whose columns are numbered by $(n-k)$-tuples of values for $x_{k+1},\dots,x_n$. The value of the function for $\alpha_1,\dots,\alpha_n$ is placed at the intersection of the $\langle \alpha_1,\dots,\alpha_k\rangle$ row and the $\langle \alpha_{k+1},\dots,\alpha_n\rangle$ column.

Let us also choose a parameter s where $1 \leqslant s \leqslant 2^k$, and subdivide the rows of the table into $\lceil 2^k/s \rceil$ groups with s consecutive rows per group.

We define $f_i(X)$, where $1 \leqslant i \leqslant \lceil 2^k/s \rceil$, as follows:

$$f_i(X) = \begin{cases} f(X) & \text{if } \langle \alpha_1,\dots,\alpha_k\rangle \text{ is a member of the } i\text{th group of rows,} \\ 0 & \text{otherwise.} \end{cases}$$

Therefore, the function $f(X)$ is a disjunction of all the functions $f_i(X)$ and the following relation holds (where $\sum$ here denotes disjunction).

$$f(X) = \sum_{i=1}^{\lceil 2^k/s \rceil} f_i(X). \tag{3}$$

It is easy to see that any column of values for the function $f_i(X)$ has zeros outside of ith group of rows and possible nonzero values within the ith group of rows. The possible number of different columns in the matrix of values for the ith function is no more than 2^s, since a column may contain only s nonzero values. The columns of the ith function can be subdivided into p groups, where $1 \leqslant p \leqslant 2^s$. Two columns belong to the same group if and only if they are identical for the function f_i.

We define $f_{ij}(X)$ as follows:

$$f_{ij}(X) = \begin{cases} f_i(X) & \text{for the columns of the } j\text{th group,} \\ 0 & \text{otherwise.} \end{cases}$$

Thus

$$f(X) = \sum_{i=1}^{\lceil 2^k/s \rceil} \sum_{j=1}^{2^s} f_{ij}(X). \tag{4}$$

A function f_{ij} has columns of only two types: either a column consisting of only zeros or a column with $\beta_1,\dots,\beta_s$ in the ith group of rows and zeros outside of it. Therefore, every f_{ij} can be considered as a conjunction of two functions: $f_{ij}^{(1)}(x_1,\dots,x_n)$ and $f_{ij}^{(2)}(x_{k+1},\dots,x_n)$, where $f_{ij}^{(1)}(x_1,\dots,x_k)$ depends only on variables $x_1,\dots,x_k$ and is equal to one in rows where the jth group of columns for the ith group of rows has at least one 1, and $f_{ij}^{(2)}(x_{k+1},\dots,x_n)$ depends only on variables $x_{k+1},\dots,x_n$ and is equal to 1 in columns of the jth group.

Thus, combining representations (3) and (4) we derive that $f(X)$ can be represented as follows.

$$f(X)=\sum_{i=1}^{\lceil Y/s\rceil}\sum_{j=1}^{2^s} f_{ij}^{(1)}(x_1,\dots,x_k)\, f_{ij}^{(2)}(x_{k+1},\dots,x_n). \tag{5}$$

Representation (5) has the following properties:

1. Every $f_{ij}^{(1)}{}^{(}(x_1,\dots,x_k)$ has no more than s values equal to 1.
2. $f_{ij_1}^{(2)}(x_{k+1},\dots,x_n)\, f_{ij_2}^{(2)}(x_{k+1},\dots,x_n)=0$, if $j_1\neq j_2$.

An example of the described representation is shown in Fig. 1 for a function of 7 variables,. where k and s are equal to 3.

Based on the representation (5) we construct a BDD A computing $f(X)$. The constructed BDD consists of three block – $B1$, $B2$, $B3$ described below.

			0000000011111111	x_4
			0000111100001111	x_5
			0011001100110011	x_6
x_1	x_2	x_3	0101010101010101	x_7
0	0	0	0000001110011011	First
0	0	1	0011110001111100	group
0	1	0	0101100000011111	of rows
0	1	1	0011110000000111	Second
1	0	0	0001110001100010	group
1	0	1	1100011111000000	of rows
1	1	0	0001110000000001	Third group
1	1	1	0110001110011111	of rows

(a) $f(x_1,x_2,x_3,x_4,x_5,x_6,x_7)$

			0000000011111111	x_4
			0000111100001111	x_5
			0011001100110011	x_6
x_1	x_2	x_3	0101010101010101	x_7
0	0	0	0000000000000000	
0	0	1	0000000000000000	
0	1	0	0000000000000000	
0	1	1	0011110000000111	
1	0	0	0001110001100010	
1	0	1	1100011111000000	
1	1	0	0000000000000000	
1	1	1	0000000000000000	

(b) $f_2(x_1,x_2,x_3,x_4,x_5,x_6,x_7)$

Fig. 1.

			0000000011111111	x_4
			0000111100001111	x_5
			0011001100110011	x_6
x_1	x_2	x_3	0101010101010101	x_7
0	0	0	0000000000000000	
0	0	1	0000000000000000	
0	1	0	0000000000000000	
0	1	1	0001100000000010	
1	0	0	0001100000000010	
1	0	1	0000000000000000	
1	1	0	0000000000000000	
1	1	1	0000000000000000	

(c) $f_{2,7}(x_1,x_2,x_3,x_4,x_5,x_6,x_7)$

			0000000011111111	x_4
			0000111100001111	x_5
			0011001100110011	x_6
x_1	x_2	x_3	0101010101010101	x_7
0	0	0	0000000000000000	
0	0	1	0000000000000000	
0	1	0	0000000000000000	
0	1	1	1111111111111111	
1	0	0	1111111111111111	
1	0	1	0000000000000000	
1	1	0	0000000000000000	
1	1	1	0000000000000000	

(d) $f^{(1)}_{2,7}(x_1,x_2,x_3,x_4,x_5,x_6,x_7)$

			0000000011111111	x_4
			0000111100001111	x_5
			0011001100110011	x_6
x_1	x_2	x_3	0101010101010101	x_7
0	0	0	0001100000000010	
0	0	1	0001100000000010	
0	1	0	0001100000000010	
0	1	1	0001100000000010	
1	0	0	0001100000000010	
1	0	1	0001100000000010	
1	1	0	0001100000000010	
1	1	1	0001100000000010	

(e) $f^{(2)}_{2,7}(x_1,x_2,x_3,x_4,x_5,x_6,x_7)$

Fig. 1 (*continued*).

1. Block $B1$ values $x_1, \ldots, x_k$ determines a group number. Let $L(B1)$ be the number of internal nodes of $B1$. Block $B1$ consists of a binary tree of k levels whose nodes are labeled with x_1 through x_k, and a number of nodes at the $(k+1)$th level equal to the number of different subfunctions f_i in the representation (3). Therefore,

$$L(B1) \leqslant \sum_{l=0}^{k-1} 2^l + \frac{Y}{s} + 1 = Y\left(1 + \frac{1}{s}\right).$$

2. Block $B2$ using a group number determined by block $B1$, determines a column group number based on the values of the last $(n-k)$ variables of the function f. Block $B2$ consists of no more than $\lceil 2^k/s \rceil$ smaller blocks $B2_i$. Each such subblock consists of a binary tree whose root is labeled with x_{k+1} and was counted as a part of block $B1$, a sequence of $n-k-1$ levels labeled with variables x_{k+2} through x_n, and a final level containing a node for each column group. Since there are no more than 2^s nodes in the final level of each subblock, the number of nodes in all $B2_i$ blocks is

$$L(B2) \leqslant \left(\frac{Y}{s} + 1\right)\left(\sum_{l=1}^{n-k-1} 2^l + 2^s\right) = \left(\frac{Y}{s} + 1\right)(2^{n-k} - 2 + 2^s)$$
$$= \frac{2^n}{s} + 2^{n-k} - \frac{2^{k+1}}{s} - 2 + 2^s + \frac{2^{k+s}}{s}.$$

