Binary Decision Diagrams

Beate Bollig, Martin Sauerhoff, Detlef Sieling, and Ingo Wegener
FB Informatik, LS2, Univ. Dortmund,
44221 Dortmund, Germany
`lastname@ls2.cs.uni-dortmund.de`

Abstract

Decision diagrams are a natural representation of finite functions. The obvious complexity measures are length and size which correspond to time and space of computations. Decision diagrams are the right model for considering space lower bounds and time-space trade-offs. Due to the lack of powerful lower bound techniques, various types of restricted decision diagrams are investigated. They lead to new lower bound techniques and some of them allow efficient algorithms for a list of operations on boolean functions. Indeed, restricted decision diagrams like ordered binary decision diagrams (OBDDs) are the most common data structure for boolean functions with many applications in verification, model checking, CAD tools, and graph problems. From a complexity theoretical point of view also randomized and nondeterministic decision diagrams are of interest.

1 Introduction

Let $f\colon D^n \to R$ be a finite function, i.e., D and R are finite sets. Such a function can be represented by the table of all $(a, f(a))$, $a \in D^n$, which always has an exponential size of $|D|^n$. Therefore, we are interested in representations which for many important functions are much more compact. The best-known representations are circuits and decision diagrams. Circuits are a hardware model reflecting the sequential and parallel time to compute $f(a)$ from a (see Chapter **??**). *Decision diagrams* (DDs), also called *branching programs* (BPs), are nonuniform programs for computing $f(a)$ from a based on only two types of instructions represented by nodes in a graph (see also Figure 1):

– *decision nodes*: depending on the value of some input variable x_i the next node is chosen,

– *output nodes* (also called *sinks*): a value from R is presented as output.

A decision diagram is a directed acyclic graph consisting of decision nodes and output nodes. Each node v represents a function f_v defined in the following way. Let $a =$

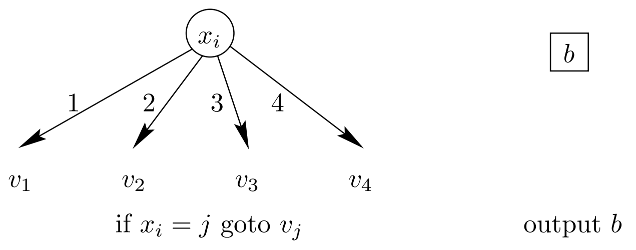

Figure 1: A decision node for $D = \{1, 2, 3, 4\}$ and an output node.

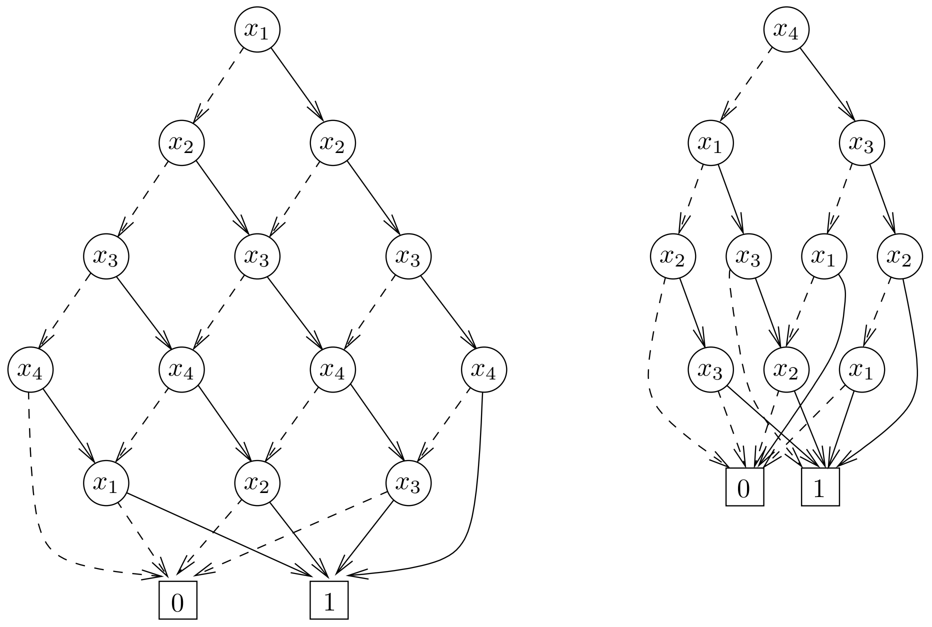

Figure 2: Two DDs for HWB on four boolean variables, dotted lines represent edges labeled by 0 and solid ones edges labeled by 1.

$(a_1, \ldots, a_n) \in D^n$. At decision nodes choose the next node as described above. The value of $f_v(a)$ is defined as the value of the output node which is finally reached when starting at v. Hence, for each node each input $a \in D^n$ activates a unique *computation path* which we follow during the computation of $f_v(a)$. An edge $e = (v, w)$ of the diagram is called *activated* by a if the computation path starting at v runs via e. A graph-theoretical path is called *inconsistent* if it is not activated by any input.

The obvious resources consumed by the DD are the *size* of the DD, defined as the number of its nodes, and the *length* of the DD, defined as the length of the longest computation path of the DD. The size and the length of a function f are defined as the smallest size and length, resp., of a DD representing f at some node. As an example, we consider the function *hidden weighted bit,* $\mathrm{HWB}_n\colon \{0,1\}^n \to \{0,1\}$, defined by $\mathrm{HWB}_n(x) = x_s$ where s is the number of ones in the input and $x_0 = 0$. Figure 2 shows two DDs for HWB_4.

It is obvious that each function $f\colon D^n \to R$ can be represented by a DD of length n implying that DDs are not adequate to prove time lower bounds. In Section 6, we investigate the relationship between DD-based complexity measures and other complexity measures. Then it turns out that the logarithm of the DD size is closely related to the space complexity. Hence, DDs are adequate for proving space lower bounds and time-space trade-offs. Such bounds are easier to obtain for large domains D implying that the boolean case $D = \{0, 1\}$ and $R = \{0, 1\}$ is of special interest. In the following we consider the boolean case (if nothing else is mentioned) and DDs for the boolean case are called *binary decision diagrams* (BDDs).

Lower bounds on the BDD complexity of explicitly defined boolean functions are difficult to prove. In Section 7, we describe the largest-known lower bound whose size is only $\Omega(n^2/\log^2 n)$. This result has not been improved since more than 35 years which motivates the investigation of restricted BDDs.

A BDD is called *s-oblivious* for a sequence $s = (s_1, \dots, s_\ell)$ of (not necessarily different) variables if the node set can be partitioned into $\ell + 1$ levels such that all nodes on level $i \leq \ell$ are decision nodes for the variable s_i, all nodes on level $\ell + 1$ are output nodes, and each edge leaving level i reaches some level $j > i$. Lower bounds for oblivious BDDs are presented in Section 8. The special case that s is a permutation $(x_{\pi(1)}, \dots, x_{\pi(n)})$ of the input variables is called *ordered binary decision diagram* (OBDD) and, if the permutation π is fixed, π-OBDD. Then $(x_{\pi(1)}, \dots, x_{\pi(n)})$ is the *variable order* of the π-OBDD. If s repeats the variable order π for k times, the resulting BDDs are called k-π-OBDDs and for arbitrary π k-OBDDs. If s is the concatenation of k (perhaps different) variable orders corresponding to $(\pi_1, \dots, \pi_k)$, the resulting BDDs are called $(\pi_1, \dots, \pi_k)$-IBDDs or simply k-IBDDs. Oblivious BDDs and, in particular, π-OBDDs allow the efficient realization of many important operations on boolean functions. Hence, we may use them not only as programs for boolean functions but also for the manipulation of boolean functions, i.e., as a dynamic data structure for boolean functions. Indeed, π-OBDDs are the most-often applied data structure for boolean functions with many applications in various areas among them verification, model checking, CAD tools, optimization, and graph algorithms. In Section 2, the requirements for data structures for boolean functions are discussed and, in Section 3, the corresponding efficient algorithms for π-OBDDs are presented.

The other type of restrictions are length-restrictions. In the *read-k model* the access to the variables is restricted. It is essential to distinguish *syntactic* from *semantic* restrictions. In a syntactic read-k BDD graph theoretical paths contain at most k decision nodes labeled with x_i for each variable x_i. In a semantic read-k BDD only computation paths have to fulfill the read-k restriction. Syntactic restrictions simplify lower bound proofs. In the case of $k = 1$, called *free binary decision diagrams* (FBDDs), the syntactic and the semantic restriction coincide, since inconsistent paths are impossible. Defining graph orders as canonical generalization of variable orders one obtains G-FBDDs (generalizing π-OBDDs) whose algorithmic properties are discussed in Section 4. Only length-restricted BDDs where the length of all computation paths is bounded allow the proof of general time-space trade-offs. In Section 9, such results are presented.

Deterministic models of computation like BDDs and their restrictions can be generalized to *randomized* and *nondeterministic models*. This is done by allowing *randomized nodes* with two outgoing edges where one of these edges is activated independently from all other random decisions with probability 1/2. In the boolean case, this leads to an acceptance and a rejection probability for each input a. We may distinguish the model of zero error, the model of one-sided error, and the model of two-sided error. The permitted error probability is a critical value for certain BDD models, since not all models allow an efficient *probability amplification*. The case of unbounded but non-trivial probability for one-sided errors is the case of *existential nondeterminism* (OR-nondeterminism) which also can be described in the usual way. An input a is accepted if it activates at least one accepting computation path. We also investigate *universal* (AND-) nondeterminism and *parity* (EXOR-) nondeterminism. Lower bounds for nondeterministic BDDs are discussed together with their deterministic counterparts in Section 8. Those restricted OR-π-OBDDs, which have good algorithmic properties are called *partitioned* BDDs (PBDDs) and are discussed in Section 5. Results on randomized BDD variants are presented in Section 10.

This chapter is a brief introduction into the rich world of BDDs and BPs including aspects of complexity theory, algorithms, data structures, and applications. A comprehensive monograph is available (Wegener (2000)).

2 Data Structures for Boolean Functions

In many applications, like e.g., symbolic verification, test pattern generation, symbolic simulation, analysis of circuits and automata, and logical synthesis, representations of boolean functions are needed which are small for many and, in particular, for important functions and simultaneously allow the efficient execution of important operations. In order to look for suitable data structures one first has to clarify which operations have to be supported. Bryant (1986) has presented a list of problems for which efficient algorithms should exist.

- Evaluation Problem. Given a representation G for some boolean function f and an input a. Compute $f(a)$.
- Satisfiability Test. Given a representation G for some boolean function f. Decide whether $f(a) = 1$ for some input a.
- Satisfiability Count. Given a representation G for some boolean function f. Compute $|f^{-1}(1)|$.
- Satisfiability All. Given a representation G for some boolean function f. Compute a list of all $a \in f^{-1}(1)$.
- Minimization (or Reduction). Given a representation G for some boolean function f. Compute a minimal-size representation G' for f within the class of representations described by the chosen data structure. If G' is unique (up to isomorphism), the computation of G' is called *reduction* of G.

- Synthesis. Given representations G_1 and G_2 for some boolean functions g and h and some binary boolean operation $\otimes$. Compute a representation G for $f := g \otimes h$.
- Equality Test. Given representations G_1 and G_2 for some boolean functions g and h. Decide whether g and h are equal.
- Replacement by Constants. Given a representation G for some boolean function g, some variable x_i, and some constant $c \in \{0, 1\}$. Compute a representation G' for $f := g_{|x_i=c}$ defined by $f(a) = g(a_1, \ldots, a_{i-1}, c, a_{i+1}, \ldots, a_n)$.
- Replacement by Functions. Given representations G_1 and G_2 for some boolean functions g and h and some variable x_i. Compute a representation G' for $f := g_{|x_i=h}$ defined by $f = (\overline{h} \wedge g_{|x_i=0}) \vee (h \wedge g_{|x_i=1})$.
- Existential Quantification. Given a representation G for some function g and some variable x_i. Compute a representation G' for $f := g_{|x_i=0} \vee g_{|x_i=1}$.
- Universal Quantification. Given a representation G for some function g and some variable x_i. Compute a representation G' for $f := g_{|x_i=0} \wedge g_{|x_i=1}$.

In order to motivate some of the listed operations, we discuss a typical computer-aided design application. The physical construction of a new VLSI circuit for a boolean function f is quite expensive. Hence, it is necessary to verify the correctness of a circuit design. *Verification of combinational circuits* is the problem to prove the equality of a specification, e.g., an already verified circuit, and a new realization. Let us denote the function realized by the new design f'. Verification can be done by transforming the specification and the realization into the chosen representation type and by an equality test for the results. If $f \neq f'$, we like to analyze the set of inputs a where $f(a) \neq f'(a)$. For the equality test, we may design a representation of $g = f \oplus f'$. The equality test for f and f' is equivalent to the satisfiablity test for g. If $g^{-1}(1)$ is not empty, the size of $g^{-1}(1)$ measures the number of inputs for which f' computes the wrong value. If the number of 1-inputs for g is small it may be useful to list $g^{-1}(1)$ in order to correct the new design.

