Electronic Colloquium on Computational Complexity, Report No. 1 (1999)

# The Complexity of Minimizing FBDDs

**Detlef Sieling**[1]
FB Informatik, LS II, Univ. Dortmund
44221 Dortmund, Fed. Rep. of Germany
sieling@ls2.cs.uni-dortmund.de

**Abstract** Free Binary Decision Diagrams (FBDDs) are a data structure for the representation and manipulation of Boolean functions. Efficient algorithms for most of the important operations are known if only FBDDs respecting a fixed graph ordering are considered. However, the size of such an FBDD may strongly depend on the chosen graph ordering and efficient algorithms for computing good or optimal graph orderings are not known. In this paper it is shown that the existence of polynomial time approximation schemes for optimizing graph orderings or for minimizing FBDDs implies NP = ZPP or NP = P, respectively, and so such algorithms are quite unlikely to exist.

## 1. Introduction

Many variants of Binary Decision Diagrams (BDDs) have been considered as a data structure for Boolean functions. Such data structures have several applications, in particular in computer aided hardware design. They are used in programs for, e.g., circuit verification, test pattern generation, model checking and logic synthesis. Data structures for Boolean functions should allow the efficient representation and manipulation of important functions. The most popular data structure proposed for this purpose are Ordered Binary Decision Diagrams (OBDDs), which were introduced by Bryant [5, 6]. The reason that many generalizations of OBDDs have been considered is that there are many practically important functions for which OBDDs are too large to be stored in a computer memory. In this paper we focus on a particular extension of OBDDs, namely Free BDDs (FBDDs).

FBDDs have also been considered in complexity theory under the name read-once branching programs. There are many papers presenting lower bound methods for FBDDs. The first ones are due to Wegener [20] and Žák [21], and in the paper of Simon and Szegedy [18] most previous approaches are handled in a unified way. Already in the early paper of Fortune, Hopcroft and Schmidt [7] it was shown that FBDDs are exponentially more powerful than OBDDs by presenting an example of a function with polynomial FBDD size but exponential OBDD size. The algorithmic aspects of FBDDs are investigated by Sieling and Wegener [17] and Gergov and Meinel [9]. It turned out that many but not all operations on Boolean functions which can be performed efficiently on functions represented by OBDDs can also be performed efficiently

[1]Supported in part by DFG grant We 1066/8.

ISSN 1433-8092

on functions represented by FBDDs if only FBDDs according to a fixed graph ordering are considered. This is similar to OBDDs where many operations can be performed efficiently only if the considered OBDDs have the same variable ordering. Graph orderings are a generalization of variable orderings. A graph ordering $G$ defines for each input the ordering in which the variables have to be tested in FBDDs respecting $G$. Unlike OBDDs FBDDs allow different orderings for different inputs. FBDDs respecting a graph ordering $G$ are called $G$-FBDDs or $G$ driven FBDDs.

We postpone the formal definition of graph orderings to Section 2. Here we only remark that (similar to OBDDs) the size of a $G$-FBDD for a particular function may strongly depend on the chosen graph ordering $G$. So it is an important problem to choose a good graph ordering. A heuristic for computing graph orderings of a tree-like shape has been proposed by Bern, Meinel and Slobodová [2]. An algorithm with a double exponential worst-case run time for minimizing FBDDs was presented by Günther and Drechsler [10]. This algorithm can also be used to compute optimal graph orderings, since for each FBDD $H$ a graph ordering $G$ can easily be computed so that $H$ is a $G$-FBDD, see Sieling and Wegener [17]. However, the question whether there are efficient algorithms for computing good or optimal graph orderings remains open. In this paper we consider the following two closely related optimization problems.

MinGraphOrdering

**Instance:** A Boolean function $f$ described by an FBDD $G$.
**Problem:** Compute a graph ordering $G^*$ so that the size of a $G^*$-FBDD for $f$ is minimal among all FBDDs for $f$.

MinFBDD

**Instance:** A Boolean function $f$ described by an FBDD $G$.
**Problem:** Compute an FBDD for $f$ which has minimal size.

We shall prove the following hardness results for these problems.

**Theorem 1:** *If there is a polynomial time approximation scheme for MinFBDD, then NP $=$ P.*

**Theorem 2:** *If there is a polynomial time approximation scheme for MinGraphOrdering, then NP $=$ ZPP.*

We remember that ZPP is the class of languages with error-free probabilistic polynomial time algorithms (Las Vegas algorithms). For an introduction into the probabilistic complexity classes see, e.g., the textbook of Papadimitriou [12]. Since it seems to be quite unlikely that all problems in NP have deterministic polynomial time algorithms or polynomial time Las Vegas algorithms, it is also quite unlikely that the two considered problems have polynomial time approximation schemes.

We remark that for OBDDs there are similar optimization problems, namely the computation of a minimal size OBDD for a function given by an OBDD and the computation of an optimal variable ordering for a function given by an OBDD. However, for OBDDs these problems are polynomially related and, therefore, usually not explicitly distinguished. The NP-hardness of these problems for OBDDs for multi-output functions was shown be Tani, Hamaguchi and Yajima [19] and for OBDDs for single-output functions by Bollig and Wegener [4]. In Sieling

[15] it is shown that the existence of polynomial time approximation schemes for these problems implies P $=$ NP and in Sieling [16] the stronger result that it is NP-hard to approximate these problems up to any constant factor.

The paper is organized as follows. In the following section we repeat the definitions and some basic properties of FBDDs and approximation schemes. The theorems are proven in Sections 3 and 4, and a technical lemma used in these proofs is shown in Section 5.

## 2. Definitions and Some Properties of FBDDs

We start with the definitions of FBDDs and graph orderings. A Binary Decision Diagram (BDD) for the representation of functions $f_1, \ldots, f_m$ over the variables $x_1, \ldots, x_n$ is a directed acyclic graph. The graph consists of terminal nodes, which have no successor and which are labeled by $0$ or $1$, and internal nodes. Each internal node is labeled by a variable and has an outgoing $0$-edge and an outgoing $1$-edge. In free BDDs (FBDDs) on each directed path each variable occurs at most once as the label of a node.

