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

The Complexity of Minimizing and Learning FBDDs*

Detlef Sieling
FB Informatik, LS 2, Univ. Dortmund,
44221 Dortmund, Fed. Rep. of Germany
sieling@Ls2.cs.uni-dortmund.de

Abstract

Free Binary Decision Diagrams (FBDDs) or read-once branching programs are a data structure for Boolean functions. They can efficiently be manipulated if only FBDDs respecting a fixed graph ordering are considered. However, the size of such FBDDs may strongly depend on the chosen graph ordering. In this paper it is shown that the existence of polynomial time approximation schemes for optimizing graph orderings or for minimizing FBDDs implies NP = P, and so such algorithms are quite unlikely to exist. The same holds for the related problem of computing minimal size FBDDs that are consistent with a given set of examples. The latter result implies that size bounded FBDDs are not PAC-learnable unless NP = RP.

Keywords: Free binary decision diagram, read-once branching program, approximation scheme, nonapproximability, PAC-learning

1. Introduction

Many variants of Binary Decision Diagrams (BDDs) have been investigated 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]. Many generalizations of OBDDs have been considered since there are many 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).

*This work was supported by DFG grant We 1066/8. A preliminary version has been accepted for MFCS'99.

ISSN 1433-8092

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 [16] most previous approaches are handled in a unified way. Already in the early paper of Fortune, Hopcroft and Schmidt [6] 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 [15] and Gergov and Meinel [9]. It turned out that many but not all operations on Boolean functions which can efficiently be performed on functions represented by OBDDs can also efficiently be performed 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 efficiently be performed 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 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 [15]. However, the question whether there are efficient algorithms for computing good or optimal graph orderings remains open. This motivates to explore the complexity of the following two optimization problems.

Minimum FBDD (MinFBDD)

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

Optimization of Graph Orderings (OptGraphOrdering)

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.

A related problem is the computation of a minimum size FBDD where the function to be represented is specified by a set of examples.

Minimum Consistent FBDD (MCFBDD)

Instance: A set E of examples, i.e. pairs $\langle x, \beta \rangle$, where $x \in \{0,1\}^N$ and $\beta \in \{0,1\}$.
Problem: Compute an FBDD of minimal size that represents a function that is consistent with all examples in E, i.e., for which $\forall \langle x, \beta \rangle \in E \colon f(x) = \beta$.

This problem occurs when searching for small size representations for incompletely specified Boolean functions that are given by a set of inputs and the corresponding function values. It is

easy to compute an FBDD of size $|E| \cdot N$ that is consistent with the set E of examples. This FBDD contains for each x, where $\langle x, \beta \rangle \in E$, a computation path that is chosen only for the input x and ends at the β-sink. However, such an FBDD realizes some kind of "table look-up" and it is a natural goal to find a representation of the set of examples that is smaller than a table containing all examples in order to save memory. Learning theory provides another motivation to search for smaller representations of sets of examples. We focus on the PAC-learning model (PAC=probably approximately correct), which was introduced by Valiant [19]. Roughly, the task is to find for a set of randomly chosen examples of a hidden function (the so-called concept) a hypothesis that approximates the concept as well as possible. An FBDD G that is consistent with E and considerable smaller than $|E| \cdot N$ "generalizes" the given set of examples, i.e., the function represented by G is likely to coincide on more inputs with the hidden concept than the function realized by a table look-up. We remark that the possibility of efficiently learning functions represented by size bounded FBDDs, i.e. of the efficient construction of size bounded FBDDs well approximating a hidden concept, is determined by the complexity of MCFBDD. This follows from the results of Pitt and Valiant [12] who showed that the NP-hardness of the construction of a hypothesis consistent with a given set of examples implies the nonlearnability of the corresponding class of hypotheses under the assumption NP $\neq$ RP.

We shall prove the following hardness results for the three optimization 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 OptGraphOrdering, then NP $=$ P.*

Theorem 3: *If there is a polynomial time approximation scheme for MCFBDD, then NP $=$ P.*

Hence, it is unlikely that the considered problems have polynomial time approximation schemes and we get the justification to give up the search for polynomial time approximation schemes. By the results of Pitt and Valiant [12] Theorem 3 also implies that for some function $s : \mathbb{N} \to \mathbb{N}$, FBDDs with $s(n)$ nodes are not learnable in the PAC-learning model unless NP $=$ RP.