Several representations for boolean functions are known. Many data structures like circuits, formulas, and branching programs allow succinct representations of many boolean functions but not all operations can be performed in polynomial time. E.g., the satisfiability test is NP-complete for these models and the circuit minimization problem seems to be even harder. For other data structures like disjunctive or conjunctive normal forms or decision trees, simple functions require representations of exponential size.

Ordered binary decision diagrams (OBDDs) introduced by Bryant (1986) are a compromise. If the order of the variables is fixed, efficient algorithms for the operations on OBDDs have to work only under the assumption that all functions are represented by OBDDs respecting the same order π of the variables. Efficient polynomial algorithms on π-OBDDs exist for all operations mentioned above (see Section 3). Although π-OBDDs are a nice data structure with many advantages there are several functions which only have OBDDs of exponential size but which appear to be simple enough to be represented in small poynomial size. This observation has led to several extensions of the OBDD model. In Section 4 and Section 5 we discuss graph driven BDDs and partitioned BDDs which are less restricted than OBDDs but have nevertheless good algorithmic properties.

3 OBDDs and π-OBDDs

The first variant of binary decision diagrams that was suggested as a data structure for boolean functions are π-OBDDs (Bryant (1986)), and they are still the most popular one. In this section we discuss the properties of π-OBDDs that make them suitable for this purpose. We furthermore present some theoretical results on π-OBDDs. For details on the efficient implementation of algorithms on π-OBDDs we refer to Somenzi (2001).

Examples of functions with small-size π-OBDDs are all symmetric functions, all outputs of the addition of two numbers or the bitwise equality test of two n-bit strings. In Figure 3, π-OBDDs for the functions $x_1 \oplus x_2 \oplus x_3$ and for the bitwise equality test of $(x_0, \ldots, x_{n-1})$ and $(y_0, \ldots, y_{n-1})$ are shown. On the other hand, there are several functions which only have OBDDs of exponential size, e.g., the hidden weighted bit function HWB from Section 1 or the middle bit of the multiplication of two n-bit numbers (Bryant (1991), see Section 8). As a tool for obtaining the π-OBDD size of some function we may look at the functions f_v represented at the internal nodes v of the π-OBDD. For the examples in Figure 3 these functions are indicated at the nodes, where $[A]$ takes the value 1 if the expression A is true and otherwise the value 0. The π-OBDD size can be obtained by the following lemma due to Sieling and Wegener (1993a). To simplify the presentation the lemma is only shown for the variable order $x_1, \ldots, x_n$, i.e., for $\pi = \text{id}$.

Lemma 3.1: Let S_i be the set of subfunctions $f_{|x_1=c_1,\ldots,x_{i-1}=c_{i-1}}$ that essentially depend on x_i and where $c_1, \ldots, c_{i-1} \in \{0, 1\}$. A minimal-size id-OBDD for f contains exactly $|S_i|$ nodes labeled by x_i.

The proof of the lemma also shows that for each function in S_i there is an internal node computing this function and that the successors of this node are uniquely determined. This implies the following result, which was first proved by Bryant (1986).

Theorem 3.2: For each function f the minimal-size π-OBDD is unique up to isomorphism.

The question arises how to obtain the minimal-size π-OBDD for some function f from an arbitrary π-OBDD for f. We assume that there are no nodes that are not reachable from the node representing f, which we therefore may call the source of the π-OBDD for f. Bryant (1986) observed that the following two simple reduction rules suffice to minimize π-OBDDs: By the deletion rule nodes where the 0- and 1-successor coincide are deleted and the incoming edges are redirected to the successor. The merging rule allows to merge two nodes v and w with the same label, the same 0-successors and the same 1-successors, i.e., v is deleted and the incoming edges are redirected to w. Similarly,

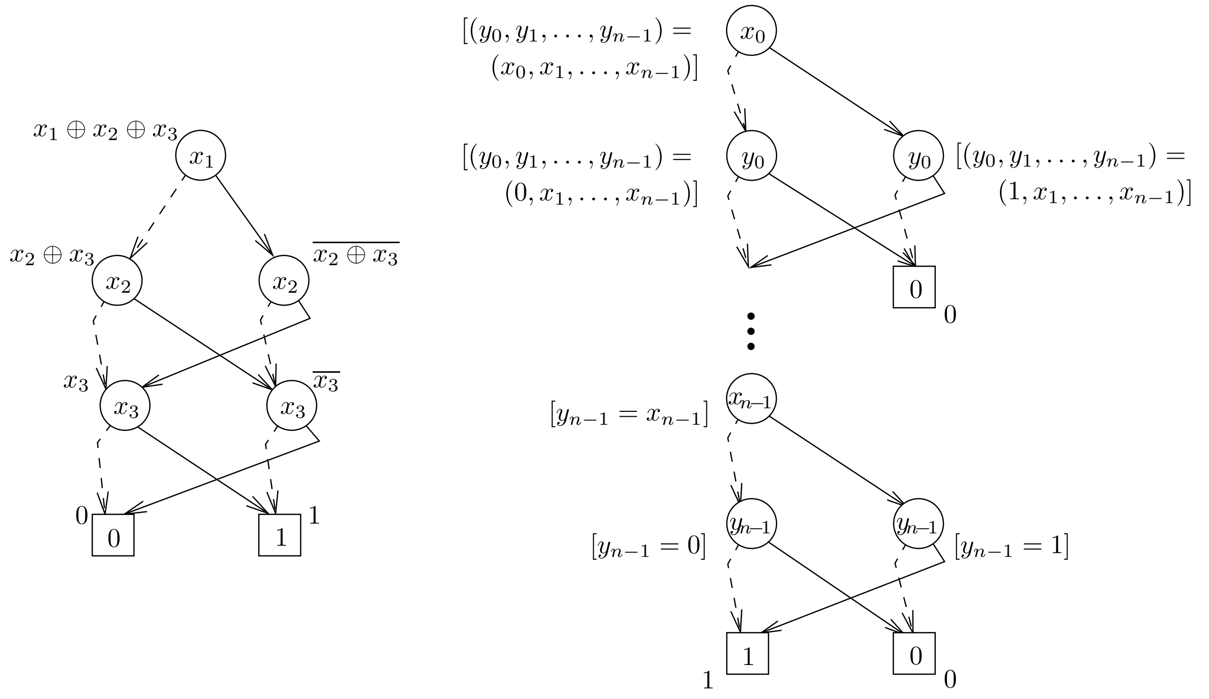
$x_1 \oplus x_2 \oplus x_3$
x_1
$x_2 \oplus x_3$
$\overline{x_2 \oplus x_3}$
x_2
x_2
x_3
$\overline{x_3}$
x_3
x_3
0
1
0
1
$[(y_0, y_1, \ldots, y_{n-1}) = (x_0, x_1, \ldots, x_{n-1})]$
x_0
$[(y_0, y_1, \ldots, y_{n-1}) = (0, x_1, \ldots, x_{n-1})]$
y_0
y_0
$[(y_0, y_1, \ldots, y_{n-1}) = (1, x_1, \ldots, x_{n-1})]$
0
0
$[y_{n-1} = x_{n-1}]$
x_{n-1}
$[y_{n-1} = 0]$
y_{n-1}
y_{n-1}
$[y_{n-1} = 1]$
1
0
1
0

Figure 3: Examples of OBDDs for the functions $x_1 \oplus x_2 \oplus x_3$ and the bitwise equality test of $(x_0, \ldots, x_{n-1})$ and $(y_0, \ldots, y_{n-1})$.

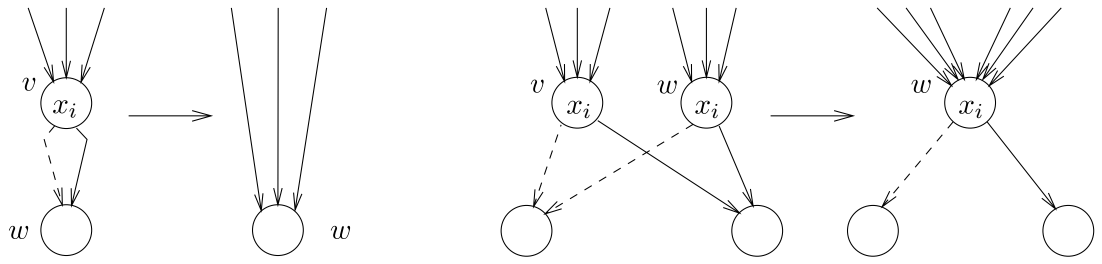

Figure 4: The reduction rules for OBDDs.

sinks with the same label can be merged. The reduction rules are illustrated in Figure 4. Bryant proved that independent of the order of the application of these rules we eventually obtain the minimal-size π-OBDD, which is also called the reduced π-OBDD. In order to obtain an efficient reduction algorithm it is useful to apply the reduction rules bottom-up, since each application of a reduction rule may allow new applications of reduction rules only for preceding nodes. The reduction algorithm of Bryant (1986) has a run time of $O(|G| \log |G|)$ and was later on improved to a linear-time algorithm by Sieling and Wegener (1993b).

The uniqueness of reduced π-OBDDs and the efficient reduction of π-OBDDs has consequences for the Equality Test of two π-OBDDs: It suffices to reduce the given π-OBDDs and test them for isomorphy which is possible in linear time due to the labels of the edges.

A careful investigation of the bitwise equality test shows that its π-OBDDs are only small for a variable order π like $x_0, y_0, x_1, y_1, \ldots$, i.e., for a variable order where the x- and y-variables with the same index are arranged near together. For, e.g., the variable order $x_0, \ldots, x_{n-1}, y_0, \ldots, y_{n-1}$ the π-OBDD size is exponential. This can easily be shown using Lemma 3.1 since the number of different subfunctions obtained by replacing $x_0, \ldots, x_{n-1}$ by constants is 2^n. Intuitively the π-OBDD has to store $x_0, \ldots, x_{n-1}$ in order to compute the function, which requires exponential size. A more general approach to describe this effect is communication complexity theory, see Section 8. Wegener (2000) has even shown that for almost all choices of the variable order the functions addition and bitwise equality test have exponential π-OBDD size. This shows that algorithms for choosing a good variable order are needed in applications.

An algorithm for computing an optimal variable order was presented by Friedman and Supowit (1990). However, this algorithm works on truth tables and has an exponential run time. Several heuristics for improving the variable order were presented in the literature. An example of such a heuristic is the sifting algorithm due to Rudell (1993). This algorithm successively chooses each variable and searches for the position leading to minimal OBDD size. This is done by trying all possible positions for the variable. The sifting algorithm is reported to obtain good results in reasonable time. However, it may also perform poorly and no efficient algorithm for optimizing the variable order is known. The problem of computing an optimal variable order was shown to be NP-hard by Bollig and Wegener (1996). In applications also efficient approximation algorithms for the variable order problem would be helpful. An approximation algorithm with the performance ratio c is an algorithm that for each function f given by an OBDD computes a variable order π such that the π-OBDD size for f is larger than the minimum size by a factor of at most c. However, even the existence of polynomial-time approximation algorithms for the variable order problem implies P = NP (Sieling (2002a)). Hence, we have to be satisfied with heuristics for optimizing the variable order.

A satisfying assignment of a function f given by a π-OBDD can be found by searching for a path from the source to a 1-sink. SATISFIABILITY ALL can be solved by enumerating all such paths where we have to take into account that variables not tested on such a path can take both values 0 and 1. Furthermore, an algorithm for SATISFIABILITY COUNT may use a labeling procedure that for each node v stores the number of assignments to the variables such that the corresponding computation path leads through v. The run time for SATISFIABILITY and SATISFIABILITY COUNT is linear with respect to the input size, and for SATISFIABILITY ALL the run time is linear with respect to the input and output size. We remark that these algorithms depend on the property that in π-OBDDs each variable may be tested at most once on each computation path such that computation paths and graph-theoretical paths coincide. For many variants of BDDs without this property the satisfiability operations are NP-hard.

In applications like hardware verification π-OBDDs have to be computed for functions represented by circuits. We first remark that this problem is NP-hard because SATISFIABILITY is NP-hard for circuits but can be done in linear time for π-OBDDs. The general approach for the transformation of circuits into π-OBDDs works in the following way: We run through the circuit in some topological order. For the functions represented at the inputs, i.e., projections on variables, it is simple to construct π-OBDDs. For a function represented at the output of a gate it is possible to compute a π-OBDD by combining the π-OBDDs representing the functions at the input of the gate with SYNTHESIS. In the following we shortly discuss SYNTHESIS.