3. Block $B3$ using a row group number and a column group number computes $f_{ij}(x_{12}, \ldots, x_n)$ and, thereby completes the computation of the function value for a given set of input values. Block $B3$ consists of no more than $(2^k/s + 1)2^s$ smaller blocks $B3_{ij}$, where each such block computes function $f_{ij}(x_1, \ldots, x_n)$. Since each such function can be represented as shown in (5), each $B3_{ij}$ is a binary tree. The root of this tree is labeled with x_1 and was counted as part of block $B2$. The tree then has $k-1$ levels labeled with variables x_2 through x_k. Finally, there is a level containing terminal nodes. The number of internal nodes in block $B3$ is

$$L(B3) \leqslant \left(\frac{Y}{s} + 1\right) 2^s \sum_{i=1}^{k-1} 2^i$$
$$= \left(\frac{2^{k+s}}{s} + 2^s\right)(Y - 2)$$
$$= \frac{2^{2k+s}}{s} + 2^{k+s} - \frac{2^{k+s+1}}{s} - 2^{s+1}.$$

Hence, the complexity of A is equal to the sum of $L(B1)$–$L(B3)$,

$$L(A) = L(B1) + L(B2) + L(B3)$$
$$= Y + \frac{Y}{s} + \frac{2^n}{s} + 2^{n-k} - \frac{2^{k+1}}{s} - 2 + 2^s + \frac{2^{k+s}}{s} + \frac{2^{2k+s}}{s}$$
$$+ 2^{k+s} - \frac{2^{k+s+1}}{s} - 2^{s+1}$$

$$= 2^{n-k} + 2^{k+s} + 2^k - 2^s - 2 + \frac{1}{s}(2^n + 2^{2k+s} - 2^k - 2^{k+s})$$

$$< \frac{2^n}{s} + 2^{n-k} + \frac{2^{2k+s}}{s} + 2^{k+s} + 2^k.$$

Let us choose $k = 2\lfloor \log n \rfloor$ and $s = n - 6\lfloor \log n \rfloor$. Since $2\log n - 2 < k \leqslant 2\log n$ and $n - 6\log n \leqslant s < n - 6\log n + 6$, we obtain

$$L(A) < \frac{2^n}{n - 6\log n} + \frac{2^{n+2}}{n^2} + \frac{2^{n+6}}{n^2(n - 6\log n)} + \frac{2^{n+6}}{n^4} + n^2 \leqslant \frac{2^n}{n}(1 + \varepsilon_n).$$

This completes the proof of the upper bound.

Lower bound: $L_{\mathrm{BDD}}(f) \geqslant (2^n/n)\ (1 - \varepsilon_n)$. To prove that for almost all Boolean functions f of n arguments, $L_{\mathrm{BDD}}(f) \geqslant (2^n/n)(1 - \varepsilon_n)$ where $\varepsilon_n \to 0$ as $n \to \infty$, we use counting arguments. Consider a BDD with k internal nodes and let us assign to each node a unique tag consisting of a number from 1 through k, assigning different tags to different nodes. Each internal node in the BDD is labeled by one of the variables from X. Without loss of generality, we assume that there are only two terminal nodes in each BDD. One terminal node is labeled with 0 and assigned tag $k + 1$, and the other is labeled 1 and assigned tag $k + 2$.

Each BDD with k tagged internal nodes can be uniquely represented as follows. Consider a table of k rows. Row i represents internal node i, and contains an entry with three values (q, j_1, j_2). Suppose internal node i of the BDD is labeled by variable x_q, is connected to node j_1 by an edge labeled 0, and is connected to node j_2 by an edge labeled 1. Then row i contains the entry (q, j_1, j_2).

Since each row of the table has $(k + 2)^2 n$ possible values, there are $(k + 2)^{2k} n^k$ such tables (not all of which represent BDDs). However a BDD with k internal nodes can be assigned tags in $k!$ different ways, and therefore is represented by $k!$ isomorphic tables. The number of nonisomorphic BDDs with k internal nodes is thus no more than

$$\frac{(k + 2)^{2k} n^k}{k!} \leqslant (ckn)^k,$$

where c is some constant.

It is well known that there are 2^{2^n} Boolean functions of n variables. Therefore, the number of different nonisomorphic BDDs with k internal nodes should be large enough to compute every Boolean function of n variables.

Thus $(ckn)^k \geqslant 2^{2^n}$. Consequently, $k \geqslant (2^n/n)\ (1 - \varepsilon_n)$. □

It should be noted that the asymptotically minimal size BDD to evaluate any Boolean function of n variables obtained by our method reads each variable no more than twice. Perhaps this fact explains why it is so difficult to obtain an exponential lower bounds for read-only twice branching program without any additional restrictions. Obtaining an exponential lower bound for read-only twice branching program would in effect allow to exhibit the most complex Boolean functions to be computed by general BDDs.

The next theorem provides a general construction of an OBDD (and hence a FBDD and a RBDD) whose size differs from the lower bound of Theorem 1 by a factor of at most 2.

Theorem 2. *For all* $n \geqslant 1$, $(2^{n+1}/n)\ (1+\varepsilon_n) \geqslant L_{\mathrm{OBDD}}(n) \geqslant (2^n/n)\ (1-\varepsilon_n)$, *where* $\varepsilon_n \to 0$ *as* $n \to \infty$.

Proof. *Upper bound*: To prove the upper bound we design a computation method that for any Boolean function f of n variables constructs a OBDD computing f that has no more than $(2^{n+1}/n)\ (1+\varepsilon_n)$ internal nodes.

Let us select a parameter k, where $1 \leqslant k \leqslant n$. Each function of n variables can be uniquely represented as follows [33]:

$$f(X) = \sum_{\alpha} x_1^{\alpha_1} \cdots x_k^{\alpha_k} \cdot f_\alpha(x_{k+1}, \ldots, x_n), \tag{6}$$

where $\alpha = \alpha_1, \ldots, \alpha_k$ ranges over the 2^k possible assignments to variables $x_1, \ldots, x_k$. The distinct functions f_α occurring in this representation can be enumerated as f_i where the number of such functions f_i is at most max $(2^k, 2^{2^{n-k}})$. Based on representation (6) we construct an OBDD A computing f as follows.

1. Block $B1$ computes all possible conjunctions of length k from variables $x_1, \ldots, x_k$. This block is a k-level tree. Each node of the ith level is labelled with variable x_i, where $1 \leqslant i < k$. Therefore,

$$L(B1) = \sum_{i=0}^{k-1} 2^i = 2^k - 1.$$

2. Block $B2$ using a conjunction computed by block $B1$, computes a function $f_i(x_{k+1}, \ldots, x_n)$, for each f_i from the representation (6). Block $B2$ can be constructed to have at most one internal node for each of the $2^{2^{n-k}} - 2$ nonconstant functions of variables $x_{k+1}, \ldots, x_n$. Suppose a given function of $x_{k+1}, \ldots, x_n$ depends essentially on x_h, but does not depend essentially on any x_j, where $j < h$. Then this function is represented in $B2$ by a node labeled by variable x_h. Thus, the number of internal nodes in $B2$ is at most $2^{2^{n-k}} - 2$. An alternate way to see this is that the number of nodes labeled with x_{k+1} is the number of functions $n-k$ variables that depend essentially on x_{k+1} (i.e. $2^{2^{n-k}} - 2^{2^{n-k-1}}$), the number of nodes labeled by x_{k+2} is $2^{2^{n-k-1}} - 2^{2^{n-k-2}}, \ldots$, and the number of nodes labeled by x_n is $2^{2^1} - 2$. Therefore,

$$L(B2) \leqslant 2^{2^{n-k}} - 2.$$

From the representation (6) we derive that the constructed OBDD computes Boolean function f. The size of the OBDD is a sum of $L(B1)$ and $L(B2)$. Thus, we obtain

$$L(A) = L(B1) + L(B2) \leqslant 2^k + 2^{2^{n-k}} - 3$$

Let us choose $k = n - \lfloor \log(n - 2\log n) \rfloor$. Then $k < n - \log(n - 2\log n) + 1$ and $n - k \leqslant \log(n - 2\log n)$, so we derive

$$L(A) < \frac{2^{n+1}}{n - 2\log n} + \frac{2^n}{n^2}.$$

This yields

$$L(A) \leqslant \frac{2^{n+1}}{n}(1 + \varepsilon_n),$$

where $\varepsilon_n \to \infty$ as $n \to \infty$. Thus the upper bound is proven.