Examples of FBDDs are shown on the right side of Figure 2. In the figures edges are always directed downwards. We draw $0$-edges as dashed lines and $1$-edges as solid lines. Internal nodes are drawn as circles and terminal nodes as squares.

Each node $v$ of an FBDD represents a Boolean function $f_v$. In order to evaluate this function for an input $a = (a_1, \ldots, a_n)$ we start at $v$. At each $x_i$-node we follow the outgoing $a_i$-edge. Finally, a terminal node is reached, and $f_v(a)$ is equal to the label of this terminal node. In an FBDD for the representation of $f_1, \ldots, f_m$ for each function $f_j$ there is a pointer to a node representing $f_j$.

A graph ordering is a BDD-like graph that does not represent a Boolean function but that describes for each input a permutation of the variables. Formally, a graph ordering $G$ is a directed acyclic graph with one source node and one terminal node. Each internal node is labeled by a Boolean variable and has an outgoing $0$-edge and an outgoing $1$-edge. Furthermore, on each path from the source to the terminal node each variable is tested exactly once. Similar to FBDDs each input $a = (a_1, \ldots, a_n)$ defines a path from the source to the terminal node of the graph ordering.

For a graph ordering $G$ we call an FBDD $G'$ a $G$-FBDD or $G$ driven FBDD if for each input the variables on the computation path in $G'$ are found in the same ordering as on the computation path in $G$, where on the computation path in $G'$ variables may be omitted.

We repeat some properties of FBDDs. The usual reduction rules for OBDDs can be applied also on FBDDs without changing the represented function. The reduction rules are depicted in Figure 1. By the deletion rule a node $v$ whose successors coincide can be deleted after redirecting the edges leading to $v$ to its successor. By the merging rule nodes $v$ and $w$ with the same label, the same $0$-successor and the same $1$-successor can be merged, i.e., the edges leading to $v$ are redirected to $w$, and $v$ is deleted. An FBDD is called reduced if neither of the reduction rules is applicable. Sieling and Wegener [17] prove that reduced $G$-FBDDs are unique up to isomorphism. Hence, we may talk about *the* (reduced) $G$-FBDD for some function

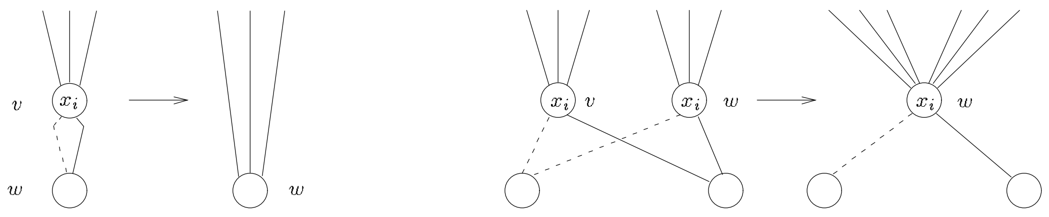

Figure 1: The deletion rule and the merging rule for OBDDs and FBDDs.

$f$. Furthermore, reduced FBDDs respecting the same graph ordering can easily be tested for equivalence by a simple isomorphy test.

If two FBDDs $G_1$ and $G_2$ do not respect the same graph ordering, then no deterministic polynomial time algorithm is known for the equivalence test, i.e., the test whether $G_1$ and $G_2$ represent the same function. A probabilistic polynomial time equivalence test with one-sided error was presented by Blum, Chandra and Wegman [3]. This algorithm always classifies equivalent FBDDs as equivalent but with a probability of at most $1/2$ it may also classify nonequivalent FBDDs as equivalent. Hence, the algorithm shows that the equivalence test for FBDDs is contained in coRP.

We call a node $v$ of an FBDD redundant, if at both successors of $v$ the same function is represented. Then $v$ can be deleted by redirecting all incoming edges to one of the successors of $v$. However, different from OBDDs the deletion rule is not necessarily applicable to redundant nodes during the reduction of FBDDs so that the most efficient known algorithm to detect redundant nodes is to apply the probabilistic equivalence test to the successors of $v$.

For the definitions of notions concerning approximation algorithms we follow Garey and Johnson [8]. Let $\Pi$ be some minimization problem, let $D_\Pi$ be the set of instances of $\Pi$ and let $A$ be some algorithm computing legal solutions of $\Pi$. For $I \in D_\Pi$ let $A(I)$ be the value of the output of $A$ on instance $I$ and let $OPT(I)$ be the value of an optimal solution for $I$. The performance ratio of $A$ is defined as $\sup_{I \in D_\Pi}\{A(I)/OPT(I)\}$. A polynomial time approximation scheme $A$ is a polynomial time algorithm that gets besides $I \in D_\Pi$ an extra input $\varepsilon > 0$. For each $\varepsilon > 0$ it has to achieve a performance ratio of at most $1 + \varepsilon$.

## 3. The Complexity of FBDD Minimization

We prove the nonapproximability results by a reduction from a variant of the satisfiability problem which we call $\varepsilon$ robust 3-SAT-$b$ ($\varepsilon$Rob3SAT-$b$). This problem is a promise problem. We remember that an algorithm for a promise problem has to be successful only on instances fulfilling the promise. In particular, it does not have to check whether the promise is fulfilled and if the promise is not fulfilled, the algorithm may behave arbitrarily.

$\varepsilon$Rob3SAT-$b$

**Instance:** A set $U$ of variables and a set $C$ of clauses fulfilling the following properties:
1. Each clause consists of at least two and at most three literals and each variable occurs in each clause at most once.
2. Each variable occurs at least once and at most $b$ times.
3. Any two clauses share at most one literal.