SYNTHESIS of π-OBDDs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ is mainly based on computing a product graph of the given π-OBDDs. The node set of the resulting π-OBDD is some subset of $V_1 \times V_2$ and reaching an x_i-node $(v_1, v_2) \in V_1 \times V_2$ for some assignment a to $x_1, \ldots, x_{i-1}$ means that in G_1 the computation path for the partial assignment a ends at the node v_1 and in G_2 at the node v_2. It follows that the number of nodes in the resulting π-OBDD is bounded above by $|V_1||V_2|$ and it is even possible to compute this π-OBDD in time $O(|V_1||V_2|)$. The resulting π-OBDD is not necessarily reduced. We remark that in implementations the reduction is integrated into the synthesis algorithm in order to avoid the construction of large non-reduced OBDDs.

The computation of the product graph is only possible if the given OBDDs G_1 and G_2 for f_1 and f_2 have the same variable order. Otherwise the construction of an OBDD for, e.g., $f_1 \wedge f_2$ is even NP-hard (Fortune, Hopcroft and Schmidt (1979)). Nevertheless, it is a common technique to change the variable order during the computation of an OBDD from a circuit in order to avoid large intermediate OBDDs (Bryant (1995)).

REPLACEMENT BY CONSTANTS can be performed by redirecting the edges leading to each x_i-node v to the c-successor of v. If the source is labeled by x_i, the c-successor is defined as the new source of the OBDD. The quantification operations are combinations of REPLACEMENT BY CONSTANTS and SYNTHESIS. Finally, to perform REPLACEMENTS BY FUNCTIONS we define the ternary operation $\text{ite}(a, b, c) = ab \vee \bar{a}c$ (if a then b else c) and compute a π-OBDD for $g_{|x_i=h} = \text{ite}(h, g_{|x_i=1}, g_{|x_i=0})$ by a generalization of SYNTHESIS to ternary operators.

Altogether, for many of the important operations on boolean functions there are efficient algorithms working on π-OBDDs, and π-OBDDs are of reasonable size for many important functions. However, in applications the operations on π-OBDDs, in particular SYNTHESIS, have to be applied successively such that we cannot bound the size of intermediate π-OBDDs. Hence, π-OBDDs are a heuristic approach for solving practically important problems.

4 Graph Driven BDDs

Several important and also quite simple functions have exponential OBDD size. Therefore, more general representations with good algorithmic behavior are necessary. With a restricted variant of FBDDs like the restriction of OBDDs to π-OBDDs we obtain a new data structure for boolean functions. Gergov and Meinel (1994) and Sieling and Wegener (1995) have generalized independently the concept of variable orders to graph orders.

A *graph order* is a BDD with a single sink, where on each path from the source to the sink all variables appear exactly once. A *graph driven* FBDD G' according to a graph order G, or G-FBDD for short, is an FBDD with the following property: If for an input a a variable x_i appears on the computation path of a in G' before the variable x_j, then x_i also appears on the computation path of a in G before x_j.

In graph driven BDDs (according to a fixed order), for each input the variables are tested in the same order, whereas (different from OBDDs) for different inputs different orders may be used. It is not difficult to see that any FBDD G' is a graph driven FBDD for a suitably chosen graph order: We merge the sinks of G' to a single sink. Then we run through G' top-down. Let $\mathit{Var}(v)$ be the set of all variables tested on some path between the source and v. Replace each edge (v, w), where v is labeled by x_i, with a list of tests for the variables in $\mathit{Var}(w) \setminus (\mathit{Var}(v) \cup \{x_i\})$ in order to obtain G. It follows that graph driven BDDs have the same expressive power as FBDDs, i.e., all boolean functions with polynomial-size FBDDs can be represented by graph driven BDDs of polynomial size.

For the operations EVALUATION, SATISFIABILITY, SATISFIABILITY ALL, and SATISFIABILITY COUNT, the OBDD algorithms (see Section 3) also work for FBDDs but not for all operations efficient algorithms are known. Blum, Chandra, and Wegman (1980) have proved that the EQUALITY TEST for FBDDs is contained in co-RP, i.e., the inequality can be tested probabilistically with one-sided error in polynomial time but no deterministic polynomial-time algorithm is known. Furthermore, SYNTHESIS may lead to an exponential blow-up and the problem REPLACEMENT BY FUNCTIONS is as hard as the synthesis problem.

Since efficient algorithms exist for all operations on π-OBDDs and a fixed variable order π, we hope for efficient algorithms on G-FBDDs and a fixed graph order G. First, we look at the functions that have to be represented at the internal nodes of a G-FBDD. Let v be a node in the graph order G and let w.l.o.g. $\mathit{Var}(v) = \{x_1, \ldots, x_{i-1}\}$, and $\mathcal{A}(v) \subseteq \{0,1\}^{i-1}$ be the set of partial assignments $(a_1, \ldots, a_{i-1})$ to the variables $x_1, \ldots, x_{i-1}$ such that v is reached for all inputs a starting with $(a_1, \ldots, a_{i-1})$. We define $\mathcal{F}_v = \{f_{|x_1=a_1,\ldots,x_{i-1}=a_{i-1}} | (a_1, \ldots, a_{i-1}) \in \mathcal{A}(v)\}$ and denote the graph order that is the subgraph of G with source v by $G(v)$. A G-FBDD representing a function f has to contain a $G(v)$-driven FBDD for each subfunction $f_v \in \mathcal{F}_v$.

Sieling and Wegener (1995) have proved that there is (up to isomorphism) a unique G-FBDD of minimal size for each function f, i.e., G-FBDDs are a canonical representation of boolean functions and the operation REDUCTION is well defined. Like π-OBDDs a G-FBDD is reduced iff neither the deletion rule nor the merging rule is applicable. Sieling and Wegener (1995) have designed a linear-time reduction algorithm for G-FBDDs. The difficulty is to decide how to proceed bottom-up. In the case of OBDDs, the variable order helps to investigate the x_i-node before the x_j-node if x_j precedes x_i in the variable order. A graph order can combine many variable orders. Therefore, the bottom-up application of the reduction rules has to be guided quite carefully in order to guarantee the linear runtime.

The representation by reduced G-FBDDs implies a linear-time EQUALITY TEST since the equality check is a simple isomorphism check.

For the SYNTHESIS of two G-FBDDs, $G=(V,E)$, $G_1=(V_1,E_1)$ and $G_2=(V_2,E_2)$ we run simultaneously through G, G_1, and G_2. The node set of the resulting G-FBDD is some subset of $V\times V_1\times V_2$. Therefore, the size of the resulting G-FBDD can be bounded above by $|V||V_1||V_2|$ and the result can be computed in time $O(|V||V_1||V_2|)$. This algorithm does not create reduced G-FBDDs. Since the application of BDDs in practice is more limited by restrictions of the available storage space than by restrictions of the available time, the reduction is integrated into the synthesis process in implementations.

REPLACEMENT BY CONSTANTS and QUANTIFICATION may cause an exponential blow-up of the size of G-FBDDs. Since $f_{|x_i=c}=(\overline{c}\wedge f_{|x_i=0})\vee(c\wedge f_{|x_i=1})$, the same holds for REPLACEMENT BY FUNCTIONS. As a consequence, G-FBDDs cannot be used efficiently if our application needs one of these operations. But if we know in advance which variables are used in these operations, we may work with a graph order with some additional properties. A graph order G is called *x_i-oblivious* if for each x_i-node its 0-successor coincides with its 1-successor. The REPLACEMENT and QUANTIFICATION problems can be solved efficiently for variables for which the considered graph order is oblivious.

Similarly to the variable ordering problem for OBDDs we are faced with the graph ordering problem for FBDDs, i.e., the problem of finding a suitable graph order. Sieling (2002b) has shown that the existence of polynomial-time approximation schemes for optimizing the graph order implies P=NP. The known heuristics do not lead to satisfactory results. The only graph ordering algorithm tested in experiments is due to Bern, Meinel, and Slobodová (1996). Their approach creates graph orders of the following kind. For a parameter d, the graph order starts with a complete binary tree of depth d. For each leaf of this tree, a variable order of the remaining $n-d$ variables follows.

5 Partitioned Binary Decision Diagrams

Nondeterminism is a powerful compexity theoretical concept and nondeterministic representations of boolean functions can be much smaller than their deterministic counterparts. E.g., the function HWB (see Section 1) has exponential OBDD size but can be represented by OR-OBDDs of size $O(n^3)$. The output bit x_i, $0\le i\le n$, is guessed, afterwards it is verified whether $x_1+\cdots+x_n=i$ and $x_i=1$. A disadvantage is that the simple operation NOT may cause an exponential blow-up of the size. In order to obtain representation types with good algorithmic behavior we have to consider restrictions where, in particular, NEGATION is not difficult. Partitioned binary decision diagrams (PBDDs) introduced by Jain, Bitner, Fussell, and Abraham (1992) and more intensively studied by Narayan, Jain, Fujita, and Sangiovanni-Vincentelli (1996) have the desired properties. PBDDs are a generalized OBDD model allowing a restricted use of nondeterminism and different variable orders. They are restricted enough such that most of the essential operations can be performed efficiently and they allow polynomial-size representations for more functions than OBDDs.

We define (k,w,π)-PBDDs where $k=k(n)$ is the number of parts, $w=(w_1,\ldots,w_k)$ is the vector of so-called window functions, and $\pi=(\pi_1,\ldots,\pi_k)$ is the vector of variable orders.

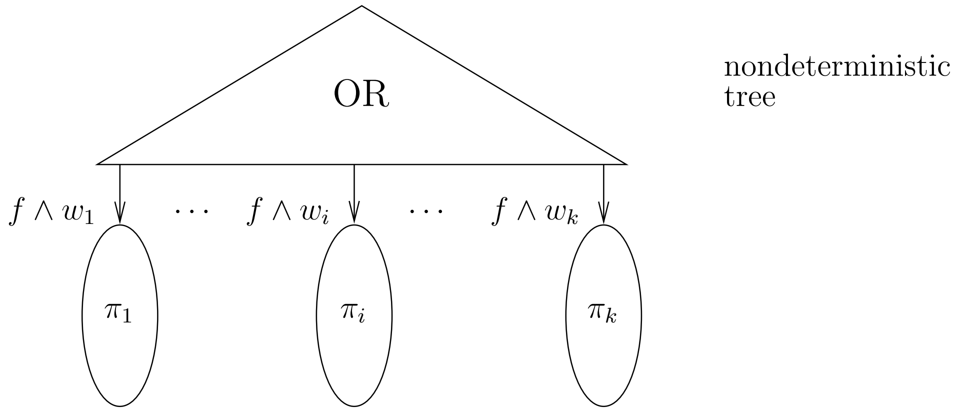

Figure 5: A partitioned (nondeterministic) BDD with k parts representing $f \wedge w_1, \dots, f \wedge w_k$.

A necessary condition for the window functions is that their disjunction is the constant 1. Figure 5 describes a (k, w, π)-PBDD. One of the k parts is chosen nondeterministically. The ith part represents $f \wedge w_i$ by a π_i-OBDD. The function represented by the (k, w, π)-PBDD equals

$$(f \wedge w_1) + \cdots + (f \wedge w_k) = f \wedge (w_1 + \cdots + w_k) = f$$

explaining the necessary condition for the window functions. The window functions are called disjoint if $w_i \wedge w_j = 0$ for $i \neq j$. Then $w_1^{-1}(1), \dots, w_k^{-1}(1)$ are a partition of the input space justifying the notion of partitioned BDDs. Then at most one part computes 1 which is known in complexity theory as unique nondeterminism.

The size of a (k, w, π)-PBDD consisting of the π_i-OBDDs $G_i, 1 \leq i \leq k$, is the sum of the sizes of its parts G_i. If only the number k of parts is fixed, we use the notion of k-PBDDs. Since the algorithms on PBDDs need π_i-OBDDs for the corresponding window functions w_i, π and w should be chosen such that the π_i-OBDD size of w_i is small.