Lower bound: Since every OBDD is also a BDD, the lower bound from Theorem 1 also applies to OBDDs. □

In Theorem 1, we showed that asymptotically $L_{\mathrm{BDD}}(n)$ is $2^n/n$ and in Theorem 2 we showed that $L_{\mathrm{OBDD}}(n)$ is asymptotically between $2^n/n$ and $2^{n+1}/n$. This raises the issue of exactly characterizing $L_{\mathrm{OBDD}}(n)$, asymptotically. The upper bound construction in Theorem 1 produces a BDD that is not free because blocks $B1$ and $B3$ involve the same variables. However, suppose the value of s in the construction is a power of 2, say $s = 2^q$. Then of the k variables that determine the row number, the first $k - q$ variables determine the row group, and the remaining q variables determine the row within the group. Thus Block $B1$ can determine the group number by testing variables $x_1, \ldots, x_{k-q}$. Furthermore, block $B3$ need not test these variables again; it can determine the relative row number within the row group by testing variables $x_{k-q+1}, \ldots, x_k$. Thus, when s is a power of 2, the constructed BDD is an OBDD. However, the construction sets $s = n - 6\lfloor \log n \rfloor$, and thus produces an OBDD only for a certain values of n.

The upper bound construction of Theorem 2 always produces an OBDD, but the asymptotic size of the constructed OBDD is often greater than $2^n/n$. However, for any $\varepsilon > 0$ there are infinite number of values of n for which $\log(n - 2\log n) - \lfloor \log(n - 2\log n) \rfloor < \varepsilon$, so that the constructed OBDD A is close to the right size:

$$L(A) < \frac{2^{n+\varepsilon}}{n - 2\log n} + \frac{2^n}{n^2}.$$

Thus the construction of Theorem 1 always produces a BDD whose size is right, but only occasionally produces a BDD that is ordered, whereas the construction of Theorem 2 always produces an OBDD, but only sometimes produces one whose size is almost right. Perhaps some merger of the constructions can be used to more exactly characterize $L_{\mathrm{OBDD}}(n)$.

4. Comparative sizes of BDTs and OBDDs

In this section we consider the relative succinctness of OBDDs and BDTs. It is well known that $L_{\mathrm{BDT}}(n) = 2^n - 1$. To see that this is an upper bound, consider any

Boolean function f of n variables, say $x_1, x_2, \ldots, x_n$. A complete binary tree T_f can be constructed whose nodes at level l are labeled with x_l, and whose bottom level contains terminal nodes. The number of internal nodes of this tree, $L(T_f)$, is $\sum_{l=0}^{n-1} 2^l = 2^{n-1}$. To see that it is a lower bound, note that for the parity function of n variables, each BDT path to a terminal node must test all the variables. Consequently, a BDT for the parity function must have at least 2^n terminal nodes, and at least $2^n - 1$ internal nodes.

It is easy to see that the parity function of n variables can be computed by an OBDD with $2n - 1$ internal nodes [36]. Thus the parity functions are a family of Boolean functions $\{f_n \mid n \geqslant 1\}$ such that f_n depends on n variables and

$$\frac{L_{\mathrm{BDT}}(f_n)}{L_{\mathrm{FBDD}}(f_n)} = \frac{2^n - 1}{2n - 1},$$

which is $2^{n-1}/n$ asymptotically. It is much less obvious that there are Boolean functions f_n that have very large minimal size OBDDs and small size BDTs. We prove that this is so in the next theorem.

Theorem 3. *For each n of the form 2^k,* where $k \geqslant 1$, *there is a Boolean function f_n of n variables such that*

$$\frac{L_{\mathrm{OBDD}}(f_n)}{L_{\mathrm{BDT}}(f_n)} \geqslant \frac{2^{(n/\log n)-4}}{n^2/\log n}.$$

Proof. We specify a Boolean function f_n of $n = 2^k$ variables such that any OBDD computing f_n has size at least $2^{(n/\log n)-3}$, but there is a BDT computing f_n with size $2n^2/\log n$. Function f_n is defined as follows. Let f_n have n variables labeled $x_0, \ldots, x_{2^k-1}$. Let $m = \lfloor 2^k/k \rfloor$. The 2^k variables can be envisioned as partitioned into m groups, each consisting of k variables, with possibly some variables left over. Let group 0 contain x_0 contain x_0 through x_{k-1}, group 1 contains x_k through x_{2k-1}, etc. Let group 0 be termed the *selection group.* Let groups 1 through $m - 1$ be termed *candidate groups.* Let N_j be the integer represented by group j when the k variables in group j are interpreted as the binary encoding of an integer. Thus, $0 \leqslant N_j \leqslant 2^k - 1$. When the integer N_0 represented by the selection group is between 1 and $m - 1$, it can be envisioned as the index of one of the candidate groups. Let this candidate group be termed the *selected group.* The selected group can in turn be envisioned as encoding the identity of one of the 2^k variables. Let this variable be called the *selected variable.* Then, the value of function f_n is defined to be the value of the selected variable. Thus, when $0 < N_0 < m$, f_n is defined to be

$$f_n(x_0, \ldots, x_{2^k-1}) = x_{N_{N_0}}.$$

Otherwise (when $N_0 = 0$ or $N_0 \geqslant m$), f_n is defined to equal 0.

Consider an OBDD A that computes f_n, using total order Π. Consider the position in Π immediately after encountering $m - 2$ variables from candidate groups. At this point in order Π, there is at least one candidate group, say group j, such that none of the variables in group j have been encountered yet. Suppose the selected group is indeed group j. Then the variables in group j might potentially make any of the $m-2$ already encountered candidate group variables the selected variable. Thus A must have at least 2^{m-2} internal nodes in order to remember the value of these $m-2$ variables. More formally, consider the 2^{m-2} partial assignments to the variables of f_n, where the variables in group 0 have values encoding integer j, and the first $m-2$ candidate group variables in Π have all 2^{m-2} possible assignments. Each of these partial assignments makes f_n a function of the remaining candidate group variables. Furthermore, each of these subfunctions is a distinct Boolean function. For, if two partial assignments differ in variable i, then if the variables in group j have values such that $N_j = i$, the value of f_n would be different. Thus, these partial assignments correspond to 2^{m-2} distinct Boolean functions of the untested variables. As a consequence, OBDD A must have at least 2^{m-2} internal nodes. Since $m = \lfloor n/\log n \rfloor$, and $\lfloor n/\log n \rfloor > (n/\log n) - 1$, A must have at least $2^{(n/\log n)-3}$ internal nodes.