**Promise:** If the set of clauses is not satisfiable, then for each assignment to the variables at least $\varepsilon|C|$ clauses are not satisfied.

**Problem:** Is there a satisfying assignment to the variables?

The restrictions on the input make the reduction of $\varepsilon$Rob3SAT-$b$ to the problem MinFBDD easier. The promise ensures a gap between satisfiable and nonsatisfiable inputs so that a hardness result for $\varepsilon$Rob3SAT-$b$ also implies a nonapproximability result for $\varepsilon$Rob3SAT-$b$ where the promise is omitted.

We do not know whether a hardness result for $\varepsilon$Rob3SAT-$b$ was explicitly stated in the literature but its hardness follows easily by reexamining the proofs of Arora, Lund, Motwani, Sudan and Szegedy [1] and Papadimitriou and Yannakakis [13]. In the first paper a reduction from the PCP-Theorem implies that there is some $\varepsilon > 0$ so that the problem $\varepsilon$Rob3SAT, i.e. the above promise problem without the restrictions on the input, is NP-hard. In the latter paper the restriction that a variable may occur at most $b$ times is introduced and a reduction from the problem of the former paper is supplied. It is easy to see that the constructed instance also fulfills the other restrictions on the input that are given in the above definition of $\varepsilon$Rob3SAT-$b$. Hence, the following theorem holds.

**Theorem 3:** *There are constants $b \in \mathbb{N}$ and $\varepsilon > 0$ so that $\varepsilon$Rob3SAT-$b$ is NP-hard.*

**Proof of Theorem 1:** We assume that there is a polynomial time approximation scheme $A$ for MinFBDD and want to construct a polynomial time algorithm for $\varepsilon$Rob3SAT-$b$ where $b$ and $\varepsilon$ are the constants ensured by Theorem 3. Let $(U = \{u_1, \dots, u_n\}, C = \{C_1, \dots, C_m\})$ be an instance for $\varepsilon$Rob3SAT-$b$ fulfilling the promise. We are going to present a polynomial time algorithm for the transformation of $(U, C)$ into an instance $H$ for MinFBDD. On $H$ we may apply the polynomial time approximation scheme $A$, and from the size of the result we can decide whether $(U, C)$ is satisfiable. Since on instances $(U, C)$ not fulfilling the promise the algorithm for $\varepsilon$Rob3SAT-$b$ may behave arbitrarily, we do not have to consider this case.

We construct an FBDD for a function $F$ which is defined over the set $V$ of variables where

$$V = \{x_1, \dots, x_n, x'_1, \dots, x'_n\} \cup \{y\} \cup \{z_i^j \mid i \in \{1,2,3\}, j \in \{1, \dots, n+m\}\}.$$

Hence, the number of variables is $5n + 3m + 1$. The function $F$ is composed of the functions $f_1, \dots, f_{n+m}$ which are defined by

$$\begin{aligned} f_i &= x_i \wedge x'_i && \text{for } 1 \le i \le n, \\ f_{n+i} &= \bigwedge_{u_j \in C_i} x_j \wedge \bigwedge_{\overline{u}_j \in C_i} x'_j && \text{for } 1 \le i \le m. \end{aligned}$$

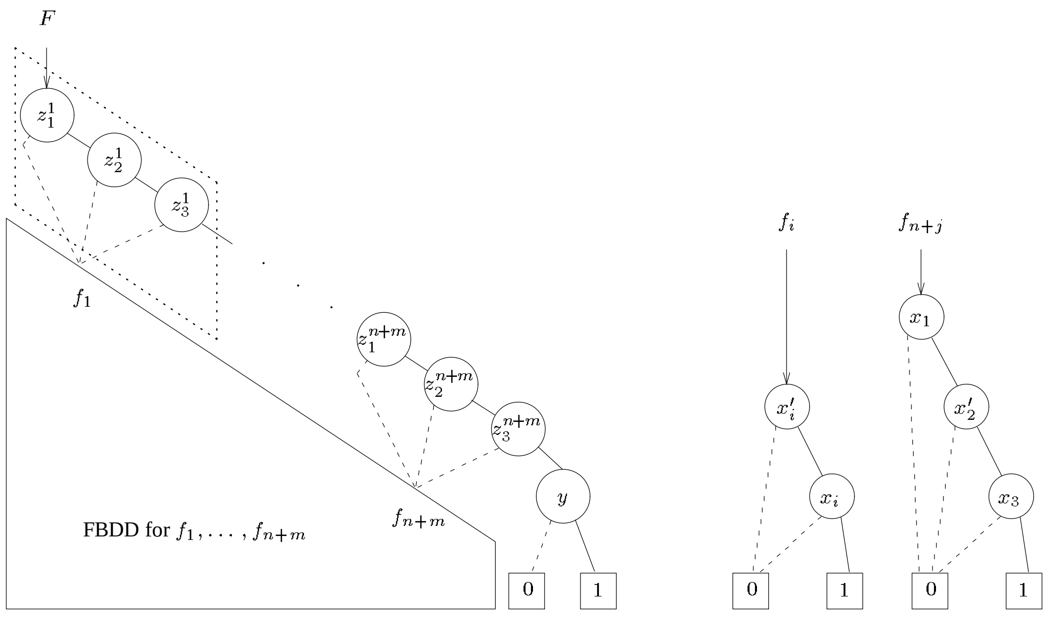

Figure 2: The shape of an FBDD for the function $F$ and FBDDs for $f_i = x_i \wedge x'_i$ and for $f_{n+j} = x_1 \wedge x'_2 \wedge x_3$, i.e., the function corresponding to the clause $C_j = \{u_1, \overline{u_2}, u_3\}$.

Intuitively, the variable $x_i$ corresponds to the variable $u_i$ of the instance $(U, C)$ and $x'_i$ corresponds to the negation of $u_i$. For each variable $u_i$ the function $f_i$ is introduced, and for each clause $C_i$ the function $f_{n+i}$ is introduced where $f_{n+i}$ computes the conjunction of the variables corresponding to the literals in $C_i$.