As in the case of OBDDs we look for efficient algorithms only if the number of parts k, the vector of window functions w, and the vector of variable orders are fixed. The representation by (k, w, π)-PBDDs is canonical and the REDUCTION of a (k, w, π)-PBDD G is possible in time $O(|G|)$ since it can be done individually for all parts. Moreover, there exists an efficient equality check for two functions f and g since we can use the EQUALITY TEST for π-OBDDs (see Section 3) to check pairwise the equality of the corresponding parts. SYNTHESIS of (k, w, π)-PBDDs G_f and G_g representing the functions f and g respectively, can be computed in time $O(|G_f||G_g|)$ for the operations AND, OR, and EXOR. It is sufficient to apply the π_i-OBDD synthesis algorithm to the ith parts of G_f and G_g, $1 \leq i \leq k$. Generally, $\overline{f \wedge w_i} \neq \overline{f} \wedge w_i$ but

$$(\overline{f \wedge w_i}) \wedge w_i = (\overline{f} \vee \overline{w_i}) \wedge w_i = \overline{f} \wedge w_i.$$

Hence, NEGATION can be performed as negation of each part followed by an AND-SYNTHESIS with the corresponding window function. A function f is satisfiable iff one of the functions $f \wedge w_i$ is satisfiable. SATISFIABILITY COUNT is difficult if the window functions are non-disjoint since many parts may compute 1 for an input. For disjoint window functions the number of satisfying inputs is the sum of the number of satisfying inputs for the parts.

The quantification operations are based on Replacement by Constants but this operation causes problems for (k, w, π)-PBDDs. Let g_n test whether a boolean matrix consists of rows with exactly one 1-entry and let h_n be the analogous function for the columns. The function $f_n = sg_n \vee \overline{s}h_n$ can be represented by 2-PBDDs and the window functions s and $\overline{s}$. For the first part we use a rowwise variable order and for the second part a columnwise variable order. For the replacement of s by 1, we obtain the function g_n. Then we have to represent $\overline{s}g_n$ in the second part by a columnwise variable order which needs exponential size.

The heuristics for the generation of window functions often construct window functions which are the minterms with respect to a small set V of variables. Then the replacement of a variable $x_i \in X_n \setminus V$ by a constant is easy. This holds more generally if the window functions do not essentially depend on x_i. Then $f_{|x_i=c} \wedge w_j = (f \wedge w_j)_{|x_i=c}$ and the replacement can be done for each part in the usual way (see Section 3). Afterwards, Quantification is a binary synthesis operation.

Why are the window functions necessary? The model of k-PBDDs in its general form has several practical drawbacks. Negation may lead to an exponential blow-up. Even if the k variable orders are fixed, it is difficult to check whether two k-PBDDs G' and G'' represent the same function. An input a may lead to 1 in the ith part of G' but only in the jth part of G'', where $i \neq j$. Furthermore, it is easy to prove that the Equality Test is co-NP complete already for 2-PBDDs.

Since the number of parts and the window functions have to be fixed if we work with (k, w, π)-PBDDs one may ask whether the choice of the number of parts and the corresponding window functions is crucial for small size PBDDs. Bollig and Wegener (1999) have proved that PBDDs of polynomial size form a tight hierarchy with respect to the number k of their parts. They have shown that k-PBDDs may be exponentially larger that $(k+1)$-PBDDs for the same function if $k = o((\log n/\log\log n)^{1/2})$. Sauerhoff (2000) has improved their result up to $k = O((n/\log^{1+\varepsilon} n)^{1/4})$, where $\varepsilon > 0$ is an arbitrarily small constant.

Sometimes we find appropriate window functions because of our knowledge of the structural properties of the functions, e.g., for HWB we use the window functions w_i, $0 \leq i \leq n$, which compute 1 iff $x_1 + \cdots + x_n = i$. Good heuristic algorithms for the automatic creation of appropriate window functions have been developed using methods known as functional partitioning (Jain, Bitner, Fussell, and Abraham (1992) and Lai, Pedram, and Vrudhula (1993)). Heuristic algorithms for which experimental results have proved the practical usefulness have also been presented by Jain, Mohanram, Moundanos, Wegener, and Lu (2000).

6 BDD Size versus Other Complexity Measures

Since BDDs are a nonuniform model of computation, we discuss relations between the size of BDDs and complexity measures for nonuniform models of computation, namely nonuniform space complexity, circuit size and formula size. We recall that for considering nonuniform models of computation we decompose the function $f\colon \{0,1\}^* \to \{0,1\}$ into a sequence $f = (f_n)$, where $f_n\colon \{0,1\}^n \to \{0,1\}$, and for each function of the sequence we may design a different algorithm or circuit. A *nonuniform Turing machine* is a Turing machine with an advice tape, whose contents are initialized only depending on the input length such that it may perform different algorithms for different input lengths. For more details we refer to Johnson (1990).

The first result concerns the relation between BDD size and nonuniform sequential space complexity. Let $\mathrm{BDD}(f_n)$ denote the minimum size of a BDD computing f_n. Let $\mathrm{S}(f_n)$ be the minimum space of a nonuniform Turing machine computing f_n. The following relationship between $\mathrm{BDD}(f_n)$ and $\mathrm{S}(f_n)$ was obtained by Cobham (1966) and Pudlák and Žák (1983).

Theorem 6.1: Let $f = (f_n)$ be a sequence of boolean functions. If $\mathrm{S}(f_n) \geq \log n$, then $\mathrm{BDD}(f_n) = 2^{O(\mathrm{S}(f_n))}$. If $\mathrm{BDD}(f_n) \geq n$, then $\mathrm{S}(f_n) = O(\log \mathrm{BDD}(f_n))$.

Proof. The theorem follows from simulations between BDDs and nonuniform space-bounded Turing machines. For the first statement we observe that the number of configurations of a nonuniform $\mathrm{S}(f_n)$ space-bounded Turing machine on input length n is at most $2^{c\,\mathrm{S}(f_n)}$ for some constant c. For each such configuration the BDD for input length n contains a node. Accepting configurations are 1-sinks and rejecting configurations are 0-sinks. In each non-halting configuration C the Turing machine reads one input bit x_i and according to the value $d \in \{0,1\}$ of x_i the successor configuration C_d is reached. Then C is simulated by a BDD node labeled by x_i and with the successors corresponding to C_0 and C_1.

For the second statement we construct a nonuniform Turing machine from a given sequence of BDDs of size $\mathrm{BDD}(f_n)$. For input length n, the contents of the advice tape is a coding of the BDD for input length n. The simulation of the computation of the BDD for the given input merely requires to store a pointer to the node reached. Such a pointer can be stored in space $O(\log \mathrm{BDD}(f_n))$. □

Now we consider the relationship between BDD size and circuit and formula size. Let $\mathrm{C}(f_n)$ and $\mathrm{L}(f_n)$ denote the minimum size of circuits and formulas over the basis {AND, OR, NOT} for f_n, resp., where it is common to count only the number of AND- and OR-gates and to neglect the NOT-gates. Let $\mathrm{L}^*(f_n)$ denote the minimum size of formulas for f_n over the basis consisting of all boolean functions with two inputs.

Theorem 6.2: Let $f = (f_n)$ be a sequence of boolean functions. Then

$$\frac{\mathrm{C}(f_n)}{3} \leq \mathrm{BDD}(f_n) \leq \mathrm{L}(f_n) + 3$$

and for each $\varepsilon > 0$:

$$\mathrm{BDD}(f_n) = O(\mathrm{L}^*(f_n)^{1+\varepsilon}).$$

Again the first inequalities can be proved by simulations. In order to simulate a BDD by a circuit we replace each c-sink of the BDD by the constant input c and we replace each node labeled by x_i and with the successors a and b by a multiplexer circuit. This circuit computes the function $a\bar{x}_i \vee bx_i$. Furthermore, we assume that the edges are now directed from the constant inputs to the former source of the BDD. Then we obtain a circuit computing the function represented by the BDD, where the output is the former source of the BDD. Each internal node is replaced by one OR-gate and two AND-gates. Hence, the size of the circuit is bounded above by three times the size of the BDD.

The simulation of a formula by a BDD can be performed using the SYNTHESIS operation, which however is performed in a way different from that for π-OBDDs. Let two BDDs G_1 and G_2 for functions f_1 and f_2 be given, where we may assume that each of the BDDs only contains one 0-sink and one 1-sink. Then a BDD for $f_1 \wedge f_2$ can be obtained by replacing the 1-sink of G_1 by a copy of G_2 and merging the 0-sinks of G_1 and G_2. Similarly a BDD for $f_1 \vee f_2$ can be obtained by replacing the 0-sink of G_1 by a copy of G_2 and merging the 1-sinks of G_1 and G_2. Obviously the size of the resulting BDD is bounded by $|G_1| + |G_2| - 2$. A BDD for $\bar{f}_1$ can be obtained from G_1 by exchanging the sinks. For $f_1 \oplus f_2$ the situation is different: We replace the 0-sink of G_1 by a copy of G_2 and the 1-sink of G_1 by a BDD for $\bar{f}_2$, which we obtain from G_2 by exchanging the sinks. Alternatively, we may exchange the roles of G_1 and G_2.

Now we may apply the algorithm given in Section 3 for the transformation of circuits into BDDs to the given formula. For formulas over the basis {AND, OR, NOT} the size bound follows by a simple induction. For formulas with arbitrary binary gates we may first replace each gate in a straightforward way by an AND- or EXOR-gate combined with some NOT-gates. In the above simulation of an EXOR-gate combining f_1 and f_2 we need two copies of G_1 or G_2. Sauerhoff, Wegener and Werchner (1999) showed how to perform this simulation in order to obtain a BDD for f_n of size bounded by $\alpha(\mathrm{L}^*(f_n)+1)^\beta$, where $\alpha \leq 1.360$ and $\beta = \log_4(3+\sqrt{5}) < 1.195$. The stronger result shown in Theorem 6.2 was obtained by Giel (2001) using much more involved arguments.

7 Lower Bounds for BDDs

In the last section, we have seen that lower bounds on the BDD size imply lower bounds of the same size on the {AND, OR, NOT}-formula size and lower bounds of almost the same size on the formula size for the full binary basis. A method due to Nechiporuk (1966) still yields the largest lower bounds for BDDs as well as for general binary formulas (see Chapter **??**). The idea is that a BDD for f of limited size can realize only a limited number of subfunctions of f which are obtained by replacing the variables outside a chosen set $S \subseteq X := \{x_1, \ldots, x_n\}$ by constants. Hence, functions with many different subfunctions need BDDs of not too small size. Before we prove Nechiporuk's bound we show that

BDDs of limited size cannot represent too many functions. The bound of the following lemma can be improved but is sufficient for our purposes.

Lemma 7.1: The number of functions $f\colon \{0,1\}^n \to \{0,1\}$ whose BDD size is bounded by s is at most $n^s(s!)^2$.

Proof. We count the syntactically different BDDs of size s. Each BDD for non-constant functions has two sinks and can be described by an ordered list of $s-2$ instructions. For each instruction, we have the choice between n variables. For the ith instruction and each successor we have the choice between $s-i$ nodes, namely the instructions $i+1,\dots,s-2$ and two sinks. Hence, the number of syntactically different BDDs of size s is bounded by $n^{s-2}((s-1)!)^2$. Each BDD represents at most s different functions. □

In the following, an *S-subfunction* of f is a subfunction obtained by replacing the variables outside S by constants.

Theorem 7.2: Let $f\colon \{0,1\}^n \to \{0,1\}$ essentially depend on all n variables and let $S_1,\dots,S_k$ be disjoint subsets of the variable set X. Then

$$\mathrm{BDD}(f) = \Omega\left(\sum_{1\le i\le k} (\log s_i)/\log\log s_i\right)$$

where s_i is the number of S_i-subfunctions.

Proof. We fix an optimal BDD G representing f. Let t_i be the number of nodes labeled by S_i-variables. This implies $\mathrm{BDD}(f) \ge t_1 + \cdots + t_k + 2$ and it is sufficient to prove that $t_i = \Omega((\log s_i)/\log\log s_i)$. Obviously, $t_i \ge |S_i|$, since f essentially depends on all variables. Moreover, for each S_i-subfunction, we obtain a BDD from G whose size is bounded by t_i. It is sufficient to replace the variables outside S_i by the appropriate constants. By Lemma 7.1 and $|S_i| \le t_i$,

$$s_i \le |S_i|^{t_i}(t_i!)^2 \le t_i^{t_i}(t_i!)^2 \le (t_i)^{3t_i}$$

implying the claimed lower bound on t_i. □

What is the largest possible size of Nechiporuk's lower bound? There are $2^{n-|S_i|}$ different assignments to the variables outside S_i and $2^{2^{|S_i|}}$ functions on S_i. Hence,

$$(\log s_i)/\log\log s_i \le \min\{(n-|S_i|)/\log(n-|S_i|), 2^{|S_i|}/|S_i|\}.$$

This implies that each set S_i contributes not more than $n/\log n$ to the lower bound. However, a large contribution is only possible for a large set S_i. By elementary calculus, it follows that Nechiporuk's lower bound is bounded above by $O(n^2/\log^2 n)$.