Now consider the following BDT, tree T, for computing function f_n. Tree T has an upper portion consisting of a complete binary tree testing all the variables in group 0, the selection group. Let the exiting arcs of this upper tree be labeled 0 through $n-1$, so that arc j corresponds to the case when $N_0 = j$. Consider arc j. If $j = 0$ or $j \geqslant m$, then arc j enters a terminal node labeled with value 0. If $0 < j < m$, then arc j enters the root of a complete binary tree that tests all the variables in group j. Each of the n exiting arcs of a given lower tree, say lower tree j, corresponds to a possible value of N_j. An exiting arc corresponding to a variable in group 0 or in group j refers to an already tested variable, and so enters a terminal node. Each of the remaining $n-2k$ exiting arcs enters an internal node labeled by variable x_{N_j}, the exiting arcs of which enter terminal nodes.

Now consider the number of internal nodes of tree T. The upper tree contains $n-1$ internal nodes. There are $m-1$ lower trees, each with $n-1$ internal nodes labeled by variables in its group, and leading to $n-2k$ internal nodes. The total number of internal nodes in tree T is

$$n - 1 + (m-1)(2n-2k-1) < 2nm < \frac{2n^2}{\log n}. \qquad \square$$

5. Construction

By Lemma 4, to establish exponential lower bounds for Boolean functions computed by RBDD, FBDDs or OBDDs, it would be sufficient to exhibit a family of Boolean functions $\{f_n \mid n \geqslant 1\}$ such that each function f_n has exponential in n the number of subfunctions obtained from f_n by substituting constants instead of any of its m variables, where $m < n$ and each such subfunction depends essentially on all its remaining variables.

In [8–10] we exhibit a family of Boolean functions $\{f_n \mid n \geqslant 1\}$ such that by substituting instead of any of its $n/2$ variables we obtain $2^{n/8}$ different subfunctions. In [11] we exhibit a different family of Boolean functions $\{f_n \mid n \geqslant 1\}$ such that by substituting instead of any of its $n/3$ variables we obtain $2^{n/6}$ different subfunctions. From those results it follows that $L_{\mathrm{RBDD}}(f_n) > 2^{n/6}$.

In this Section we exhibit a family $\{f_n \mid n \geqslant 12\}$ of Boolean functions such that by substituting instead of any its $n/3$ variables we obtain $2^{n/3}$ different subfunctions each depending essentially on its remaining variables.

We show the construction for $N = 3n$. Let X, Y, Z be three disjoint sets of variables, each containing n variables, where $n \geqslant 4$. We define an auxiliary function $f^1(X, Y, Z)$ as follows: Function f^1 is equal to the value of the variable in group X with an index one less than the number of ones mod $(n + 1)$ among variables in groups Y and Z. It is easy to see that

$$f^1(X, Y, Z) = \sum_{i=1}^{n} x_i \, (S_{2n}^{i-1}(Y, Z) + S_{2n}^{n+i}(Y, Z)). \tag{7}$$

Let $f^2(X, Y, Z) = f^1(Y, Z, X)$; $f^3(X, Y, Z) = f^1(Z, X, Y)$. It is easy to see that $f^1(f^2$, and f^3, respectively) is symmetric in (Y, Z) $((X, Z)$, (X, Y), respectively) and linear in X $(Y, Z$, respectively). Thus, by Lemma 3, it depends essentially on all of its variables.

Let function $F(X, Y, Z)$ be a Boolean function of $3n$ arguments defined as follows:

$$F(X, Y, Z) = f^1 + f^2 + f^3, \tag{7a}$$

In the rest of the section we prove that for any n selected variables of F, the functions obtained from F by substituting different n-tuples α and β for the selected variables, are different and depend essentially on all its nonselected variables. To do that we need first to establish some structural properties of F that are formulated in several next Lemmas.

Let us choose x^*, y^*, z^* where x^* contains k_1 variables chosen from X, y^* contains k_2 variables chosen from Y, and z^* contains k_3 variables chosen from Z, where $k_1 + k_2 + k_3 = n$. Let $x' = X - x^*$, $y' = Y - y^*$ and $z' = Z - z^*$. It is easy to see that x' contains $(k_2 + k_3)$ variables, y' contains $(k_1 + k_3)$ variables, and z' contains $(k_1 + k_2)$ variables.

Let $\alpha = \alpha_{x^*} \alpha_{y^*} \alpha_{z^*}$ and $\beta = \alpha_{x^*} \beta_{y^*} \beta_{z^*}$ be two different n-tuples. Let $\alpha_{x^*} \alpha_{y^*} (\beta_{x^*} \beta_{y^*})$ contain $t_{11} (t_{21})$ ones, $\alpha_{x^*} \alpha_{z^*} (\beta_{x^*} \beta_{z^*})$ contain $t_{12} (t_{22})$ ones and $\alpha_{y^*} \alpha_{z^*} (\beta_{y^*} \beta_{z^*})$ contain $t_{13} (t_{23})$ ones. Without loss of generality, let us assume that $t_{13} \leqslant t_{23}$ (or $t_{12} \leqslant t_{22}$, or $t_{11} \leqslant t_{21}$, respectively)

Since $\alpha_{y^*} \alpha_{z^*} (\alpha_{x^*}, \alpha_{z^*}, \alpha_{x^*}, \alpha_{y^*}$, respectively) contains $t_{13} (t_{12}, t_{11}$, respectively) ones, using representation (7) and (1a), we obtain (8) given below.

$$\begin{aligned} f^{1, \alpha_{y^*} \alpha_{z^*}} = {} & x_{t_{13}+1} S_{n+k_1}^{0} + x_{t_{13}+2} S_{n+k_1}^{1} + \cdots + x_n S_{n+k_1}^{n-t_{13}-1} \\ & + x_1 S_{n+k_1}^{n-t_{13}+1} + \cdots x_{t_{13}} S_{n+k_1}^{n} + \cdots + x_{k_1+t_{13}} S_{n+k_1}^{n+k_1}. \end{aligned} \tag{8}$$

Similar expressions can be easily obtained for $f^{2, \alpha_{x^*} \alpha_{z^*}}$ and for $f^{3, \alpha_{x^*} \alpha_{y^*}}$.

Lemma 5. *$f^{1,\alpha}$ depends essentially on all of its variables.*

Proof. From representation (8) it follows that each variable from x' is present as a coefficient for some $S^i_{n+k_1}$ $(0 \leqslant i \leqslant n)$. The lemma's assertion follows now from Lemma 3. □

Similar lemmas hold for $f^{2,\alpha}$ and $f^{3,\alpha}$. From representation (8) by an appropriate change in the summation variable in (8) and assuming that variable indices are interpreted by modulo $(n + 1)$, we obtain (9) and analogous equations give representations (10) and (11) below:

$$f^{1,\alpha_{y^*}\alpha_{z^*}} + f^{1,\beta_{y^*}\beta_{z^*}} = \sum_{i=t_{13}+1}^{n+k_1+t_{13}+1} (x_i + x_{i+t_{23}-t_{13}})\, S^{i-t_{13}-1}_{n+k_1} \quad \text{when } t_{13} \leqslant t_{23}, \tag{9}$$

$$f^{2,\alpha_{x^*}\alpha_{z^*}} + f^{2,\beta_{x^*}\beta_{z^*}} = \sum_{i=t_{12}+1}^{n+k_2+t_{12}+1} (y_i + y_{i+t_{22}-t_{12}})\, S^{i-t_{12}-1}_{n+k_2} \quad \text{when } t_{12} \leqslant t_{22}, \tag{10}$$

$$f^{3,\alpha_{x^*}\alpha_{y^*}} + f^{3,\beta_{x^*}\beta_{y^*}} = \sum_{i=t_{11}+1}^{n+k_3+t_{11}+1} (z_i + z_{i+t_{21}-t_{11}}) S^{i-t_{11}-1}_{n+k_3} \quad \text{when } t_{11} \leqslant t_{21}, \tag{11}$$

where variable indices are taken modulo $(n + 1)$ and x_0 is equal to 0, by definition. Analogous representations hold when $t_{13} > t_{23}$, $t_{12} > t_{22}$, or $t_{11} > t_{21}$, respectively.