In the following we use $z^j$ as an abbreviation of $z_1^j \wedge z_2^j \wedge z_3^j$. Then the function $F$ is defined by

$$F = \begin{cases} f_1 & \text{if } z^1 = 0, \\ f_2 & \text{if } z^1 = 1 \text{ and } z^2 = 0, \\ \vdots & \vdots \\ f_i & \text{if } z^1 = \cdots = z^{i-1} = 1 \text{ and } z^i = 0, \\ \vdots & \vdots \\ f_{n+m} & \text{if } z^1 = \cdots = z^{n+m-1} = 1 \text{ and } z^{n+m} = 0, \\ y & \text{if } z^1 = \cdots = z^{n+m} = 1. \end{cases}$$

An FBDD for $F$ is shown in the left side of Figure 2. We see that the FBDD consists of a switch that chooses which of the functions $f_i$ has to be evaluated. It is easy to construct FBDDs for $f_1, \dots, f_{n+m}$ in polynomial time, see the right side of Figure 2. The left side of Figure 2 shows how to combine these FBDDs to an FBDD for $F$ in polynomial time.

Let $L$ denote the total number of literals in the clauses in $C$. Let $\delta = \varepsilon/(18b + 9b^2)$. We apply the polynomial time approximation scheme $A$ for MinFBDD on the constructed FBDD for $F$ for the performance ratio $1 + \delta$. Then we count the number of internal nodes in the computed FBDD $G$ for $F$. We are going to prove the following claim.

**Claim:** *$(U, C)$ is satisfiable iff $G$ consists of at most $(1+\delta)(5n + 2m + L + 1)$ internal nodes.*

We conclude that it suffices to compare the number of internal nodes of $G$ with $(1 + \delta)(5n + 2m + L + 1)$ in order to decide to whether $(U, C)$ is satisfiable. Hence, Theorem 1 follows from the claim and Theorem 3.

In order to prove the claim we first assume that $(U, C)$ is satisfiable. Let $\sigma$ be a satisfying assignment. We show that the size of a minimal FBDD for $F$ is bounded by $5n + 2m + L + 1$ by presenting an FBDD of this size. Since we chose the performance ratio $1 + \delta$ for $A$ it follows that the size of the result of $A$ is bounded by $(1 + \delta)(5n + 2m + L + 1)$.

First we construct an FBDD representing $f_1, \dots, f_n$ with $2n$ internal nodes. Since $f_i = x_i \wedge x_i'$, there are FBDDs for $f_i$ that consist of two internal nodes, one labeled by $x_i$ and the other one labeled by $x_i'$. If $u_i$ has the value $0$ in $\sigma$, we arrange the test of $x_i$ before the test of $x_i'$, and if $u_i$ has the value $1$ in $\sigma$, we arrange $x_i'$ before $x_i$.

For each function $f_{n+i}$ we construct an FBDD consisting of $|C_i|$ internal nodes, where the last variable is a variable corresponding to a literal of $C_i$ that is satisfied by $\sigma$. If we join the constructed FBDDs for $f_1, \dots, f_{n+m}$, the last internal node of $f_{n+i}$ can be merged with a node of one of the FBDDs for $f_1, \dots, f_n$. Hence, the FBDD for all these functions consists of $2n + \sum_{i=1}^{m}(|C_i| - 1) = 2n + L - m$ internal nodes. Finally, we construct from this FBDD for $f_1, \dots, f_{n+m}$ an FBDD for $F$ as outlined in Figure 2. Then the number of nodes labeled by $y$ and the $z$-variables is $3n + 3m + 1$. Hence, the FBDD has at most $5n + 2m + L + 1$ internal nodes. This implies the only-if part of the claim.

In order to prove the other implication of the claim we assume that $(U, C)$ is not satisfiable. By the promise for each assignment to the variables in $U$ at least $\varepsilon m$ clauses are not satisfied. We show that a minimal FBDD for $F$ consists of more than $(1+\delta)(5n+2m+L+1)$ internal nodes. Then also the output of $A$ has to consist of more than this number of nodes, which implies the claim.

We start with some minimal FBDD $G$ for $F$. If this FBDD does not have the shape of the FBDD in Figure 2, i.e., if the $z$-variables are not arranged in the top of the FBDD, we shall rearrange this FBDD without changing the represented function and without increasing the size. Afterwards, the number of nodes labeled by $y$ or a $z$-variable is minimal, since $F$ essentially depends on each of these variables and the FBDD only contains one node testing each of these variables. Then we compute the minimal size of an FBDD representing $f_1, \dots, f_{n+m}$ under the assumption that the instance $(U, C)$ is not satisfiable. Altogether, we obtain a lower bound on the size of an FBDD for $F$.

Before we consider the rearrangement of the FBDD in detail, we point out that we do not present a polynomial time algorithm for the rearrangement. This is not necessary since it suffices to prove a lower bound on the FBDD size. In fact, we always assume that the considered FBDD does not contain redundant nodes. We already remarked in Section 2 that redundant nodes can be deleted by redirecting the incoming edges to one of the successors, but that no deterministic polynomial time algorithm for the detection of redundant nodes is known.