There is a simple function called ISA modeling indirect storage access where Nechiporuk's bound is of size $\Omega(n^2/\log^2 n)$. ISA is defined on $n+k$ variables $x_0,\dots,x_{n-1},y_0,\dots,y_{k-1}$ where $n = 2^m$ and $k = m - \lfloor \log m \rfloor$. The vector $y = (y_{k-1},\dots,y_0)$ is interpreted as a binary number $|y|$. If $|y| \geq \lfloor n/m \rfloor$, $\mathrm{ISA}(x,y) := 0$. Otherwise, the x-block $x(y) := (x_{|y| \cdot m}, \ldots, x_{|y| \cdot m+m-1})$ is considered as an address. Then $\mathrm{ISA}(x,y) := x_{|x(y)|}$ is the contents of the storage cell addressed by $x(y)$. It is easy to see that even the FBDD size of ISA is bounded above by $O(n^2/\log n)$. We read y, then $x(y)$ and, if not already done, $x_{|x(y)|}$. These are at most $2m - \lfloor \log m \rfloor + 1$ variables leading to an upper bound of $2^{2m - \lfloor \log m \rfloor + 1} = O(n^2/\log n)$. We apply Nechiporuk's bound to ISA and define $S_i := \{x_{im}, \ldots, x_{im+m-1}\}, 0 \leq i \leq \lfloor n/m \rfloor - 1$. The number of S_i-subfunctions of ISA is bounded below by 2^{n-m}, since for $|y| = i$ all 2^{n-m} assignments to the x-variables outside S_i lead to different S_i-subfunctions. Nechiporuk's bound consists of $\lfloor n/m \rfloor$ terms of size $(n-m)/\log(n-m)$ implying the lower bound $\Omega(n^2/\log^2 n)$.

8 Lower Bounds for Oblivious BDDs

Each BDD with s decision nodes is an oblivious BDD with s levels, since we can use the topological order of the decision nodes and can define the ith level as the set containing only the ith decision node. Hence, this BDD is even an (sn)-π-OBDD for each variable order π. However, oblivious BDDs with a small number of levels are strongly restricted BDDs. Nevertheless, there are many important functions which have short representations by small-size and small-length oblivious BDDs:

- All bits of n-bit-addition can be represented by an OBDD of size $9n - 5$.
- The multiplexer or direct storage access function has OBDD size $2n + 1$.
- Symmetric functions have quadratic OBDD size.
- The *equality test* ($\mathrm{EQ}(x,y) = 1$ iff $x = y$) and the *inner product* ($\mathrm{IP}(x,y) = x_1 y_1 \oplus \cdots \oplus x_n y_n$) have linear OBDD size.
- The test PERM whether an $n \times n$ boolean matrix is a permutation matrix can be represented by a 2-IBDD of linear size (choose a rowwise and a columnwise variable order).

Furthermore, nondeterminism is a powerful concept for oblivious BDDs.

- HWB (see Section 1) and ISA (see Section 7) can be represented in nondeterministic OBDDs of polynomial size: Guess the output bit and verify that the guess was correct and the output bit equals 1. Since we may also check whether the output bit equals 0 and since we obtain at most one accepting path, these bounds hold for all three types of nondeterminism.
- The test whether a matrix is a permutation matrix is easy for AND-OBDDs, since it is sufficient to check whether each row and each column contains exactly one 1-entry.

These upper bounds motivate the investigation of lower-bound techniques. The most common approach is based on the theory of communication complexity (see Chapter **??**, Hromkovič (1987), Kushilevitz and Nisan (1997)). The communication game is defined for a "distributed" function $f\colon \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$, where Alice holds the first n variables and Bob the last m variables. Before knowing the specific values of her and his variables they agree on a protocol how to interpret their communication. The communication can be stopped if one of the players knows $f(a,b)$. The length of a protocol is the largest number of bits (with respect to all inputs) exchanged between the two players. The *communication complexity* $C(f)$ of f with respect to a given partition of the variables is the minimal length of all communication protocols for f. In the following we prove a simple upper bound on the communication complexity of f depending on parameters of oblivious BDDs representing f. This implies that we can apply lower bounds on the communication complexity of boolean functions to obtain lower bounds on the size of oblivious BDDs.

We identify the variables given to Alice with A and the variables given to Bob with B. A level of an oblivious BDD G is "owned" by the player holding the variable which is the label of the decision nodes of this level. A *layer* of G is a maximal block of consecutive levels owned by the same player. The *layer depth* $\mathrm{ld}(G)$ equals the number of layers of G.

Lemma 8.1: Let G be an oblivious BDD representing $f\colon \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$. Then

$$C(f) \le (\mathrm{ld}(G) - 1) \cdot \lceil \log |G| \rceil$$

and the communication protocol works in $\mathrm{ld}(G) - 1$ rounds.

Proof. The player holding the variables of the first layer, w.l.o.g. Alice, starts the communication. She knows the variables tested in the first layer. Hence, she can compute the first node on the computation path which lies outside her layers. Her first message equals the number of this node. Then the communication is continued in the same way by Bob. After at most $\mathrm{ld}(G) - 1$ messages of length $\lceil \log |G| \rceil$ each, the player owning the last layer can compute $f(a,b)$. □

The result of this lemma can be restated as a lower bound on the size of oblivious BDDs G, namely

$$|G| > 2^{C(f)/(\mathrm{ld}(G)-1)-1}.$$

We have stated and proved the lemma for deterministic BDDs and deterministic communication protocols. We can state and prove it in the same way for nondeterministic protocols and all considered modes of nondeterminism. Nondeterministic protocols are defined in the obvious way. E.g., an EXOR-nondeterministic protocol accepts (a,b) if the number of accepted bit sequences describing possible communications is odd and the corresponding complexity measure is called $C_{\mathrm{EXOR}}(f)$. The theory of communication complexity provides methods for proving large lower bounds on the deterministic and even nondeterministic communication complexity of important functions. However, we only obtain large lower bounds on $|G|$ if $\mathrm{ld}(G)$ is not too large.

First, we discuss the basic lower-bound techniques. The *communication matrix* M consists of 2^n rows corresponding to the possible inputs of Alice and of 2^m columns for the inputs of Bob. The entry at position (a,b) equals $f(a,b)$. A *rectangle* R is defined as the submatrix consisting of all entries of M which belong to a certain subset of the rows and a certain subset of the columns. It is called *monochromatic* if all entries have the same color (or value). Let t, t_0, and t_1 denote the minimal number of monochromatic, 0-colored, and 1-colored rectangles, resp., which are a disjoint cover of M, the zeros of M, and the ones of M, resp. Let t^*, t_0^*, and t_1^* be the corresponding numbers of not necessarily disjoint covers. Then

$$C(f) \geq \log t,\ C_{\mathrm{OR}}(f) \geq \log t_1^*, \text{ and } C_{\mathrm{AND}}(f) \geq \log t_0^*.$$

The fooling set method is a method to get lower bounds on t^*, t_0^*, and t_1^*. A *c-fooling set* is a set S_c of entries (a,b) such that $f(a,b)=c$ for all $(a,b)\in S_c$ and $f(a,b')\neq c$ or $f(a',b)\neq c$ for all distinct (a,b), $(a',b')\in S_c$. Then $t_c^* \geq |S_c|$, since no c-rectangle can cover two elements of S_c. Moreover, let $R_1,\dots,R_k$ be disjoint rectangles covering all 1-entries of M. Let M_i be the $2^n\times 2^m$-matrix where all entries of R_i equal 1 and all other entries equal 0. Then M is the sum of all M_i and, by the sub-additivity of the rank operator and the fact that all M_i have rank 1, we get the lower bound $\mathrm{rank}(M)$ on $C(f)$. This does not hold for non-disjoint covers with the exception of the case of EXOR-nondeterminism and the field $\mathbb{Z}_2$. This leads to the lower bound $\mathrm{rank}_{\mathbb{Z}_2}(M)$ on $C_{\mathrm{EXOR}}(f)$.

The next step is to investigate the layer depth. For a k-π-OBDD we can give the first $r, 1\leq r\leq n$, variables according to the variable order to Alice and the remaining variables to Bob. Then the layer depth equals $2k$ and it is sufficient to prove lower bounds on the length of protocols with $2k-1$ rounds. For lower bounds for k-OBDDs we have to consider all variable orders and the corresponding partitions of the variables between Alice and Bob.

For k-IBDDs and general oblivious BDDs the situation is more difficult. The idea is to find not too small disjoint subsets A and B of the variable set such that the layer depth with respect to A and B is small. Then we have the freedom to choose "good" assignments to the variables outside $A\cup B$ such that the communication complexity of the resulting subfunction is large. Alon and Maass (1988) have used a Ramsey-type argument to prove that, for oblivious BDDs with kn levels, A and B can be chosen such that they contain at least $n/2^{4k+1}$ variables each and such that the layer depth is bounded by $4k+1$. This approach works only if $k=o(\log n)$. We have no complete control which variables survive. Hence, we obtain lower bounds only for functions where subfunctions on many subsets of variables and arbitrary partitions of the variables between Alice and Bob are difficult.

An example is the middle bit of integer multiplication where we are interested in the bit at position $n-1$ of the product of two n-bit numbers x and y. We decide that no y-variable survives and we assign constants to the y-bits such that only two y-bits equal 1. This implies that $x\cdot y = x2^i + x2^j$ for some $i,j\in\{0,\dots,n-1\}$, $i\neq j$. By the pigeonhole principle, there exist i and j such that $x2^i$ and $x2^j$ have many bit positions $k\leq n-1$ such that Alice owns bit k of one of the terms and Bob owns bit k of the other term. All other positions $\ell \le n-1$ are fixed such that one term contains 1 and the other 0. Then we are basically in the situation of the addition of two numbers a and b such that Alice owns a and Bob owns b and we are interested in the carry bit. The communication matrix is of size $2^r \times 2^r$ for some r and contains ones at positions (i,j) such that $i+j \ge 2^r, 0 \le i,j \le 2^r-1$. The matrix has rank 2^r-1 over $\mathbb{R}$ as well as over $\mathbb{Z}_2$. It contains a 0-fooling set of size 2^r (all (i,j) where $i+j=2^r$) and a 1-fooling set of size 2^r-1 (all (i,j) where $i+j=2^r+1$). Choosing the right parameters (Bryant (1991), Gergov (1994)) this leads to the bound $2^{\Omega(n/k^3 2^{4k})}$ for oblivious BDDs of length $2kn$ and all types of nondeterminism.

In order to separate the different modes of nondeterminism we again apply results from communication complexity:

- The equality test $\mathrm{EQ}(x,y)$ (Alice owns x and Bob owns y) has linear communication complexity for OR- and EXOR-nondeterminism but only logarithmic communication complexity for AND-nondeterminism, for the negation $\overline{\mathrm{EQ}}$ the roles of OR and AND are interchanged.

- The inner product function $\mathrm{IP}(x,y)$ has linear communication complexity for OR- and AND-nondeterminism but only logarithmic communication complexity for EXOR-nondeterminism.

However, these results only hold for the bad partition of the input where Alice gets x and Bob gets y. All these functions have linear OBDD size for the interleaved variable order $x_1, y_1, \ldots, x_n, y_n$. A general technique working with so-called bit masks a and b generalizes the lower bounds on communication complexity to all balanced partitions of the input. For $f_n\colon \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ we define the generalized function $f_n^* : \{0,1\}^{4n} \to \{0,1\}$. The input is (a,b,x,y). We obtain x' as vector of all x_i where $a_i = 1$, similarly y' based on y and b. If x' and y' have different length, $f_n^*(a,b,x,y)=0$. If they have the same length m, then $f_n^*(a,b,x,y) = f_m(x',y')$. Then we obtain for each type of nondeterminism a function ($\overline{\mathrm{EQ}}^*$ for OR, EQ^* for AND, IP^* for EXOR) such that the function has polynomial-size OBDDs for all variable orders and the chosen type of nondeterminism but exponential size for oblivious BDDs of linear length and the other two types of nondeterminism.

Finally, we compare k-OBDDs and k-IBDDs. The permutation matrix test PERM can be represented by linear-size 2-IBDDs but needs size $2^{\Omega(n/k)}$ for k-OBDDs. The lower bound method for k-OBDDs works, since we can argue about all variables, and it breaks down for 2-IBDDs, since, as shown above, we have to fix too many variables. Moreover, the number k is of importance. The *pointer jumping function* is defined as follows. The function describes a graph on $2n+1$ vertices $u, v_0, \ldots, v_{n-1}, w_0, \ldots, w_{n-1}, n = 2^k$, and for each vertex v^* there are k boolean variables describing a number $j \in \{0, \ldots, n-1\}$. If $v^* \in \{u, w_0, \ldots, w_{n-1}\}$, the pointer from v^* leads to v_j, for $v^* \in \{v_0, \ldots, v_{n-1}\}$ the pointer from v^* leads to w_j. Moreover, there are n boolean variables describing the colors of $v_0, \ldots, v_{n-1}$. The function $\mathrm{PJ}_{n,k}$ outputs the color of the vertex reached by the unique path of length $2k+1$ starting at u. Obviously, the k-OBDD size of $\mathrm{PJ}_{n,k}$ is bounded by $O(kn^2)$. Based on lower bounds on the length of communication protocols with a fixed number of rounds due to Nisan and Widgerson (1993), Bollig, Sauerhoff, Sieling, and Wegener (1998) have proved that $\mathrm{PJ}_{n,k}$ has exponential $(k-1)$-IBDD size if k is a constant.