For OBDDs there are similar optimization problems. In the following we give an overview over the most important results concerning the corresponding optimization problems for OBDDs and compare these results with our results for FBDDs.

The problem corresponding to MinFBDD is the problem MinOBDD, i.e. the problem to compute a minimal size OBDD for a function given by an OBDD. Similarly one may define the analog of the problem OptGraphOrdering: OptVarOrdering is the problem to compute an optimal variable ordering for a function given by an OBDD. However, MinOBDD and OptVarOrdering are polynomially related and, therefore, usually not explicitly distinguished. On the other hand it is not clear whether MinFBDD and OptGraphOrdering are polynomially related, since it is not known whether a polynomial time algorithm for OptGraphOrdering implies a polynomial time algorithm for MinFBDD. The NP-hardness of MinOBDD and OptVarOrdering for OBDDs for multi-output functions was shown by Tani, Hamaguchi and Yajima [18] and for OBDDs for single-output functions by Bollig and Wegener [4]. In Sieling [13] it is shown that the existence of polynomial time approximation schemes for these problems implies P $=$ NP and in Sieling

[14] that it is NP-hard to approximate these problems up to any constant factor. For FBDDs it remains an open problem whether MinFBDD and OptGraphOrdering can be approximated up to some constant factor or whether a similar nonapproximability result can be proved.

The learnability of size bounded OBDDs under the PAC-learning model was considered by Takenaga and Yajima [17]. They prove the NP-completeness of the problem Minimum OBDD Identification, i.e., the problem to find for a set of examples and for a number B an OBDD with the *fixed* variable ordering $x_1, \ldots, x_N$ and size at most B that is consistent with all examples. This result implies that size bounded OBDDs with a fixed variable ordering are not PAC-learnable unless NP $=$ RP. A difference to our result is that the variable ordering is fixed, while for our result we do not assume anything about the graph ordering.

The paper is organized as follows. In the following section we repeat some definitions and results concerning FBDDs and approximation algorithms. In Section 3 the main proof idea is presented and Theorem 1 is proved. Then we discuss the adaptation of this proof to the problems OptGraphOrdering and MCFBDD (Sections 4 and 5). Finally, we prove two technical lemmas in Section 6 and conclude the paper with some open problems.

2. Preliminaries

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. In OBDDs we have the extra condition that on all paths the variables are tested according to a fixed ordering. Examples of FBDDs are shown on the right side of Figure 1. In the figure edges are 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 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 also be applied to FBDDs: 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 [15] prove that reduced G-FBDDs are unique up to isomorphism. Hence, we may talk about *the* (reduced) G-FBDD for some function f.

If two FBDDs G_1 and G_2 do not respect the same graph ordering, 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 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 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 Π be some minimization problem, let D_Π be the set of instances of Π and let A be some algorithm computing legal solutions of Π. For $I \in D_\Pi$ let $A(I)$ be the value of the output of A on instance I and let $\mathit{OPT}(I)$ be the value of an optimal solution for I. The performance ratio of A is defined as $\sup_{I \in D_\Pi}\{A(I)/\mathit{OPT}(I)\}$. 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 ε robust 3-SAT-b (εRob3SAT-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. It does not have to detect that the promise is not fulfilled and in this case it may behave arbitrarily.

εRob3SAT-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, for each assignment to the variables at least $\varepsilon|C|$ clauses are not satisfied.

Problem: Is there a satisfying assignment to the variables?

The promise ensures a gap between satisfiable and nonsatisfiable inputs, so that a hardness result for εRob3SAT-b also implies a nonapproximability result for εRob3SAT-b where the promise is omitted and the goal is to maximize the number of satisfied clauses. The restrictions on the input make the reduction of εRob3SAT-b to MinFBDD easier. We do not know whether a hardness result for εRob3SAT-b was explicitly stated in the literature. The following hardness result follows easily by reexamining the proofs of Arora, Lund, Motwani, Sudan and Szegedy [1] and Papadimitriou and Yannakakis [11]. In the first paper a reduction from the PCP-Theorem implies that there is some $\varepsilon > 0$ so that the problem εRob3SAT, 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 εRob3SAT-b. Hence, the following theorem holds.