Lemma 6. *If $t_{13} \neq t_{23}$, then $f^{1,\alpha} + f^{1,\beta}$ depends essentially on all variables from x'. If $t_{13} = t_{23}$, then $f^{1,\alpha} + f^{1,\beta}$ does not depend essentially on any variables from x'.*

Proof. Without loss of generality, assume $t_{13} \leqslant t_{23}$. Using representation (9) for $f^{1,\alpha} + f^{1,\beta}$ and substituting α_{x^*} and β_{x^*} for variables from x^* in the first and second addends, respectively, we obtain that

$$f^{1,\alpha} + f^{1,\beta} = \sum_{i=0}^{n+k_1} L_i S^i_{n+k_1}. \tag{12}$$

If $t_{12} = t_{23}$, then from (9) we obtain each L_i in (12) is a constant and therefore $f^{1,\alpha} + f^{1,\beta}$ does not depend on any variable from x'.

If $t_{13} \neq t_{23}$, then i and $i + t_{23} - t_{13}$ are distinct values modulo $(n + 1)$. Therefore, in (9) the coefficient of each S is the sum of two different variables from x'. Note that each variable from x' is a part of the coefficient of at least one of the S's. Therefore (9) leads to representation (12) of $f^{1,\alpha} + f^{1,\beta}$ in which regardless of what the selected k_1 variables from x are, for each variable in x', the coefficient of at least one of the elementary symmetric functions of y' and z', and thus depends essentially on that variable. Therefore, by Lemma 3, $f^{1,\alpha} + f^{1,\beta}$ depends essentially on all variables from x'. □

Similar lemmas hold for $f^{2,\alpha} + f^{2,\beta}$ and $f^{3,\alpha} + f^{3,\beta}$.

Lemma 7. *If $F^\alpha = F^\beta$, then $f^{1,\alpha} + f^{1,\beta}$ is symmetric in x'.*

Proof. Since $F^\alpha + F^\beta = 0$, it is symmetric in x'. Since both $f^{2,\alpha} + f^{2,\beta}$ and $f^{3,\alpha} + f^{3,\beta}$ are symmetric in x', it follows that $f^{1,\alpha} + f^{1,\beta}$ is symmetric in x'. □

Lemma 8. *If $t_{13} \neq t_{23}$ and x' contains at least two variables, then $f^{1,\alpha} + f^{1,\beta}$ is not symmetric in x'.*

Proof. From (9) we conclude that $f^{1,\alpha} + f^{1,\beta}$ is linear in x'. Let us assume that it is also symmetric in x'. Then by Lemma 1, we derive that

$$f^{1,\alpha} + f^{1,\beta} = \Phi_1(y',z') + \Phi_2(y',z')\left(\sum_{x_i \in x'} x_i\right), \tag{13}$$

where, from Lemma 6, Φ_2 is not the constant 0. On the other hand, considering (9), we obtain that each coefficient for $S_{n+k_1}^{i-t_{13}-1}$ involves no more than two variables from x'. If x' contains more than two variables, then (13) contradicts (9).

Suppose x' contains exactly two variables. Let i be the smallest value $\geqslant t_{13} + 1$ such that $i \bmod (n+1)$ is the index of a variable in x'. Since representation (9) contains the term

$$(x_i + x_{i+t_{23}-t_{13}})S_{n+k_1}^{i-t_{13}-1},$$

symmetry in x' implies that the index of the other variable in x' is $i + t_{23} - t_{13} \bmod (n+1)$. But representation (9) also contains the term

$$(x_{i+t_{23}-t_{13}} + x_{i+2(t_{23}-t_{13})})S_{n+k_1}^{i+t_{23}-2t_{13}-1}.$$

Since x' only contains two variables, we obtain $i + 2(t_{23} - t_{13}) \bmod (n+1) = i \bmod (n+1)$, i.e. $2(t_{23} - t_{13}) \bmod (n+1) = 0$. But since x^* contains $n-2$ variables,, $t_{23} - t_{13}$ is at most 2. Since $n \geqslant 4$, we have a contradiction. □

Lemma 9. *If $t_{13} = t_{23}$, $k_1 < \lfloor n/2 \rfloor$, and $f^{1,\alpha} + f^{1,\beta}$ is not a constant function, then it is a non-linear function.*

Proof. From (9), we obtain that, since $t_{13} = t_{23}$,

$$\begin{aligned} f^{1,\alpha_{y*}\alpha_{z*}} + f^{1,\beta_{y*}\beta_{z*}} = {} & x_1 S_{n+k_1}^{n+1-t_{13}} + x_2 S_{n+k_1}^{n+2-t_{13}} + \cdots + x_{t_{13}} S_{n+k_1}^{n} \\ & + x_{t_{13}+1}(S_{n+k_1}^{0} + S_{n+k_1}^{n+1}) + \cdots + x_{t_{13}+k_1}(S_{n+k_1}^{k_1-1} + S_{n+k_1}^{n+k_1}) \\ & + x_{t_{13}+k_1+1} S_{n+k_1}^{k_1} + \cdots + x_n S_{n+k_1}^{n-t_{13}+1}. \end{aligned}$$

Consequently,

$$\begin{aligned} f^{1,\alpha} + f^{1,\beta} = {} & \gamma_1 S_{n+k_1}^{n+1-t_{13}} + \gamma_2 S_{n+k_1}^{n+2-t_{13}} + \cdots + \gamma_{t_{13}} S_{n+k_1}^{n} \\ & + \gamma_{t_{13}+1}(S_{n+k_1}^{0} + S_{n+k_1}^{n+1}) + \cdots + \gamma_{t_{13}+k_1}(S_{n+k_1}^{k_1-1} + S_{n+k_1}^{n+k_1}) \\ & + \gamma_{t_{13}+k_1+1} S_{n+k_1}^{k_1} + \cdots + \gamma_n S_{n+k_1}^{n-t_{13}+1}, \end{aligned}$$

where γ_i is 1 if x_i is in x^* and variable x_i has different values in α and β, and is 0 otherwise.

Note that for each i, $1 \leqslant i \leqslant n$, there is a j, $0 \leqslant j \leqslant n + k_1$, such that the coefficient of $S^j_{n+k_1}$ is γ_i. From Corollary 1, the only linear symmetric nonconstant functions are the even and odd parity functions. Therefore, for $f^{1,\alpha} + f^{1,\beta}$ to be a linear nonconstant function, at least $\lfloor n/2 \rfloor$ of the γ_i must equal 1. Since $k_1 < \lfloor n/2 \rfloor$, it follows that $f^{1,\alpha} + f^{1,\beta}$ is nonlinear. □

Lemma 10. *Let $S_1(x, y)$ be a symmetric nonlinear function of its variables, and let $S_2(x, z)$ be a symmetric function of its variables, where x, y and z are disjoint nonempty groups of variables. Then $S_1(x, y) + S_2(x, z)$ depends essentially on x and is symmetric in x.*

Proof. Since both S_1 and S_2 are symmetric in x, their sum $S_1 + S_2$ is also symmetric in x.

Since S_1 is both symmetric and nonlinear, its ring sum form contain at least one product involving variables from both x and y. Since this product does not appear in the ring sum form of S_2, the product appears in the ring sum form of $S_1 + S_2$. This implies that function $S_1 + S_2$ depends essentially on each variable from x appearing in this product. Then, by Lemma 1, $S_1 + S_2$ depends essentially on every variable in x. □

Theorem 4. *$F^\alpha = F^\beta$, then $\alpha = \beta$.*

Proof. Assume that $F^\alpha = F^\beta$. Let us consider several cases.