Altogether, oblivious BDDs allow a much more compact representation than OBDDs and powerful lower-bound techniques based on communication complexity are available.

9 Lower Bounds for Length-Restricted DDs

In this section, we consider general DDs, not just BDDs. Presently available methods allow to prove exponential size lower bounds for DDs whose length is bounded above by a function of order $O(n \log n)$. These results have the nice feature that they can be interpreted as length-size trade-offs for unrestricted DDs or also as time-space trade-offs for general sequential models of computation such as register machines. The proof methods for length-restricted DDs, of which we will give only a brief outline here due to the highly technical nature of the subject, have been developed in a succession of papers by Beame, Jayram, and Saks (2001), Ajtai (1999a, 1999b), and Beame, Saks, Sun, and Vee (2000, 2002). These are in turn based on a long history of methods for much more restricted BDDs, e. g., oblivious BDDs (see Section 8), read-once BDDs (Wegener (1988), Žák (1984)) and syntactic read-k BDDs for $k > 1$ (Borodin, Razborov, and Smolensky (1993), Okol'nishnikova (1993)).

A good starting point for the proof methods is again communication complexity theory. In Section 8, we have seen that the minimum number t_1^* of 1-colored rectangles required to cover the ones in the communication matrix of f provides a lower bound on the nondeterministic communication complexity of f and in turn on the size of nondeterministic oblivious BDDs. Here it is more appropriate to regard rectangles as sets of the form $R = R_A \times R_B$, where R_A and R_B are sets of partial assignments to the variables in the sets A and B given to Alice and Bob, resp. Then t_1^* can be characterised as the minimum number of rectangles $R \subseteq f^{-1}(1)$ required to cover $f^{-1}(1)$. Lower bounds on the number t_1^* can be proved by the *rectangle size method* from communication complexity:

1. Choose a probability distribution μ on the inputs.
2. Prove a lower bound on $\mu(f^{-1}(1))$ and an upper bound on the measure (density) $\mu(R)$ of each rectangle $R \subseteq f^{-1}(1)$, say $\mu(R) \leq \beta$ for each such R.

Then, obviously, $t_1^* \geq \mu(f^{-1}(1))/\beta$. The fooling set method is the special case where μ is the uniform distribution and $\beta = 2^{-n}$ for $f\colon \{0,1\}^n \to \{0,1\}$.

Essentially, an extension of the rectangle size method is also behind the proof method for length-restricted DDs. We develop a simple version of that method (based on the paper of Beame, Jayram, and Saks (2001)) and comment on further developments later on. The main work is to show how a BDD can be translated into a rectangle cover of $f^{-1}(1)$. We first generate a cover of $f^{-1}(1)$ by functions that can be represented by DDs with a simple structure, so-called *decision forests*. Then we further partition the sets of inputs accepted by each of these decision forests into rectangles. In the following, we consider functions $f\colon D^n \to \{0,1\}$, D any finite set, that are defined on the variable set $X = \{x_1, \dots, x_n\}$.

Definition 9.1: A *decision tree* is a DD where each node has indegree at most 1. The function of a decision tree T is also denoted by T. An (r,k)*-decision forest* is a set of decision trees $F = \{T_1, \dots, T_r\}$ where for each input a and for each T_i, the number of variable accesses on the path activated by a in T_i is at most $\lceil kn/r \rceil$. The function computed by F is $F = \bigwedge_{1 \le i \le r} T_i$.

Lemma 9.2: Let r, k be positive integers with $r \le kn$. Let G be a DD of length at most kn for any function $f\colon D^n \to \{0,1\}$. Then there are at most $|G|^{r-1}$ (r,k)-decision forests whose sets of accepted inputs form a partition of $f^{-1}(1)$.

Proof. Modify the given DD G of length $\ell \le kn$ as follows: First, use $\ell + 1$ copies $G_0, \dots, G_\ell$ of G and redirect each edge originating in copy G_i to the copy of its successor node in G_{i+1}. Then replace the sinks in G_i with nodes testing an arbitrary variable and having the respective sinks in G_{i+1} as successors. The new graph G' still computes f, edges lead only from the nodes of G_i to the nodes of G_{i+1}, and each path in G' has length exactly ℓ. For different nodes v and w in G', let $f_{v,w}(a) = 1$ if there is a path in G' activated by a leading from v to w. For $i = 1, \dots, r-1$, let C_i be the set of nodes of $G_{i\lceil \ell/r \rceil}$. Let v_0 be the source of G' and v_r the 1-sink. Then f is the disjunction of the functions $F_{v_1,\dots,v_{r-1}} = \bigwedge_{0 \le i \le r-1} f_{v_i,v_{i+1}}$ over all $(v_1, \dots, v_{r-1}) \in C_1 \times \cdots \times C_{r-1}$, the sets of inputs accepted by these functions are pairwise disjoint, and there are at most $|C_1| \cdots |C_{r-1}| \le |G|^{r-1}$ terms in this disjunction. Furthermore, since each of the functions $f_{v_i,v_{i+1}}$ can obviously be computed by a decision tree where at most $\lceil kn/r \rceil$ variables are read during each computation, each $F_{v_1,\dots,v_{r-1}}$ is an (r,k)-decision forest. This proves the claim. □

The idea behind the decomposition into decision trees is that we can now observe the behavior of the DD in a sufficiently coarse way by just looking at the points in time where the computations have been cut (after reading roughly $i \cdot kn/r$ variables, $i = 0, \dots, r$). Next, we want to find rectangles in a decision forest. For that, we assign each of the decision trees to Alice and Bob randomly with probability 1/2 for both of them. Typically, many variables read during a computation will occur in both Alice's and Bob's trees. But we can nevertheless hope that there will be some variables that occur exclusively in Alice's trees and some that will be only in Bob's trees, resp. This is because each of the trees reads only kn/r variables during a computation, and thus there are not too many pairs of variables that occur together in one of the trees. We make this precise now.

Definition 9.3: For a decision forest F and a subforest $F' \subseteq F$, let $\mathtt{core}_F(a, F')$ be the set of variables read *exclusively* in the trees in F' during the computation for a, called *F'-core of a* (in F).

We usually omit the index F since we only consider one decision forest at a time. The random assignment of decision trees in a decision forest F to Alice and Bob yields a random partition (F_A, F_B) of F. We show that $\texttt{core}(a, F_A)$ and $\texttt{core}(a, F_B)$ will be not too small with high probability.

Lemma 9.4: Let F be an (r, k)-decision forest with $r \leq n$ and let $a \in \{0,1\}^n$ be an input. Let (F_A, F_B) be a random partition of F as described above. Then $\texttt{core}(a, F_A)$ and $\texttt{core}(a, F_B)$ have the same expected size $\mu(a) \geq n/2^{k+1}$ and $\big||\texttt{core}(a, F_A)| - \mu(a)\big| \geq \mu(a)/2$ with probability at most $4(k+1)^2 2^{2(k+1)}/r$, analogously for F_B.

Proof. By symmetry, it suffices to consider F_A. Let $t(i)$ be the number of trees in F that access variable x_i during the computation for a. Let Z_i be the indicator variable for the event $x_i \in \texttt{core}(a, F_A)$ and let $Z = Z_1 + \cdots + Z_n$. We have $\Pr\{Z_i = 1\} = 2^{-t(i)}$ for all i and $\mu(a) = E(Z) = \sum_{1 \leq i \leq n} 2^{-t(i)}$. Since F is an (r, k)-decision tree, $t(1) + \cdots + t(n) \leq r\lceil kn/r \rceil \leq (k+1)n$. It is easy to see that the minimum of the convex function $\sum_{1 \leq i \leq n} 2^{-t(i)}$ of the $t(i)$ as variables under the constraint $t(1) + \cdots + t(n) \leq (k+1)n$ is $n/2^{k+1}$. This proves the first part of the claim. For the second part, we first upper bound the variance of Z,

$$V(Z) = \sum_{1 \leq i,j \leq n} (\Pr\{Z_i = 1 \wedge Z_j = 1\} - \Pr\{Z_i = 1\} \cdot \Pr\{Z_j = 1\}).$$

If there is no tree that reads both the variables x_i and x_j, then Z_i and Z_j are independent and the respective term in the above sum is zero. We crudely estimate all other terms with 1 and count their number. For a fixed i, there are at most $t(i) \cdot \lceil kn/r \rceil$ variables x_j that are read together with x_i in one of the trees. Thus, altogether, there are at most $\sum_{1 \leq i \leq n} t(i)\lceil kn/r \rceil \leq (k+1)^2 n^2/r$ non-zero terms in the sum which is also an upper bound for $V(Z)$. By Chebyshev's inequality, we obtain

$$\Pr\big\{|Z - E(Z)| \geq E(Z)/2\big\} \leq 4V(Z)/E(Z)^2 \leq 4(k+1)^2 2^{2(k+1)}/r.$$

□

Let F be a fixed decision forest. Set $r = 16(k+1)^2 2^{2(k+1)}$. Then for each input a, the above lemma yields a fixed partition (F_A, F_B) of F such that for $A := \texttt{core}(a, F_A)$ and $B := \texttt{core}(a, F_B)$, we have $|A|, |B| \geq m$ with $m = n/2^{k+2}$. We are now essentially in the same situation as for oblivious BDDs in Section 8 where we had obtained suitably large subsets A and B of the variables for the two players.

As for oblivious BDDs, we want to fix the variables outside A and B. We first group together inputs for which the same sets A and B are suitable (there is no longer one fixed choice for all inputs here). For two disjoint sets of variables A, B, let $Q = Q(A, B, F_A, F_B)$ be the set of inputs $a \in F^{-1}(1)$ with $A \subseteq \texttt{core}(a, F_A)$ and $B \subseteq \texttt{core}(a, F_B)$. Each input $a \in F^{-1}(1)$ is contained in such a set with $|A| = |B| = m$ by Lemma 9.4. Notice that usually there will be more than one such set covering a given input. Let c be any partial assignment to all variables in $X - (A \cup B)$, and let Q_c be the set of all partial assignments to $A \cup B$ that together with c form an assignment in Q. We show that Q_c is a (possibly empty) rectangle. For assignments u and v to disjoint sets of variables let uv denote the joint assignment to the union of these sets. Let $Q_{c,A}$ be the set of all assignments a to A such that, for all assignments b_0 to B, $F_A(ab_0c) = 1$ and $\mathtt{core}(ab_0c, F_A) \supseteq A$, and let $Q_{c,B}$ be the set of all assignments b to B such that, for all assignments a_0 to A, $F_B(a_0bc) = 1$ and $\mathtt{core}(a_0bc, F_B) \supseteq B$. Then $Q_c = Q_{c,A} \times Q_{c,B}$. Thus, we can first cover all inputs in $F^{-1}(1)$ by the sets $Q(A, B, F_A, F_B)$ and then partition each of these sets into rectangles. It remains to quantify this approach.

We first extend our notion of rectangles to include fixed input parts that do not belong to any of the two players and introduce a name for sets like $Q(A, B, F_A, F_B)$.

Definition 9.5: Let $A, B \subseteq X$ be disjoint sets with $|A| = |B| = m$. A set of inputs R is called *m-rectangle with respect to* (A, B) if there is an assignment c to $X - (A \cup B)$ and there are sets of assignments R_A and R_B to A and B, resp., such that $R = R_A \times R_B \times \{c\}$. A set of inputs Q is called *m-pseudo-rectangle with respect to* (A, B) if for each assignment c to $X - (A \cup B)$, $Q_c \times \{c\}$ is an m-rectangle with respect to (A, B).

Lemma 9.6: Let $k \le n$, $r = 16(k+1)^2 2^{2(k+1)} \le n$, and $m = n/2^{k+2}$. Let F be an (r, k)-decision forest. Then there is a family of at most $2^{4(k+2)m+r}$ m-pseudo-rectangles that cover $F^{-1}(1)$.