Theorem 4: *There are constants $b \in \mathbb{N}$ and $\varepsilon > 0$ so that εRob3SAT-b is NP-hard.*

Proof of Theorem 1: We assume that there is a polynomial time approximation scheme A for MinFBDD and construct a polynomial time algorithm for εRob3SAT-b where b and ε are the constants ensured by Theorem 4. Let $(U = \{u_1, \ldots, u_n\}, C = \{C_1, \ldots, C_m\})$ be an instance for εRob3SAT-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. To H we apply the approximation scheme A, and from the size of the result we can decide whether (U, C) is satisfiable. Since on instances not fulfilling the promise algorithms for εRob3SAT-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

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

of variables. The function F is composed of the functions $f_1, \ldots, f_{n+m}$ defined by

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

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 $z_1^j \wedge z_2^j \wedge z_3^j$.

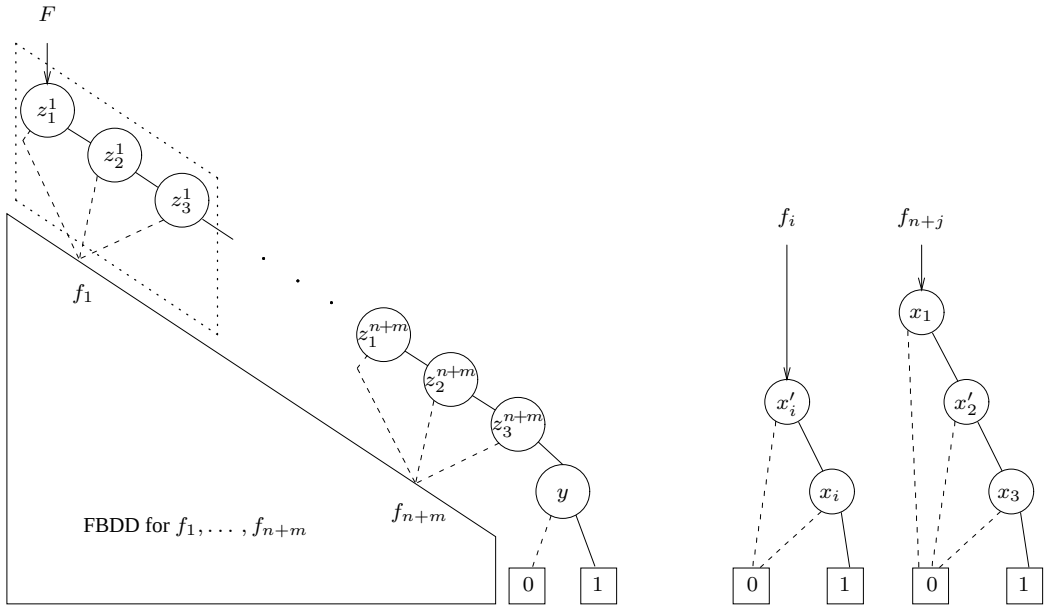

Figure 1: The shape of an FBDD for the function F and FBDDs for $f_i = x_i \wedge x_i'$ and for $f_{n+j} = x_1 \wedge x_2' \wedge x_3$ (the function corresponding to the clause $C_j = \{u_1, \overline{u_2}, u_3\}$)

Then we define

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

The shape of an FBDD for F is shown in the left side of Figure 1. We see that the FBDD consists of a switch that chooses which of the functions f_i has to be evaluated. Figure 1 shows that it is easy to construct FBDDs for $f_1, \ldots, f_{n+m}$ in polynomial time and 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 $\delta = \varepsilon/(21b)$. We apply the polynomial time approximation scheme A for MinFBDD to the constructed FBDD for F for the performance ratio $1 + \delta$ and obtain an FBDD G. By the following claim it suffices to compare the number of internal nodes of G with $(1 + \delta)(5n + 2m + L + 1)$ in order to decide whether (U, C) is satisfiable. Hence, Theorem 1 follows from the claim and Theorem 4.

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

In order to prove the claim we first assume that (U, C) is satisfiable. Let σ 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 $(1+\delta)(5n+2m+L+1)$.

It is easy to construct an FBDD for $f_1, \ldots, f_n$ with $2n$ internal nodes in such a way that in the representation of f_i the variable x_i is arranged before x'_i if u_i has the value 0 in σ, and x'_i is arranged before x_i otherwise. 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 σ. If we join the constructed FBDDs for $f_1, \ldots, f_{n+m}$, the last internal node of the FBDD for f_{n+i} can be merged with a node of one of the FBDDs for $f_1, \ldots, f_n$. 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 $f_1, \ldots, f_{n+m}$ an FBDD for F as outlined in Figure 1. Then the number of nodes labeled by y and the z-variables is $3n+3m+1$ and the total number of internal nodes is $5n + 2m + L + 1$. 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 εm 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 an arbitrary FBDD G for F. If this FBDD does not have the shape of the FBDD in Figure 1, 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 so that afterwards the z-variables are arranged as shown in Figure 1. Then 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. Finally, we compute the minimal size of an FBDD representing $f_1, \ldots, f_{n+m}$ 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.