Case 1: *There is a k_i such that $k_i = n$.* Without loss of generality, let us assume that $k_1 = n$, $k_2 = 0$, $k_3 = 0$. By the definition of f^1, $f^{1,\alpha} + f^{1,\beta} = \sum \gamma_i S^i_{2n}(Y, Z)$, where each γ_i is a constant. Similarly,

$$f^{2,\alpha} + f^{2,\beta} = \sum L_i(Y) S^i_n(Z), \qquad f^{3,\alpha} + f^{3,\beta} = \sum L'_i(Z) S^i_n(Y),$$

where $L_i(Y)$ and $L'_i(Z)$ are either sums of two variables, or only a single variable, or identically 0.

First, consider the case when $t_{11} = t_{21}$. Since no variables from Z and Y are selected, $t_{12} = t_{22} = t_{13} = t_{23} = 0$. From (10) and (11) we obtain that

$$f^{2,\alpha} + f^{2,\beta} = 0 \quad \text{and} \quad f^{3,\alpha} + f^{3,\beta} = 0.$$

Since $F^\alpha = F^\beta$, we obtain $f^{1,\alpha} = f^{1,\beta}$. Thus each $\gamma_i = 0$, where $0 \leqslant i \leqslant 2n$. Since $t_{13} = t_{23} = 0$, we obtain from (9) that $\gamma_i = \alpha_{i+1} + \beta_{i+1}$. Thus $\alpha_i = \beta_i$ for each x_i in x^*. Therefore $\alpha = \beta$.

Let us now consider the case where $t_{11} \neq t_{21}$. Note that $f^{2,\alpha} + f^{2,\beta}$ is linear in Y, and by Lemma 6, depends essentially on each variable from Y. Therefore at least one $L_i(Y)$ is not identically 0.

$L_i(Y)$ is either a sum of two variables, or contains exactly one variable. Since Y contains n variables and $n \geqslant 4$, the function $f^{2,\alpha} + f^{2,\beta}$ is not symmetric in Y, by Lemma 1. But since $F^\alpha = F^\beta = 0$, Lemma 7 applies, so $f^{2,\alpha} + f^{2,\beta}$ is symmetric in Y, a contradiction.

Case 2. *There is a k_i such that $k_i = n - 1$.* Without loss of generality, let us assume that $k_1 = n - 1$, $k_2 = 1$ and $k_3 = 0$. Let x_j be the variable from X that was not selected. By Lemma 6, we obtain that in case $t_{13} = t_{23}$:

$$f^{1,\alpha} + f^{1,\beta} = \sum \gamma_i S^i_{2n-1}(y', Z),$$

where γ_i are constants; or, in case $t_{13} \neq t_{23}$,

$$f^{1,\alpha} + f^{1,\beta} = x_j R_{11}(y', Z) + \sum \delta_i S^i_{2n-1}(y', Z),$$

where δ_i are constants and R_{11} is a symmetric function that is equal to the sum of no more than 4 elementary symmetric functions, (see (9)).

Also, note that

$$f^{2,\alpha} + f^{2,\beta} = \sum L_i(y') S^i_{n+1}(x_j, Z),$$

$$f^{3,\alpha} + f^{3,\beta} = \sum L'_i(Z) S^i_n(x_j, y'), \tag{14}$$

where each $L_i(y')$ and $L'_i(Z)$ are linear functions of no more than 2 variables. Let us consider several cases.

Subcase 2.1: $t_{11} \neq t_{21}$. By Lemma 6, $f^{3,\alpha} + f^{3,\beta}$ depends essentially on each variable from z. From (14), it follows that $f^{3,\alpha} + f^{3,\beta}$ is not symmetric in z. From Lemma 7, we obtain that $F^\alpha = F^\beta$ implies $f^{3,\alpha} + f^{3,\beta}$ is symmetric in Z, a contradiction.

Subcase 2.2: $t_{12} \neq t_{22}$. Arguments similar to those for case 2.1 hold, with replacement of f^3 by f^2. Thus, there is a contradiction in this case.

Subcase 2.3: $t_{11} = t_{21}$ *and* $t_{12} = t_{22}$. Then the value of the single variable selected from y is the same in α and β, so we also have $t_{13} = t_{23}$. From representation (10), we obtain that $f^{2,\alpha} + f^{2,\beta}$ is identically 0. Similarly, from representation (11), $f^{3,\alpha} + f^{3,\beta}$ is identically 0.

Thus, $F^\alpha + F^\beta = f^{1,\alpha} + f^{1,\beta}$. Since $F^\alpha = F^\beta$, we have $f^{1,\alpha} + f^{1,\beta} = 0$. Since $t_{13} = t_{23}$, Eq. (16) holds, where each γ_i in (14) equals 0. From representation (9), this implies that α and β have the same value for each variable in x^*. Thus $\alpha = \beta$.

Case 3: *No k_i exceeds $n - 2$.* In this case we obtain that each group x', y' and z' contain at least two variables. There are two possibilities: there is a j such that $t_{1j} \neq t_{2j}$, or $t_{1j} = t_{2j}$ for all $1 \leqslant j \leqslant 3$.

Let us consider the first possibility. Without loss of generality, assume that $t_{13} \neq t_{23}$. By Lemma 8, we obtain that $f^{1,\alpha} + f^{1,\beta}$ is not symmetric in x'. From Lemma 7, this contradicts $F^\alpha = F^\beta$.

Let us now consider the second possibility, that is $t_{11} = t_{21}$, $t_{12} = t_{22}$, and $t_{13} = t_{23}$. By Lemma 6, $f^{1,\alpha} + f^{1,\beta}$ does not depend essentially on x'. It cannot be identically 1, since it is equal to 0 for any tuple where y' and z' contain $(n - t_{13})$ ones (see representation (9)). Thus, $f^{1,\alpha}+f^{1,\beta}$ is either identically 0 or a nonconstant symmetric function of y' and z'.

By the same token, $f^{2,\alpha}+f^{2,\beta}$ and $f^{3,\alpha}+f^{3,\beta}$ is identically 0 or nonconstant symmetric functions of (x',z') and (x',y'), respectively.

Let us assume that neither $f^{1,\alpha}+f^{1,\beta}$ nor $f^{2,\alpha}+f^{2,\beta}$, nor $f^{3,\alpha}+f^{3,\beta}$ is identically 0. Since $k_1+k_2+k_3=n$ and $n \geqslant 4$, there is a k_i such that $k_i < \lfloor n/2 \rfloor$. Without loss of generality, assume $k_3 < \lfloor n/2 \rfloor$. Then from Lemma 9, $f^{3,\alpha}+f^{3,\beta}$ is nonlinear. Thus, from Lemma 10, the function $(f^{2,\alpha}+f^{2,\beta})+(f^{3,\alpha}+f^{3,\beta})$ depends essentially on the variables from x'. But since $F^\alpha = F^\beta$, we have $f^{1,\alpha}+f^{1,\beta}=f^{2,\alpha}+f^{2,\beta}+f^{3,\alpha}+f^{3,\beta}$. Since $f^{1,\alpha}+f^{1,\beta}$ does not depend essentially on x', we have a contradiction.

Thus at least one of $f^{1,\alpha}+f^{1,\beta}$ or $f^{2,\alpha}+f^{2,\beta}$ or $f^{3,\alpha}+f^{3,\beta}$ is identically 0, say $f^{1,\alpha}+f^{1,\beta}$. Then $f^{2,\alpha}+f^{2,\beta} = f^{3,\alpha}+f^{3,\beta}$. Since $f^{2,\alpha}+f^{2,\beta}$ is symmetric in z', and since $f^{3,\alpha}+f^{3,\beta}$ does not depend on z', we obtain that $f^{2,\alpha}+f^{2,\beta}=0$ and $f^{3,\alpha}+f^{3,\beta}=0$.