Proof. It only remains to prove the upper bound on the number of pseudo-rectangles. Each input $a \in F^{-1}(1)$ is contained in a set $Q(A, B, F_1, F_2)$ with $|A| = |B| = m$ due to Lemma 9.4, and there are at most $2^r \binom{n}{m}^2$ such sets. Using that $\binom{n}{m} \le 2^{H(m/n)n}$ with $H(x) = -(x \log x + (1-x)\log(1-x))$ and the Taylor series approximation $H(x) \le -2x \log x$ for $x \le 1/2$, we get $2^r \binom{n}{m}^2 \le 2^{4(k+2)m+r}$. □

Analogously to the rectangle size method for the uniform distribution, we obtain the following connection between the density of 1-colored rectangles and the size of DDs:

Theorem 9.7: Let $2 \le k \le n$, $r = 16(k+1)^2 2^{2(k+1)} \le n$, and $m = n/2^{k+2}$. Let G be a DD for $f\colon D^n \to \{0, 1\}$ of length at most kn. Then there is an m-rectangle $R \subseteq f^{-1}(1)$ such that $|R|/|D|^{2m} \ge 2^{-4(k+2)m-r} \cdot (1/|G|)^r \cdot |f^{-1}(1)|/|D|^n$.

Proof. Applying Lemma 9.2 and Lemma 9.6, we obtain a cover of $f^{-1}(1)$ by at most $2^{4(k+2)m+r} \cdot |G|^r$ m-pseudo-rectangles. Each of these pseudo-rectangles can be partitioned into m-rectangles as described above. By averaging, it follows that there is at least one such rectangle with the required density. □

Beame, Jayram, and Saks (2001) have applied this theorem to *quadratic form functions*. We consider one concrete example. Let $n = 2^d$ and let S be the $n \times n$-Sylvester matrix defined by $S_{a,b} = (-1)^{a^\top b}$ for $a, b \in \{0, 1\}^d$. Let S^* be the matrix obtained from S by replacing the diagonal with zeros. For any odd prime power q and $x \in \mathbb{Z}_q^n$, let $\mathrm{SQF}_{q,n}(x) = 1$ if $x^\top S^* x \equiv 0 \bmod q$.

The matrix S has the remarkable property that each of its submatrices has large rank compared to its size. In Beame, Jayram, and Saks (2001), a bound for the submatrix rank of S and elementary algebra have been used to prove that, for any m-rectangle $R \subseteq \mathrm{SQF}_{q,n}^{-1}(1)$, $|R|/|D|^{2m} \leq |D|^{-m^2/n}$. Furthermore, it is easy to see that even after fixing $n-2$ variables of x, the function $x^\top S^* x$ can still attain any value in $\mathbb{Z}_q$. Thus, $|\mathrm{SQF}_{q,n}^{-1}(1)| \geq q^{n-2}$. Plugging these facts into Theorem 9.7, we get:

Theorem 9.8: For any constant $\varepsilon > 0$ there is a constant $c > 0$ such that for k, n, q with $\log\log q \geq ck$ and $n \geq 8(k+1)^2 2^{2(k+1)}$, any DD for $\mathrm{SQF}_{q,n}$ of length kn requires size at least $2^{n \log^{1-\varepsilon} q}$.

To illustrate this result further, we choose q as the smallest prime greater than or equal to n and set k to roughly $\log\log n$. Furthermore, we use the well-known correspondence between length and size of DDs and time and space for register machines (see, e. g., Borodin and Cook (1982)).

Corollary 9.9: For any constant $\varepsilon > 0$ there is a constant $c' > 0$ such that each deterministic algorithm for $\mathrm{SQF}_{q,n}$ on a register machine with space $n \log^{1-\varepsilon} n$ requires time at least $c' n \log\log n$.

Theorem 9.7 only makes sense for large domains D that grow with the input length. In the boolean case $D = \{0,1\}$, the number of pseudo-rectangles eats up even the best possible upper bound of 2^{-2m} on the rectangle density. By a further, even more sophisticated variant of the rectangle method, Ajtai (1999a, 1999b) and Beame, Saks, Sun, and Vee (2000), have managed to overcome this problem. They have been able to show the existence of a rectangle $R = R_A \times R_B \times \{c\} \subseteq f^{-1}(1)$, where R_A and R_B are sets of assignments to disjoint sets A and B, with the following properties:

(i) $|A|, |B| \geq m$ for some large m.

(ii) There is a constant ε with $0 < \varepsilon < 1$ such that both densities $|R_A|/|D|^{|A|}$ and $|R_B|/|D|^{|B|}$ satisfy the lower bound $2^{-\varepsilon m} \cdot (1/|G|^r) \cdot |f^{-1}(1)|/|D|^n$.

Being able to bound the densities of *both* parts of the rectangle separately makes the approach applicable to a larger class of functions. Actually, Beame, Saks, Sun, and Vee (2000) have even managed to show that all but a small fraction of the accepted inputs of f can be covered by rectangles with the properties (i) and (ii), and, furthermore, that these rectangles do not overlap too much. Although one only needs a single rectangle for the deterministic case, the stronger result also allows the method to be used for proving exponential lower bounds on the size of randomized length-restricted BDDs (see next section).

The first result for deterministic BDDs (and, therefore, boolean inputs) of linear length has been achieved by Ajtai (1999b) for a quadratic form function similar to SQF but based on modified Hankel matrices, for which Ajtai has obtained an especially strong bound on the rank of submatrices. Furthermore, he has proved (1999a) deterministic lower bounds for the practically interesting functions *element distinctness* (check whether there are two identical numbers in a list) and *Hamming closeness* (check whether there are two vectors in a list that have small Hamming distance). The latter two results are again for the large domain case.

10 Randomized BDDs

Many practically relevant complexity theoretical questions regarding randomized algorithms are still open up today, e. g., we even do not know whether randomization helps at all to solve more problems in polynomial time compared to deterministic algorithms. BDDs allow to study such questions in the scenario where space is the primary resource. We look at examples of upper and lower bounds for randomized oblivious BDDs and discuss a proof method and results for randomized length-restricted BDDs, the so-far most general restricted type of BDDs that can be handled by lower bound methods.

For OBDDs, it is easy to prove that allowing randomization can indeed lead to exponential savings in size (Ablayev and Karpinski (1996)). As a simple example, we consider the bit mask version of the equality function, EQ_n^*, from Section 8. Recall that nondeterministic oblivious OBDDs of linear length for EQ_n^* require exponential size.

Theorem 10.1: The complement of EQ_n^* can be computed by a randomized OBDD of polynomial size with one-sided error probability $1/n$.

Proof. The construction is based on the so-called *fingerprinting technique.* To compare objects from a large universe we map these objects to "fingerprints" or "hash codes" from a small universe that are efficiently comparable. Using randomization, we can ensure that the probability of different objects having the same fingerprint is small.

The function EQ_n^* is defined on vectors $a, b, x, y \in \{0,1\}^n$. Recall that $\mathrm{EQ}_n^*(a,b,x,y) = 1$ if the sub-vectors of x' of x and y' of y chosen by a and b, resp., are equal. The randomized OBDD G for $\overline{\mathrm{EQ}_n^*}$ uses the variable order $a_1, x_1, b_1, y_1, \ldots, a_n, x_n, b_n, y_n$ and works as follows. By a tree of randomized nodes at the top of the OBDD, a prime from the set of the first n^2 primes is chosen. For a prime p, a subprogram computes the fingerprints $h_{x',p} = \left(\sum_{1\le i\le n} x'_i 2^{i-1}\right) \bmod p$ and $h_{y',p} = \left(\sum_{1\le i\le n} y'_i 2^{i-1}\right) \bmod p$. This can be done by storing the intermediate results with p^2 nodes per level in the OBDD. The OBDD accepts an input if $h_{x',p} \ne h_{y',p}$.

If $x' = y'$, $h_{x',p} = h_{y',p}$ for each p and the OBDD G correctly rejects the input. Let $x' \ne y'$. Since $\left|\sum_{1\le i\le n} x'_i 2^{i-1} - \sum_{1\le i\le n} y'_i 2^{i-1}\right| \le 2^n - 1$, there are fewer than n primes p with $h_{x',p} = h_{y',p}$. Hence, $\Pr\{G(a,b,x,y) = 1\} < n/n^2 = 1/n$. Using that each of the first n^2 primes is at most of size $O(n^2 \log n)$ by the prime number theorem, it is easy to prove that the OBDD is of polynomial size. $\square$

By the same technique, the complement of the permutation matrix test PERM can be shown to have randomized OBDDs of polynomial size with small one-sided error (Sauerhoff (1998)), while we know that PERM requires exponential size even for nondeterministic FBDDs (Jukna (1988) and Krause, Meinel, and Waack (1991)).

The indirect storage access function ISA is a typical example for a function that is hard for deterministic OBDDs (recall the definition from Section 7). Since variables may be read only once in a fixed order and since each x-variable may occur as a bit in $x(y)$ as well as the output bit, a deterministic OBDD has to store a large number of bits and thus requires exponential size. The function can be easily computed by a nondeterministic OBDD of polynomial size (see Section 8). Nevertheless, randomization does not help here (Sauerhoff (2001)).

Theorem 10.2: Each randomized OBDD with arbitrary two-sided error ε, $0 < \varepsilon < 1/2$, for ISA_n has exponential size.

Proof. Consider a randomized OBDD for ISA_n with an arbitrary variable order given as a list. Cut the list in two parts such that the first part contains $\ell = b - 1$ of the x-variables, where $b = \Theta(n/\log n)$ is the number of blocks $x(y)$. Then there is at least one block $x(y_0)$ that lies completely in the second part. Give the first and last part of the list to Alice and Bob, resp., and set the y-variables to y_0 in the randomized OBDD. Then the two players can solve the following *direct storage access problem*: Alice has an ℓ-bit "memory" vector (her x-variables), Bob has an "address" in this memory (encoded in $x(y)$), and Bob has to output the addressed bit after receiving only a single message from Alice (encoded as the number of an OBDD node). In communication complexity theory, it is proved that Alice essentially has to tell Bob her complete memory contents for solving this task, i. e., $\Omega(\ell) = \Omega(n/\log n)$ bits of communication are required even in the randomized case. This amount of information has to be encoded by nodes in the OBDD, leading to size $2^{\Omega(n/\log n)}$. □

Applying suitable lower bounds for one-round communication protocols as in the proof of Theorem 10.2, several other functions have been shown to require exponentially large randomized OBDDs. For general oblivious BDDs, lower bounds for randomized communication protocols with several rounds are required. As one practical example, we consider the multiplication function from Section 8. By the same ideas as in Section 8, each randomized oblivious BDD of length $2kn$ for the multiplication of n-bit numbers yields a randomized k-round communication protocol for the carry bit problem of input length $r = \Theta(n/((k+1)^2 2^{4k}))$ described in Section 8. Smirnov (1988) has shown that each randomized k-round communication protocol for the latter problem with error bounded by a constant smaller than $1/2$ requires $\Omega(r^{1/k} \log r)$ bits of communication. This implies an exponential lower bound on the size of randomized oblivious BDDs of linear length for the multiplication function.

Upper and lower bound results have also been proved for randomized variants of FBDDs and syntactic read-k BDDs for $k > 1$. We skip these results and devote the rest of the section to randomized length-restricted DDs.

We sketch a variant of the proof method from Section 9 that also works for the randomized case. We again consider a function $f\colon D^n \to \{0, 1\}$. We say that a deterministic DD G *approximates f with error ε* if the output of G agrees with f on at least a $(1-\varepsilon)$-fraction of all inputs with respect to the uniform distribution on the inputs. By a generally applicable counting argument, we may consider approximating DDs rather than randomized DDs.

Lemma 10.3: Let G be a randomized DD that computes f with two-sided error ε. Then there is a deterministic DD G' with $|G'| \le |G|$ that approximates f with error at most ε.

Proof. Let $G_1, \ldots, G_N$ be the different deterministic DDs obtained by selecting one outgoing edge of each randomized node in G and removing the other one. Let G_i also denote the function represented by G_i. For each input $a \in D^n$, let $d(i,a) = 1$ if $G_i(a) \neq f(a)$ and 0 otherwise. Then $\frac{1}{N}\sum_{1\le i\le N} d(i,a) \le \varepsilon$ for each a due to the error bound of G, and thus

$$|D|^{-n} \sum_{a\in D^n} \frac{1}{N} \sum_{1\le i\le N} d(i,a) = \frac{1}{N} \sum_{1\le i\le N} |D|^{-n} \sum_{a\in D^n} d(i,a) \le \varepsilon.$$

This implies that there is an i_0 with $|D|^{-n}\sum_{a\in D^n} d(i_0,a) \le \varepsilon$. □

In Section 9, we have considered a deterministic DD for f and have obtained a cover of the accepted inputs of f by pseudo-rectangles. We have shown that the number of pseudo-rectangles is not too large and have argued that this implies the existence of one dense pseudo-rectangle that contains only accepted inputs. Here we consider a deterministic DD G approximating the given function f with error ε and our aim is to produce a rectangle that is dense and contains only few inputs that are not accepted by f. Let G also denote the function represented by G.