Since our goal is to prove a lower bound on the FBDD size, it is not necessary to present a polynomial time algorithm for the rearrangement. 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 1 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. The first step of the rearrangement is to make sure that in the FBDD the tests of z^1-variables are always arranged as z^1-blocks. Let a_1, a_2 and a_3 be the numbers of z^1_1-, z^1_2- and z^1_3-nodes, respectively. W.l.o.g. let a_1 be the minimum of these three numbers. Then we replace in G the variables z^1_2 and z^1_3 by the constant 1, i.e., we redirect each edge leading to a node labeled by z^1_2 or z^1_3 to the 1-successor of this node. The resulting FBDD represents the function $F_{|z^1_2=1, z^1_3=1}$. 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 and that 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 1. 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. For $j \in \{1, \ldots, n+m\}$ the FBDD G has the Property $P(j)$ if the z^1-block, ... , z^j-block in G are arranged as shown in Figure 1, 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. In order to simplify the notation we say that G always has the (empty) Property $P(0)$.

Lemma 5: *Let G be an FBDD for F without redundant nodes and let $j \in \{1, \ldots, n+m\}$. If G has the Property $P(j-1)$ and does not have the Property $P(j)$, it contains at least two z^j-blocks.*

We postpone the proof of this lemma to Section 6 where we prove a more general result. Now we can rearrange G in the following way. We search for the smallest j so that G does not have the Property $P(j)$. Then the z^1-, ... , z^{j-1}-blocks are arranged as shown on the left side of Figure 2. In order to simplify the figure z^j-blocks are drawn as ordinary internal nodes of an FBDD. The SubFBDDs indicated by $G_1, \ldots, G_{j-1}$ are representations of $f_1, \ldots, f_{j-1}$, respectively, which may share internal nodes. Since we may assume that G does not contain redundant nodes, $G_1, \ldots, G_{j-1}$ do 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 edges leading to a node labeled by one of these variables to the 1-successor of this node. By Lemma 5 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. As 0-successor of the z^j-block we create 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. The resulting BDD is shown in the right of Figure 2. It is easy to see that the constructed BDD is an FBDD for 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 $P(1), \ldots, P(j)$. Hence, we may iterate this procedure until the FBDD has the shape shown in Figure 1. It consists of $3(n+m)+1$ nodes labeled by y and the z-variables, which is optimal, and an FBDD for $f_1, \ldots, f_{n+m}$. In the following we estimate the size of this FBDD.

Since the FBDD does not contain redundant nodes, the representation of $f_1, \ldots, f_n$ consists of exactly $2n$ internal nodes. We interpret the relative ordering of x_i and x_i' as an assignment to the variable u_i where $u_i = 0$ iff in the representation of f_i the variable x_i is arranged before x_i'. For the representation of f_{n+i} there are $|C_i|$ internal nodes if we ignore mergings, because the FBDD does not contain redundant 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 node, namely 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