In Figure 2 three $z^1$-nodes are surrounded by a dotted line. We call this arrangement of $z^1$-nodes a $z^1$-block. In the same way we define $z^j$-blocks. In the following we shall make sure that in the FBDD the tests of $z^1$-variables are always arranged as $z^1$-blocks. If this is not the case, we perform the following steps. Let $a_1$, $a_2$ and $a_3$ the numbers of $z_1^1$-, $z_2^1$- and $z_3^1$-nodes, respectively. Without loss of generality let $a_1$ be the minimum of these three numbers. Then we replace in $G$ the variables $z_2^1$ and $z_3^1$ by the constant $1$, i.e., we redirect each pointer leading to such a node to the $1$-successor of this node. The resulting FBDD represents the function $F_{|z_2^1=1,z_3^1=1}$. Afterwards, we replace each $z_1^1$-node $v$ by a $z^1$-block, i.e., we create a $z^1$-block and redirect all edges leading to $v$ to this $z^1$-block. The $0$-edges leaving the nodes of the $z^1$-block are directed to the $0$-successor of $v$ and the $1$-edge leaving the last node of the $z^1$-block is directed to the $1$-successor of $v$. It is easy to verify that we again obtain an FBDD for $F$. Altogether $a_1 + a_2 + a_3$ nodes are removed and $3a_1$ nodes are inserted. Hence, the size does not increase. In the same way we may ensure that the tests of the $z^2$-variables are arranged as $z^2$-blocks and so on. We call the resulting FBDD again $G$.

The next step is to ensure that the $z^i$-blocks are arranged in the top of the FBDD as shown in Figure 2. In order to show that it is possible to rearrange the FBDD in such a way without increasing the size we define the property $P(j)$ of $G$. The FBDD $G$ always has the property $P(0)$. For $j \in \{1, \dots, n+m\}$ the FBDD $G$ has the property $P(j)$ if the $z^1$-block, … ,$z^j$-block in $G$ are arranged as shown in Figure 2, i.e., at the source there is a $z^1$-block, the $1$-successor of the last node of this block is a $z^2$-block and so on up to the $z^j$-block.

**Lemma 4:** *Let $j \in \{1, \dots, n+m\}$. If $G$ has the property $P(j-1)$ and does not have the property $P(j)$ then it contains at least two $z^j$-blocks.*

The technical proof of this lemma is given in Section 5. We rearrange $G$ in the following way. We search for the smallest $j$ so that $G$ does not have the property $P(j)$. Then $G$ has the shape outlined in the left of Figure 3. (In order to simplify Figure 3, we draw the $z^i$-blocks as circles with the label $z^i$.) Since we may assume that $G$ does not contain redundant nodes, the part of $G$ that computes the functions $f_1, \dots, f_{j-1}$ does not contain tests of $z$-variables. Let $v$ be the node which is the $1$-successor of the last node of the $z^{j-1}$-block of $G$ (if $j = 0$ let $v$ be the source of $G$). Let $G^*$ be the FBDD starting at $v$. In $G^*$ we replace the variables $z_1^j$, $z_2^j$ and $z_3^j$ by the constant $1$, i.e., we redirect the pointers leading to a node labeled by one of these variables to the $1$-successor of this node. This does not affect the part of the FBDD computing $f_1, \dots, f_{j-1}$ as remarked above. By Lemma 4 at least 6 internal nodes are deleted by the replacement. Then we create a $z^j$-block and redirect the edge leading to $v$ to this $z^j$-block. The $0$-successor of the $z^j$-block is an FBDD computing $f^j$, which consists of at most 3 internal nodes. The $1$-successor is $G^*$ after the replacement of $z_1^j$, $z_2^j$ and $z_3^j$ by $1$. It is easy to see that the constructed BDD is an FBDD and that it represents $F$. The number of internal nodes does not increase since at most $6$ internal nodes are inserted and at least $6$ internal nodes are deleted by the replacement. This is the reason why for the selection of each of the functions $f_i$ three instead of only one $z$-variable is used. It is easy to see that the FBDD has the properties $P(1), \dots, P(j)$. Hence, we may iterate this procedure until the FBDD has the shape shown in Figure 2. It consists of $3(n+m)+1$ nodes labeled by $y$ and the $z$-variables, which is optimal, and an FBDD of $f_1, \dots, f_{n+m}$. In the following we estimate the size of this representation.

For the representation of $f_1, \dots, f_n$ we need at least $2n$ internal nodes since these functions essentially depend on $2n$ different variables. Since the FBDD does not contain redundant nodes,

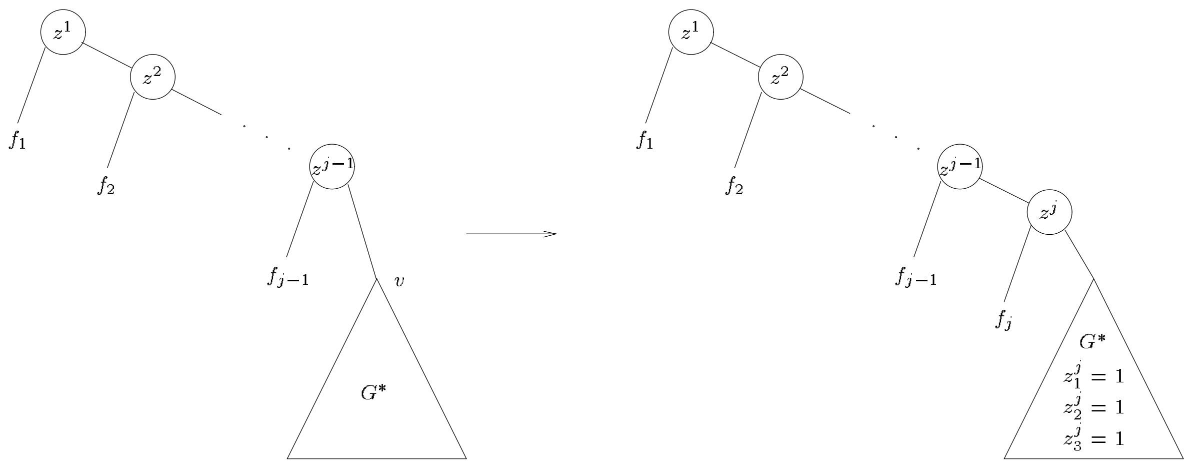

Figure 3: The rearrangement of the FBDD $G$.

we may assume that this representation consists of exactly $2n$ internal nodes. Again we interpret the relative ordering of $x_i$ and $x_i'$ as an assignment to the variable $u_i$. If in the representation of $f_i$ the variable $x_i$ is arranged before $x_i'$ then $u_i = 0$ and otherwise $u_i = 1$.