Since G is deterministic, Lemma 9.2 yields a decomposition of $G^{-1}(1)$ into disjoint sets of inputs accepted by decision forests. Instead of covering the inputs accepted by each decision forest by overlapping pseudo-rectangles as in Section 9, we directly partition these sets into rectangles (we omit the details how this can be done). This yields a partition of the whole set $G^{-1}(1)$ into rectangles. By averaging arguments, it can then be shown that at least half of all accepted inputs of G can be partitioned into dense rectangles (again we omit the details). Due to the error bound of G, f is 0 for at most $\varepsilon|D|^n$ inputs in the obtained rectangle partition. On the other hand, these rectangles contain at least $(|f^{-1}(1)| - \varepsilon|D|^n)/2$ accepted inputs altogether. Hence, again by averaging, there is at least one dense rectangle R such that f is 0 for at most a $2\varepsilon/(|f^{-1}(1)|\cdot|D|^{-n} - \varepsilon)$-fraction of the inputs in R. Altogether, we arrive at the following analog of the method from Section 9 for approximating DDs.

Theorem 10.4: Let k, n, r be suitable integer parameters. Let G be a deterministic DD of length at most kn that approximates $f\colon D^n \to \{0,1\}$ with error ε. Then there is an m-rectangle R with respect to variable sets A and B such that

(i) $|R|/|D|^{|A|}, |R|/|D|^{|B|} \ge 2^{-12(k+1)m} \cdot (1/|G|^r) \cdot (\eta - \varepsilon)$, where $\eta = |f^{-1}(1)|/|D|^n$ and

(ii) f equals 0 for at most a $2\varepsilon/(\eta-\varepsilon)$-fraction of the inputs in R.

We apply this theorem to the following variant of the Sylvester matrix function from Section 9. Let p be a prime, let $M \subseteq \mathbb{Z}_p$ with $|M| = \lfloor p/2 \rfloor$, and let S be the $n\times n$-Sylvester matrix as in Section 9. Define the function BSQF (*balanced* SQF) for $x \in \mathbb{Z}_p^n$ by $\mathrm{BSQF}_{p,M,n}(x) = a$ if $x^\top S x \in M$ and $1-a$ otherwise, where $a \in \{0,1\}$ is chosen such that $|\,\mathrm{BSQF}^{-1}_{p,M,n}(1)| \ge 1/2$. Beame, Saks, Sun, and Vee (2002) have shown that for each m-rectangle $R \subseteq \mathbb{Z}_p^n$ with $|R|/|D|^{2m} \ge |D|^{-m^2/(2n)+3}$ and any $c \in \mathbb{Z}_p$ the fraction of inputs $x \in R$ with $x^\top S x \equiv c \bmod p$ is at least $1/(4p)$. It follows that each dense rectangle contains a large fraction of inputs from $\mathrm{BSQF}_{p,M,n}^{-1}(0)$. For $p \geq n$, Theorem 10.4 and simple calculations yield the following result:

Theorem 10.5: For any constant $\varepsilon > 0$, each deterministic DD of size $2^{n^{1-\varepsilon}}$ that approximates $\mathrm{BSQF}_{p,M,n}$ with error at most $1/50$ has length $\Omega(n \log\log n)$.

Similar results have been obtained by Beame, Saks, Sun, and Vee (2000, 2002) for quadratic form functions with other matrices. The best lower bound on the length is of order $\Omega(n \log n)$ for DDs of size $2^{n^{1-\varepsilon}}$ with error $1/100$. As a more practically relevant example, we finally mention their particularly nice result for the computation of the element distinctness function on register machines which is a direct consequence of an analogous DD result. Let $\mathrm{ED}_n(x_1, \ldots, x_n) = 1$ if the numbers $x_1, \ldots, x_n \in \{1, \ldots, n^2\}$ are pairwise distinct and 0 otherwise.

Theorem 10.6: For any $\varepsilon > 0$ there is a constant $c_\varepsilon > 0$ such that any randomized algorithm that computes ED_n with error $n^{-\varepsilon}$ and runs on a register machine with space $n^{1-\varepsilon}$ requires time at least $c_\varepsilon n \sqrt{\log n / \log\log n}$.

All above results are for functions on a large domain D growing with the input length. In the boolean case $D = \{0, 1\}$, Beame, Saks, Sun, and Vee (2000) have used a more sophisticated version of the method described above (see also the end of Section 9) to obtain time-space trade-offs for randomized and approximating BDDs representing quadratic form functions based on Hankel matrices.

References

[1] Ablayev, F. and Karpinski, M. (1996). On the power of randomized branching programs. In *Proc. of 23rd ICALP, LNCS 1099*, 348–356.

[2] Ajtai, M. (1999a). Determinism versus non-determinism for linear time RAMs with memory restrictions. In *Proc. of 31st STOC*, 632–641.

[3] Ajtai, M. (1999b). A non-linear time lower bound for boolean branching programs. In *Proc. of 40th FOCS*, 60–70.

[4] Alon, N. and Maass, W. (1988). Meanders and their applications in lower bound arguments. *Journal of Computer and System Sciences 37*, 118–129.

[5] Beame, P., Jayram, T. S., and Saks, M. (2001). Time-space tradeoffs for branching programs. *Journal of Computer and System Sciences 63*, 542–572.

[6] Beame, P., Saks, M., Sun, X., and Vee, E. (2000). Super-linear time-space tradeoff lower bounds for randomized computation. In *Proc. of 41st FOCS*, 169–179.

[7] Beame, P., Saks, M., Sun, X., and Vee, E. (2002). Time-space tradeoff lower bounds for randomized computation of decision problems. Manuscript.

[8] Bern, J., Meinel, C., and Slobodová, A. (1996). Some heuristics for generating tree-like FBDD types. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems 15*, 127–130.

[9] Blum, M., Chandra, A. K., and Wegman, M. N. (1980). Equivalence of free boolean graphs can be decided probabilistically in polynomial time. *Information Processing Letters 10*, 80–82.

[10] Bollig, B., Sauerhoff, M., Sieling, D., and Wegener, I. (1998). Hierarchy theorems for kOBDDs and kIBDDs. *Theoretical Computer Science 205*, 45–60.

[11] Bollig, B. and Wegener, I. (1996). Improving the variable ordering of OBDDs is NP-complete. *IEEE Trans. on Computers 45*, 993–1002.

[12] Bollig, B. and Wegener, I. (1999). Complexity theoretical results on partitioned (nondeterministic) binary decision diagrams. *Theory of Computing Systems 32*, 487–503.

[13] Borodin, A. and Cook, S. (1982). A time-space tradeoff for sorting on a general sequential model of computation. *SIAM Journal on Computing 11*, 287–297.

[14] Borodin, A., Razborov, A. A., and Smolensky, R. (1993). On lower bounds for read-k-times branching programs. *Computational Complexity 3*, 1–18.

[15] Bryant, R. E. (1986). Graph-based algorithms for boolean function manipulation. *IEEE Trans. on Computers 35*, 677–691.

[16] Bryant, R. E. (1991). On the complexity of VLSI implementations and graph representations of boolean functions with application to integer multiplication. *IEEE Trans. on Computers 40*, 205–213.

[17] Bryant, R. E. (1995). Binary decision diagrams and beyond: enabling technologies for formal verification. In *Proc. of International Conference on Computer-Aided Design*, 236–243.

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

[19] Fortune, S., Hopcroft, J., and Schmidt, E. M. (1978). The complexity of equivalence and containment for free single variable program schemes. In *Proc. of 5th ICALP, LNCS 62*, 227–240.

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

[21] Gergov, J. and Meinel, C. (1994). Efficient boolean manipulation with OBDDs can be extended to FBDDs. *IEEE Trans. on Computers 43*, 1197–1209.

[22] Giel, O. (2001). Branching program size is almost linear in formula size. *Journal of Computer and System Sciences 63*, 222–235.

[23] Hromkovič, J. (1997). *Communication Complexity and Parallel Computing.* Springer-Verlag.

[24] Jain, J., Bitner, J., Fussell, D. S., and Abraham, J. A. (1992). Functional partitioning for verification and related problems. *Brown MIT VLSI Conf.*, MIT Press, Cambridge, MA 45, 210–226.

[25] Jain, J., Mohanram, K., Moundanos, D., Wegener, I., and Lu, Y. (2000). Analysis of composition, and how to obtain smaller canonical graphs. In *Proc. of 37th Design Automation Conference*, 681–686.

[26] Johnson, D. S. (1990). A catalog of complexity classes. In *Handbook of Theoretical Computer Science, Vol. A*, van Leeuwen, J., ed., Elsevier, Amsterdam, Chapter 2, 67–161.

[27] Jukna, S. P. (1988). Entropy of contact circuits and lower bounds on their complexity. *Theoretical Computer Science 57*, 113–129.

[28] Krause, M., Meinel, C., and Waack, S. (1991). Separating the eraser Turing machine classes L_e, NL_e, co-NL_e and P_e. *Theoretical Computer Science 86*, 267–275.

[29] Kushilevitz, E. and Nisan, N. (1997). *Communication Complexity.* Cambridge University Press.

[30] Lai, Y.-P., Pedram, M., and Vrudhula, S. B. K. (1993). BDD based decomposition of logic functions with application to FPGA synthesis. In *Proc. of 30th Design Automation Conference*, 642–647.

[31] Narayan, A., Jain, J., Fujita, M., and Sangiovanni-Vincentelli, A. (1996). Partitioned ROBDDs – a compact, canonical, and efficiently manipulable representation for Boolean functions. In *Proc. of International Conference on Computer-Aided Design*, 547–554.

[32] Nechiporuk, È. I. (1966). A boolean function. *Soviet Mathematics Doklady 7*, 999–1000.

[33] Nisan, N. and Wigderson, A. (1993). Rounds in communication complexity revisited. *SIAM Journal on Computing 22*, 211–219.

[34] Okol'nishnikova, E. A. (1993). On lower bounds for branching programs. *Siberian Advances in Mathematics 3*, 152–166.

[35] Pudlák, P. and Žák, S. (1983). Space complexity of computations. Preprint Univ. Prague.

[36] Rudell, R. (1993). Dynamic variable ordering for ordered binary decision diagrams. In *Proc. of International Conference on Computer-Aided Design*, 42–47.

[37] Sauerhoff, M. (1998). Lower bounds for randomized read-k-times branching programs. In *Proc. of 15th STACS*, *LNCS 1373*, 105–115.

[38] Sauerhoff, M. (2000). An improved hierarchy result for partitioned BDDs. *Theory of Computing Systems 33*, 313–329.

[39] Sauerhoff, M. (2001). On the size of randomized OBDDs and read-once branching programs for k-stable functions. *Computational Complexity 10*, 155–178.

[40] Sauerhoff, M., Wegener, I., and Werchner, R. (1999). Relating branching program size and formula size over the full binary basis. In *Proc. of 16th STACS*, *LNCS 1563*, 57–67.

[41] Sieling, D. (2002a). The nonapproximability of OBDD minimization. *Information and Computation 172*, 103–138.

[42] Sieling, D. (2002b). The complexity of minimizing and learning OBDDs and FBDDs. *Discrete Applied Mathematics 122*, 263–282.

[43] Sieling, D. and Wegener, I. (1993a). *NC*-algorithms for operations on binary decision diagrams. *Parallel Processing Letters 3*, 3–12.

[44] Sieling, D. and Wegener, I. (1993b). Reduction of OBDDs in linear time. *Information Processing Letters 48*, 139–144.

[45] Sieling, D. and Wegener, I. (1995). Graph driven BDDs – a new data structure for boolean functions. *Theoretical Computer Science 141*, 283–310.

[46] Smirnov, D. V. (1988). *Shannon's Information Methods for Lower Bounds for Probabilistic Communication Complexity.* Master's thesis, Moscow University.

[47] Somenzi, F. (2001). Efficient manipulation of decision diagrams. *Software Tools for Technology Transfer 3*, 171–181.

[48] Wegener, I. (1988). On the complexity of branching programs and decision trees for clique functions. *Journal of the ACM 35*, 461–471.

[49] Wegener, I. (2000). *Branching Programs and Binary Decision Diagrams – Theory and Applications.* SIAM Monographs on Discrete Mathematics and Applications.

[50] Žák, S. (1984). An exponential lower bound for one-time-only branching programs. In *Proc. of 11th MFCS, LNCS 176*, 562–566.