Therefore each of $f^{1,\alpha}+f^{1,\beta}$, $f^{2,\alpha}+f^{2,\beta}$, and $f^{3,\alpha}+f^{3,\beta}$ is identically 0. From Lemma 3, the coefficient of each $S_{n+k_1}^{i-t_{13}-1}$ in representation (9) equals 0. Therefore $\alpha_{x^*}=\beta_{z^*}$. Similarly $\alpha_{y^*}=\beta_{y^*}$ and $\alpha_{z^*}=\beta_{z^*}$. We conclude that $\alpha=\beta$. □

6. Comparative complexity of BDD classes

In the next three theorems we compare the relative economy of description of Boolean functions by BDDs, RBDDs, FBDDs and OBDDs. We prove that each class is exponentially more complex comparatively with its superset in this sequence. At the same time, for RBDDs, FBDDs and OBDDs we also establish the largest lower bound known to date. First we compare relative economy of computation by BDDs and RBDDs.

Theorem 5. *For each $n \geqslant 4$, there is a Boolean function F of $3n$ variables, such that any* RBDD *computing F requires at least 2^n internal nodes, and there is a* BDD *that computes F with* $O(n^2)$ *nodes.*

Proof. Consider function F defined by Eq. (7a). From Theorem 4, it follows that for any n variables selected from the $3n$ variables of the function F, if n-tuples, $\alpha \neq \beta$, then $F^\alpha \neq F^\beta$. Furthermore, from Lemma 5. F^α and F^β depend essentially on all the remaining $2n$ variables. Therefore, from Theorem 4 and from Lemma 4, any RBDD computing F requires 2^n internal nodes.

To complete the proof we design a BDD for F with $10n^2+10n$ nodes. First note that for p variables $u=\{u_1,u_2,\dots,u_p\}$, a subBDD can be constructed consisting of $p(p+1)/2$ internal nodes, such that for each exiting edge of the subBDD, all assignments to u which cause the subBDD to be exited via that edge have the same number of 1's. The subBDD consists of p levels, where level i, for $1 \leqslant i \leqslant p$, consists of a sequence of i nodes each of which is labeled by variable u_i. Let the nodes at level i be denoted as $n_{i,j}$, where $0 \leqslant j < i$. For $1 \leqslant i < p$, let the 0-edge exiting $n_{i,j}$ enter $n_{i+1,j}$, and let the 1-edge exiting $n_{i,j}$ enter $n_{i+1,j+1}$. Using this scheme, node $n_{i,j}$ is entered only for those assignments in which the number of 1's among variables $u_1, \ldots, u_{i-1}$ is exactly j. Furthermore, the edge exiting the subBDD for any given assignment to u can be envisioned as representing the number of 1's in u.

A BDD for f^1 can begin with a subBDD which counts the number of 1's among the 2^n variables y and z. The number of 1's from among y and z either determines the value of f^1, or determines that the value of f^1 depends on a single selected variable from x. Thus by adding n interior nodes labeled with variables from x, a subBDD can be constructed that computes the value of f^1. This BDD has $(2n)(2n+1)/2$ nodes labeled with variables from y and z, and n nodes labeled with variables from x, for a total of $2n^2 + 2n$ internal nodes.

A BDD for F can be constructed using a BDD for f^1, two copies of a BDD for f^2, and two copies of a BDD for f^3. The value of f^1 selects one of the copies of the BDD for f^2 . The value of $f^1 + f^2$ selects one of the two copies of the BDD for f^3. The edges exiting each copy of f^3 enter a sink node based on the value of $(f^1 + f^2) + f^3$.

The size of the constructed BDD is 5 times the size of a BDD for f^1, i.e. the size is $10n^2 + 10n$. □

Since every OBDD and FBDD are also RBDD, the lower bound on number of internal nodes to compute F by RBDD also applies to FBDD and OBDD as well. In the next two theorems we exhibit Boolean functions of n variables that have smaller (but still exponential in n) lower bound for a number of internal nodes to be computed by either FBDD or OBDD. However, these functions would only require $O(n^2)$ sizes to be computed by a superset class of BDDs.

Theorem 6. *For every $n \geqslant 4$, there exists a Boolean function Φ of $6n + 4$ variables, such that every* FBDD *computing Φ contains at least 2^n nodes, but there is an* RBDD *computing Φ with no more than $O(n^2)$ nodes.*

Proof. Consider function F defined by Eq. (7a). Let T be a set of $3n + 3$ Boolean variables that are different from any variable of F. In addition, let v be a variable that does not appear either in F or in T. Function Φ is defined as follows:

$$\Phi = \bar{v}F + v \sum_{t \in T} t.$$

Every FBDD computing Φ must also compute F. From Theorem 5, it follows that any FBDD computing F requires 2^n internal nodes. To complete the proof we design BDD A that computes Φ with $10n^2 + 16n + 6$ nodes. BDD A starts computing Φ with variable v. If $v = 0$, then A computes F as described in Theorem 5. If, on the other hand, $v = 1$, then A computes a linear Boolean function of $3n + 3$ variables. Thus,

A has $10n^2 + 16n + 6$ internal nodes. Note, that the depth of any path in BDD computing F does not exceed $6n + 3$. Thus, the total depth of A is $6n + 4$. Furthermore, every variable of Φ labels at least one internal node of A. Thus, A is RBDD. The theorem is proven. □

Finally, we exhibit an example of a Boolean function that requires an exponential number of nodes to be computed by OBDD and $O(n^2)$ nodes to be computed by FBDD.

Theorem 7. *For every $n \geqslant 4$, there exists a Boolean function G of $3n + 2$ variables, such that very* OBDD *computing G contains at least $2^{n/3}$ nodes, but there is an* FBDD *computing F with no more than $3(2n^2 + 2n) + 3$ nodes.*

Proof. Let v and w be two Boolean variables distinct from any variables from X, Y, and Z. Let f_1, f_2 and f_3 be Boolean functions defined in (7).

Let

$$G = \bar{v}\bar{w}f^1(X, Y, Z) + \bar{v}wf^2(X, Y, Z) + v\bar{w}f^3\ (X, Y, Z).$$

Then a FBDD computing G with $3(2n^2 + 2n) + 3$ nodes can be constructed as follows. The FBDD first test v and w and evaluates a subBDD for either f^1, f^2, or f^3. Each of these subBDDs can be constructed, as described in the proof of Theorem 5, as an OBDD.

Let us consider now an arbitrary order O on v, w, X, Y, Z and, without loss of generality, we assume that v, w among first $n + 2$ variables of G in the order O. Let O includes k_1 variables from X, k_2 variables from Y, k_3 variables from Z, where $k_1 + k_2 + k_3 = n$. Then there is i such that $k_i \geqslant n/3$. Without loss of generality, assume that $k_1 \geqslant n/3$. Thus $f^{1,\alpha}$ and $f^{1,\beta}$ for $\alpha\}$ and $\beta\}$ that differ in positions for selected variables from X will be different. Thus, any OBDD computing G will need at least $2^{n/3}$ internal nodes. □

Theorems 5–7 are "close" to being as strong as possible. For example, the family of Boolean functions $\{ f_n \mid n \geqslant 1\}$ in Theorem 7 has the property that, for $n \geqslant 1$, function f_n depends essentially on all n variables and satisfies

$$\frac{L_{\mathrm{OBDD}}(f_n)}{L_{\mathrm{FBDD}}(f_n)} \geqslant c \cdot \frac{2^{n/9}}{n^2}.$$

But in comparison, for all $n \geqslant 1$ and for all Boolean functions f depending essentially upon n variables,

$$\frac{L_{\mathrm{OBDD}}(f)}{L_{\mathrm{FBDD}}(f_n)} \leqslant \frac{2^{n+1}}{n^2} \cdot (1 + \varepsilon_n), \quad \text{where } \varepsilon_n \to 0 \text{ as } n \to \infty.$$

7. Conclusions

In this paper we compare a computational power of BDDs, RBDDs, FBDDs, OBDDs and BDT. Our results show that FBDD can be exponentially more complex than BDD computing the same Boolean function and FBDD can be exponentially more complex than OBDD computing the same Boolean functions. We also show that the computational power of BDTs and OBDDs is generally incomparable.