For the representation of $f_{n+i}$ at least $|C_i|$ internal nodes are necessary if we ignore mergings. Since the FBDD does not contain redundant nodes, we may assume that this representation consists of exactly $|C_i|$ internal nodes. Since each two clauses share at most one literal, at most one node of the representation of $f_{n+i}$, namely the node of the last level can be merged with some other representation. This node may be merged with a node of the representation for $f_1, \ldots, f_n$ or with a node of the representation of $f_{n+j}$. In the former case $C_i$ is satisfied by the assignment defined above. If $C_i$ is not satisfied by this assignment, then a node of the representation of $f_{n+i}$ may be merged with a node of the representation of $f_{n+j}$ but not with a node of the representation of $f_1, \ldots, f_n$. Since two clauses share at most one literal, the representation of $f_{n+i}$ contains at least $|C_i| - 1$ nodes that are not merged with any other node. Since each variable occurs in at most $b$ clauses, at most $b$ representations of functions $f_{n+i}$ that correspond to unsatisfied clauses can share a node. Hence, besides the $\sum_{i=1}^{m}(|C_i| - 1)$ nodes that cannot be merged with other nodes, for each $b$ unsatisfied clauses there is at least one node. By the promise at least $\varepsilon m$ clauses are not satisfied. Hence, the representation of the functions $f_1, \ldots, f_{n+m}$ consists of at least $2n + \sum_{i=1}^{m}(|C_i| - 1) + \varepsilon m/b$ nodes. Together with the $y$- and $z$-nodes, the FBDD contains at least $5n + 2m + L + 1 + \varepsilon m/b$ nodes. By the choice of $\delta$ and because of the inequalities $L \le bn$, $m \ge n/3$, $m \le bn$ and (without loss of generality) $n > 1$, we have $5n + 2m + L + 1 + \varepsilon m/b > (1 + \delta)(5n + 2m + L + 1)$. This completes the proof of the claim and of Theorem 1. $\square$

## 4. The Complexity of Optimizing Graph Orderings

In order to prove Theorem 2 we assume that there is a polynomial time approximation scheme $B$ for MinGraphOrdering. We try to adapt the proof of Theorem 1. This means we construct for the instance $(U, C)$ of $\varepsilon$Rob3SAT-$b$ an FBDD for the function $F$ as described in the last section and apply $B$ on this FBDD. Now we get the problem that the result of $B$ is a graph ordering $H$ rather than an FBDD, and in order to adapt the proof of Theorem 1 we need the size of the $H$-FBDD for $F$. Hence, we would like to solve the following problem in polynomial time.

Reordering of FBDDs

**Instance:** An FBDD $G$ representing some function $F$ and a graph ordering $H$.
**Problem:** Compute the $H$-FBDD for $F$.

We remark that (similar to OBDDs) the output of the reordering problem of FBDDs may be exponential in the input size. Hence, there may only be an algorithm with polynomial run time with respect to the size of the input and the output. However, no such algorithm is known. Meinel and Slobodová [11] present a polynomial time algorithm (with respect to input and output size) for the construction of an OBDD for a given variable ordering and a function given by an FBDD, and they remark that this algorithm can be used for the reordering of FBDDs if there is a polynomial time algorithm for the equivalence test for FBDDs. An algorithm for reordering FBDDs which uses as a subprogram an equivalence test for FBDDs was also presented in Sieling [14]. For this subprogram we use the probabilistic equivalence test for FBDDs of Blum, Chandra and Wegman [3]. Since this algorithm may err, the output of the resulting reordering algorithm may be incorrect, i.e., we obtain a Monte Carlo algorithm. But we would like to point out that this reordering algorithm never computes an FBDD which is larger than the $H$-FBDD for $F$; if it does not err, it computes the $H$-FBDD for $F$, and otherwise it may only compute an $H$-FBDD for some function different from $F$ whose size is not larger than the size of the $H$-FBDD for $F$. This has the following reason. The reordering algorithm for FBDDs uses the equivalence test in order to check subfunctions of $F$ for equivalence where FBDDs for these subfunctions are obtained from $G$ by replacing variables by constants. This is done in order to avoid the creation of nodes of the $H$-FBDD on which any reduction rule is applicable. Because of the one-sided error the probabilistic equivalence test may classify different subfunctions of $F$ as equal. Then (depending on $H$) it may happen that for only one of these subfunctions a node $v$ is created and pointers leading to the node representing the other subfunction falsely lead to $v$. In this case the resulting $H$-FBDD is smaller than the $H$-FBDD for $F$. On the other hand the algorithm cannot fail to detect that two subfunctions coincide.

By iterating the probabilistic equivalence test for FBDDs its error probability can be made exponentially small. If the size of the output of the reordering operation is polynomial, the probabilistic equivalence test is called only a polynomial number of times in the Monte Carlo algorithm described above. Hence, the error probability of the reordering algorithm can be made even exponentially small if the size of the output is bounded by some polynomial. We shall apply the probabilistic reordering algorithm only in this situation.

Hence, we may prove Theorem 2 in the following way. We assume that there is a polynomial time approximation scheme $B$ for MinGraphOrdering. Let $(U, C)$ be an instance for $\varepsilon$Rob3SAT-$b$. We compute an FBDD for the function $F$ as described in the proof of Theorem 1. We apply $B$ for the performance ratio $1 + \delta$ given in that proof and obtain a graph ordering $G$. Then we apply the probabilistic reordering algorithm outlined in the last paragraphs. We decide that $(U, C)$ is satisfiable iff the resulting FBDD consists of at most $(1+\delta)(5n+2m+L+1)$ internal nodes.

In order to analyze this algorithm for $\varepsilon$Rob3SAT-$b$ we distinguish two cases.

**Case 1: $(U, C)$ is satisfiable.** Then by the proof of Theorem 1 the size of a minimum FBDD for $F$ is bounded by $5n+2m+L+1$. The algorithm $B$ computes a graph ordering $H$ so that the size of the $H$-FBDD for $F$ is bounded by $(1+\delta)(5n+2m+L+1)$. Hence, the probabilistic reordering algorithm computes an FBDD with at most $(1+\delta)(5n+2m+L+1)$ internal nodes (possibly for a function different from $F$), and the output of the constructed algorithm for $\varepsilon$Rob3SAT-$b$ is correct.

**Case 2: $(U, C)$ is not satisfiable.** Then by the proof of Theorem 1 the size of a minimum FBDD for $F$ is larger than $(1+\delta)(5n+2m+L+1)$ and $B$ can only obtain a graph ordering $H$ so that the size of the $H$-FBDD for $F$ is larger than $(1+\delta)(5n+2m+L+1)$. (Of course, the size of the $H$-FBDD for $F$ is polynomial since $B$ is an approximation scheme and the size of an optimal FBDD for $F$ is polynomial.) However, with an exponential small probability the reordering algorithm may compute an FBDD for a function different from $F$, and size of this FBDD may be smaller than $(1+\delta)(5n+2m+L+1)$. Hence, with exponentially small probability the constructed algorithm may make the wrong decision that $(U, C)$ is satisfiable.

Altogether, we obtain a probabilistic polynomial time algorithm for $\varepsilon$Rob3SAT-$b$ with an exponentially small one-sided error. This implies $\varepsilon$Rob3SAT-$b$ $\in$ coRP. Since $\varepsilon$Rob3SAT-$b$ is an NP-hard problem, it follows NP $\subseteq$ coRP. Together with RP $\subseteq$ NP and ZPP $=$ RP $\cap$ coRP we conclude NP $=$ ZPP.

## 5. Proof of Lemma 4

In the following it is convenient to consider the $z^i$-blocks as ordinary internal nodes of the FBDD, i.e., we think of $z^i$ as a variable that takes the value $z_1^i \wedge z_2^i \wedge z_3^i$. Then we have to prove that the FBDD contains at least two internal nodes labeled by $z^j$. In $G$ we replace the variables $z^1, \dots, z^{j-1}$ by the constant $1$ and prove the claim for the resulting FBDD $G'$, which computes the function $F' := F_{|z^1=1,\dots,z^{j-1}=1}$. We may obtain $G'$ from $G$ by considering the $1$-successor of the $z^{j-1}$-block of $G$ as the new source and removing all nodes not reachable from the new source. Our first goal is to show that $G'$ contains a $z^j$-node $v$. Afterwards, we show that there is a second $z^j$-node.

Let $X$ be the set of variables tested in $G'$. In order to obtain $v$ we choose an assignment $\xi$ to all variables in $X \setminus \{z^j\}$ and show that the resulting subfunction $F'_{|\xi}$ of $F'$ essentially depends on $z^j$. Hence, if we run through $G'$ on the computation path that is chosen for $\xi$ we find a $z^j$-node, which we call $v$. Since $G$ has the property $P(j)$, the node $v$ is not the first node found on the computation path for $\xi$ in $G'$.

The assignment $\xi$ is defined in the following way: All $x$-variables and all $z$-variables take the value $1$, and $y$ takes the value $0$. Then it follows from the definition of $F$ that $F'_{|\xi,z^j=0} = f_{j|\xi} = 1$ and $F'_{|\xi,z^j=1} = y = 0$, i.e., $F'$ essentially depends on $z^j$.

Hence, there is a $z^j$-node $v$. Let $Q$ be the computation path for $\xi$ from the source of $G'$ to $v$ (excluding $v$). In order to find a second $z^j$-node in $G'$ we perform the following two steps.

- We construct an assignment $\sigma$ to the variables in $X \setminus \{z^j\}$ and show that $F'_{|\sigma}$ essentially depends on $z^j$. This implies that on the computation path for $\sigma$ a $z^j$-block $w$ has to be reached.

- In order to prove that $v \neq w$ we construct from $\xi$ and $\sigma$ the following assignment $\pi$. In $\pi$ the variable $z^j$ remains undefined, all variables tested on $Q$ get the same value as in $\xi$ and all other variables get the same value as in $\sigma$. Hence, $\sigma$ and $\pi$ coincide on the variables tested after $v$. Furthermore, the computation path for $\sigma$ leads to $w$ and the computation path for $\pi$ leads to $v$. If $v$ and $w$ represent the same function, then $F'_{|\sigma,z^j=c} = F'_{|\pi,z^j=c}$ for each $c \in \{0,1\}$. On the other hand, if we show that this equality does not hold for some $c$, we know that $v$ and $w$ represent different functions, i.e., $v$ and $w$ cannot coincide. This proves the claim that there are two $z^j$-nodes in $G'$.

The choice of an appropriate assignment $\sigma$ depends on the set of variables that are tested on $Q$. Hence, we obtain the following case distinction. Let $V_i$ denote the set of variables on which $f_i$ essentially depends.

**Case 1:** On $Q$ a variable $x^* \in V_j$ is tested.
Then we define $\sigma$ as the following assignment: The variable $x^*$ gets the value $0$ and all other $x$-variables, $y$ and all $z$-variables except $z^j$ get the value $1$. Then $f_{j|\sigma} = 0$ and $F'_{|\sigma,z^j=1} = 1$. Therefore, $F'_{|\sigma} = z^j$. Hence, $F'_{|\sigma}$ essentially depends on $z^j$ and on the computation path for $\sigma$ we reach a $z^j$-node $w$. Now we choose from $\sigma$ and $\xi$ the assignment $\pi$ as described above. In $\pi$ the variable $x^*$ and, therefore, all $x$-variables take the value $1$. Then $f_{j|\pi} = 1$ and, therefore, $F'_{|\pi,z^j=0} = 1$ while $F'_{|\sigma,z^j=0} = 0$. Hence, $v \neq w$ and the claim follows for Case 1.