A technique that we used in obtaining lower bounds on the size of FBDDs and OBDDs for families of functions goes back to [8] and careful application of that technique allows us to obtain the best known to date lower bounds for complexity of Boolean functions on FBDD and OBDD.

We also establish asymptotically sharp bounds as a function of n on the minimum size of arbitrary BDDs representing almost all Boolean functions of n variables and provide asymptotic lower and upper bounds, differing only by a factor of two, on the minimum size OBDDs representing almost all Boolean functions of n variables.

References

[1] M.S. Abadir and H.K. Reghbati, Functional test generation for digital circuits using binary decision diagrams, *IEEE Trans. Comput.* **C-35** (1986) 375–379.

[2] S. Aborhey, Binary decision tree test functions, *IEEE Trans. Comput.* **C-37** (1988) 1461–1465.

[3] M. Ajtai et al. Two lower bounds for branching programs in: *Proc. 18th Ann. ACM Symp. on Theory of Computing* (Berkeley, CA, 1986) 30–38.

[4] S.B. Akers, Binary decision diagrams, *IEEE Trans. Comput.* **C-27** (1978) 509–516.

[5] S.B. Akers, Functional testing with binary decision diagrams, in: *Proc. 8th Ann. Internat. Conf. on Fault-Tolerant Computing* (IEEE New York 1978), 75–82.

[6] L. Babai, P. Hajnal, E. Szeremédi and G. Turán, A lower bound for read-once-only branching programs, *J. Comput. System Sci.* **35** (1987) 153–162.

[7] A. Borodin, D. Dolev, F.E. Fich and W. Paul, Bounds for width two branching programs, *SIAM J. Comput.* **15** (1986) 549–560.

[8] Y. Breitbart, Comparison of complexities of realization of Boolean functions by automata and Turing machines. *Dokl. Acad. Nauk SSSR* **180** (5); an English Translation appears in: *Sovjet Physics Dokl. 13* (1968).

[9] Y. Breitbart, Complexity of the calculation of predicates by finite automata, Doctoral Thesis Technion, Haifa, Israel, 1973.

[10] Y. Breitbart, Some bounds on the complexity of predicate recognition by finite automata, *J. Comput. System Sci* **12** (3) (1976).

[11] Y. Breitbart, A family of Boolean functions with nonlinear formula size, in: *Proc. 16th Allerton Conf. on Communication, Control and Computing* (1978).

[12] R.E. Bryant, Symbolic manipulation of Boolean functions using a graphical representation, in: *Proc. 22nd ACM/IEEE Design Automation Conf.* (1985) 688–694.

[13] R.E. Bryant, Graph-based algorithms for Boolean function manipulation, *IEEE Trans. Comput.* **C-35** (1986) 677–691.

[14] E. Cerny and J. Gecse, Simulation of MOS circuits by decision diagrams, *IEEE Trans. Comput. Aided Design* **CAD-4** (1985) 685–693.

[15] A.K. Chandra, M.L. Furst and R.J. Lipton, Multi-parity protocols, in: *Proc. 15th Ann. ACM Symp. on Theory of Computing* (1983) 94–99.

[16] A. Cobham, The recognition problem for the set of perfect squares, in: *Proc. 7th Symp. on Switching and Automata Theory* (1966) 78–87.

[17] P.E. Dunne, Lower bounds for the complexity of 1-time-only branching programs, Lecture Notes in Computer Science, Vol. 199 (Springer, Berlin/New York, 1985).

[18] M. Fujita, H. Fujisawa and N. Kawoto, Evaluation and improvements of Boolean comparison method based on binary decision diagrams, in: *Proc. IEEE Internat. Conf. on Computer Aided Design ICCAD*-88 (Santa Clara, CA 1988) 2–5.

[]19] Fortune et al., The complexity of equivalence and containment for free single variable program schemes, in: *G. Ausiello and C. Bohm, eds. Proc. Conf. Automata, Languages, and Programming*, Lecture Notes in Computer Science, Vol. 62 (Springer, Berlin, 1978) 227–240.

[20] S.J. Friedman and K.J. Supowit, Finding the optimal variable ordering for binary decision diagrams, *IEEE Trans. Comput.* **39** (1990) 710–713.

[21] J. Hromkovic, M. Krause, C. Meinel and S. Waack, Branching programs provide lower bounds on the areas of multilective deterministic and nondeterministic VLSI circuits, *Inform. and Comput.* **96** (1992) 168–178.

[22] M. Krause, Exponential lower bounds on the complexity of real-time and local branching programs, *J. Inform. Process. Cybernet* **24** (3) 1988).

[23] M. Krause, Lower bounds for depth-restricted branching programs, *Inform. and Comput.* **91** (1991) 1–14.

[24] K. Kriegel and S. Waack, Exponential lower bounds for real-time branching programs, in: *Proc. FCT' 87*, Lecture Notes in Computer Science, Vol. 278 (Springer, Berlin, 1987).

[25] V.A. Kuzmin, Complexity estimate of Boolean function computation by simplest types of binary programs, in: *Methods of Discrete Analysis in Theory of Coding and Schema, Vol. 29* (Novosibirsk, 1976) 11–39.

[26] C. Lee, Representation of switching circuits by binary decision programs, *Bell Systems Techn. J.* **38** (1959) 985–999.

[27] S. Malik, A.R. Wang, R.K. Brayton and A. Sangiovanni-Vincentelli, Logic verification using binary decision diagrams in a logic synthesis environment, *Proc. IEEE Internat. Conf. on Computer Aided Design ICCAD-88* (Santa Clara, CA, 1988) 6–9.

[28] J.S. Matos and J.V. Oldfield, Mapping of binary decision diagrams into silicon, in: *Proc. IEEE Internat. Symp. on Circuits and Systems* (1983) 210–213.

[29] C. Meinel, Restricted branching programs and their computational power in: *Proc. Mathematical Foundations of Computer Science*, Lecture Notes in Computer Science, Vol. 452 (Springer, Berlin, 1990) 61–75.

[30] C. Meinel, Polynomial size Ω – branching progams and their computational power, *Inform. and Comput.* **85** (1992) 163–182.

[31] B.M.E. Moret, Decision trees and diagrams, *ACM Comput. Surveys* **14** (1982) 593–623.

[32] P. Pudlák, A lower bound on complexity of branching programs in: M.P. Chytil and V. Koubek, eds., *Proc. Mathematical Foundations of Computer Science, Vol. 176* (Springer, Berlin, 1984) 480–489.

[33] J. Savage, *The Complexity of Computing* (Wiley, New York, 1976).

[34] I. Wegener, Optimal decision trees and one-time only branching programs for symmetric Boolean functions, *Inform. and Control* **62** (1984) 129–143.

[35] I. Wegener, Time-space trade-offs for branching programs, *J. Comput. System Sci.* **32** (1986) 91–96.

[36] I. Wegener, *The Complexity of Boolean Functions* (Wiley, New York, 1987).

[37] I. Wegener, On the complexity of branching Functions and decision trees for clique functions, *J. Assoc. Comput. Mach.* **35** (1988) 461–471.

[38] S. Zák, An exponential lower bound for one-time-only branching programs, in: M.P. Chytil and V. Koubek, eds., *Proc. Mathematical Foundations of Computer Science*, Lecture Notes in Computer Science, Vol. 176 (Springer, Berlin, 1984) 562–566.