**Case 2:** On $Q$ no variable of $V_j$, but a variable $z^l$ is tested.
Note that on $Q$ all $z$-variables are tested for $1$. We obtain $\sigma$ in the following way. We choose some variable $x^* \in V_j \setminus V_l$. (We remark that $V_j \setminus V_l \neq \emptyset$, since by the requirements on the input of $\varepsilon$Rob3SAT-$b$ each of the sets $V_j$ and $V_l$ contains at least two variables and both sets share at most one variable.) In $\sigma$ the variable $x^*$ gets the value $0$ and all other $x$-variables get the value $1$, the variables $y$ and $z^l$ get the value $0$, and all other $z$-variables except $z^j$ get the value $1$. Then $f_{j|\sigma} = 0$ and we get $F'_{|\sigma,z^j=1}$ by evaluating $f_{l|\sigma}$, which takes the value $1$. Then $F'_{|\sigma} = z^j$. We choose $\pi$ as described above. By $\pi$ all $z$-variables except $z^j$ get the value $1$ and $y$ gets the value $0$. Hence, $F'_{|\pi,z^j=1} = 0$ while $F'_{|\sigma,z^j=1} = 1$. This implies the claim for Case 2.

**Case 3:** On $Q$ neither a variable of $V_j$ nor a $z$-variable are tested, but $y$ is tested on $Q$.
In $\sigma$ the variable $y$ and all $z$-variables except $z^j$ get the value $1$, some variable of $V_j$ gets the value $0$ and all other $x$-variables the value $1$. Then $F'_{|\sigma} = z^j$. If we choose $\pi$ as described above, we have $F'_{|\pi,z^j=1} = 0$ since in $\pi$ all $z$-variable have the value $1$ and $y$ has the value $0$. On the other hand $F'_{|\sigma,z^j=1} = 1$, which implies the claim for Case 3.

**Case 4:** On $Q$ neither a variable of $V_j$, nor a $z$-variable nor $y$ are tested.
Since on $Q$ at least one variable is tested, the only possibility is that on $Q$ some $x$-variable $x^* \notin V_j$ is tested. Let $l$ be a number so that $x^* \in V_l$. We choose $\sigma$ in the following way. The variables $x^*$ and $y$ get the value $0$ and all other $x$-variables the value $1$. The $z$-variables except $z^j$ are chosen in such a way that $f_l$ is selected ($z^i = 1$ for $i \in \{j+1, \ldots, l-1, l+1, \ldots, n+m\}$

and $z^l = 0$). Then $F'_{|\sigma} = \overline{z^j}$. We choose $\pi$ as described above. Since in $\pi$ and $\sigma$ the $z$-variables have the same values, we have $F'_{|\pi,z^j=1} = f_{l|\pi} = 1$ while $F'_{|\sigma,z^j=1} = 0$.
This completes the proof of Lemma 4.

## Acknowledgment

I thank Ingo Wegener for fruitful discussions on the proofs in this paper.

## References

[1] Arora, S., Lund, C., Motwani, R., Sudan, M. and Szegedy, M. (1992). Proof verification and hardness of approximation problems. In *Proc. of 33rd Symposium on Foundations of Computer Science*, 14–23.

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

[3] 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.

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

[5] Bryant, R.E. (1985). Symbolic manipulation of Boolean functions using a graphical representation. In *Proc. of 22nd Design Automation Conference*, 688–694.

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

[7] 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 International Colloquium on Automata, Languages and Programming*, Lecture Notes in Computer Science 62, 227–240.

[8] Garey, M.R. and Johnson, D.S. (1979). Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman.

[9] Gergov, J. and Meinel, C. (1994). Efficient Boolean manipulation with OBDD's can be extended to FBDD's. *IEEE Transactions on Computers* 43, 1197–1209.

[10] Günther, W. and Drechsler, R. (1999). Minimization of free BDDs. To appear in *Proc. of Asia and South Pacific Design Automation Conference*.

[11] Meinel, C. and Slobodová, A. (1994). On the complexity of constructing optimal ordered binary decision diagrams. In *Proc. of International Symposium on Mathematical Foundations of Computer Science*, Lecture Notes in Computer Science 841, 515–524.

[12] Papadimitriou, C.H. (1994). *Computational Complexity*. Addison-Wesley.

[13] Papadimitriou, C.H. and Yannakakis, M. (1991). Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences* 43, 425–440.

[14] Sieling, D. (1995). *Algorithmen und untere Schranken für verallgemeinerte OBDDs.* In German. Ph.D. thesis, Universität Dortmund. Shaker, Aachen.

[15] Sieling, D. (1998). On the existence of polynomial time approximation schemes for OBDD-Minimization (Extended Abstract). In *Proc. of 15th Symposium on Theoretical Aspects of Computer Science*, Lecture Notes in Computer Science 1373, 205–215.

[16] Sieling, D. (1998). The nonapproximability of OBDD minimization. ECCC Report TR98-001 (available from www.eccc.uni-trier.de).

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

[18] Simon, J. and Szegedy, M. (1993). A new lower bound theorem for read-only-once branching programs and its applications. In *Advances in Computational Complexity Theory*, Jin-Yi Cai, ed., DIMACS Series in Discrete Mathematics and Theoretical Computer Science 13, American Mathematical Society, 183–193.

[19] Tani, S., Hamaguchi, K. and Yajima, S. (1993). The complexity of the optimal variable ordering problems of shared binary decision diagrams. In *Proc. of 4th International Symposium on Algorithms and Computation*, Lecture Notes in Computer Science 762, 389–398.

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

[21] Žák, S. (1984). An exponential lower bound for one-time-only branching programs. In *Proc. of International Symposium on Mathematical Foundations of Computer Science,* Lecture Notes in Computer Science 176, 562–566.