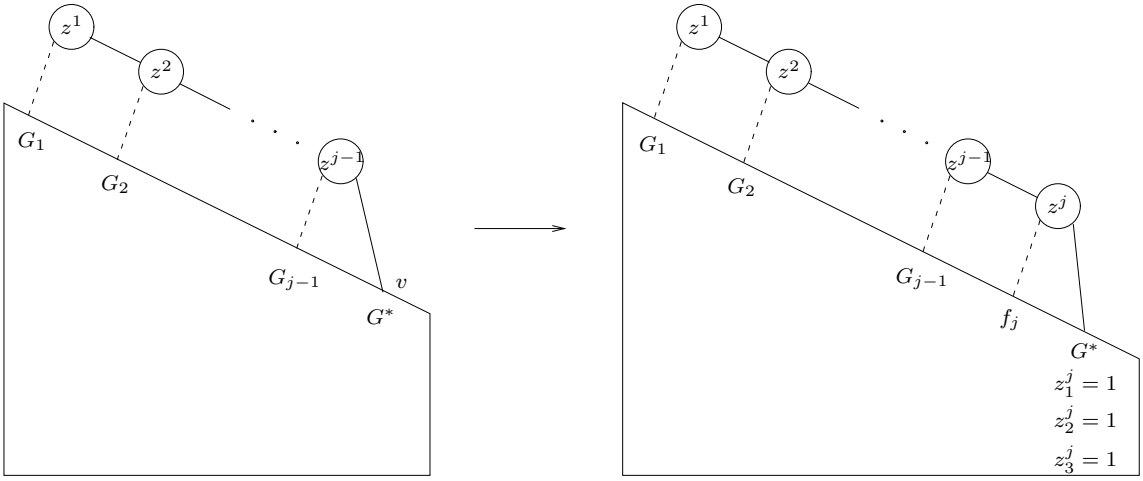

Figure 2: The construction of an FBDD with the Property $P(j)$ from an FBDD with the Property $P(j-1)$

above. If C_i is not satisfied by this assignment, 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 εm 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 $3(n+m)+1$ nodes labeled by y and the z-variables, the FBDD contains at least $5n + 2m + L + 1 + \varepsilon m/b$ nodes. By the choice of δ and because of the inequalities $L \leq 3m$, $n \leq 3m$ and (w.l.o.g.) $m > 1$, 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. □

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 OptGraphOrdering. We try to adapt the proof of Theorem 1. This means we construct for the instance (U, C) of εRob3SAT-b an FBDD for the function F as described in the last section and apply B to this FBDD. Now we get the problem that the result of B is a graph ordering H instead of an FBDD, while by the results of Section 3 we only know of a relation between the size of an H-FBDD for F and the satisfiability of (U, C). Hence, we would like to compute an H-FBDD for F in order to determine its size. However, for the computation of an H-FBDD from a graph ordering H and a function given by an FBDD no polynomial time algorithm is known. Hence, we use a different approach which only works in our special situation.

Assume for a moment that it is possible to construct an H-FBDD G' for the function F. Then we may apply transformation steps similar to those described in the proof of Theorem 1 in order to obtain an equivalent FBDD G with the shape shown in Figure 1. By these steps the size of the considered FBDD does not increase. Hence, we may count the number of nodes of G in order to decide whether (U, C) is satisfiable. In the following we show that we can compute G from H in polynomial time without computing G'. Hence, it is possible to determine in polynomial time whether (U, C) is satisfiable. This implies Theorem 2.

Now let a graph ordering H be given. In order to construct G it suffices to determine for each function f_i the relative ordering of the variables that f_i depends on. In the following we show how to determine the relative ordering of x_1 and x'_1 in the representation of f_1. During the rearrangement of the hypothetical H-FBDD G' the following steps are performed without increasing the size:

1. It is made sure that all tests of z^1-variables are arranged as z^1-blocks.

2. If afterwards at the source there is no z^1-block, the FBDD contains at least two z^1-blocks. Hence, it is possible to rearrange the FBDD by creating a new z^1-block as the source, whose 0-successor is an arbitrary FBDD for f_1 and whose 1-successor is the previous FBDD after replacing the z^1-variables by 1.

3. At the 0-successor of the z^1-block the function f_1 is computed. Since f_1 only depends on x_1 and x'_1, all nodes labeled by other variables are redundant and can be removed. From the resulting FBDD for f_1 we can determine the relative ordering of x_1 and x'_1.

It is crucial to observe that the relative ordering of x_1 and x'_1 is not determined by H and can be chosen arbitrarily, if at the source of H a variable that is not a z^1-variable is tested, or if in the third step there are computation paths with both relative orderings of x_1 and x'_1. Hence, we can determine the relative ordering of x_1 and x'_1 directly from H: If at the source of H there is not a z^1-variable, then we may choose the ordering arbitrarily. Otherwise we consider the graph ordering H^* that is obtained from H by replacing the z^1-variable z^1_l at the source by 0 and the other z^1-variables by 1. This is the same replacement as in the rearrangement of the z^1-nodes to z^1-blocks, since there is at most one node labeled by z^1_l, i.e. $a_l = 1$ and, hence, it is the minimum of a_1, a_2 and a_3. By a simple depth first search approach it can be determined whether there is a computation path in H^* on which x_1 is arranged before x'_1, whether there is computation path on which x'_1 is arranged before x_1, or whether both types of computation paths occur. In the latter case the relative ordering of x_1 and x'_1 can be chosen arbitrarily, and in the former cases the relative ordering is the same as in H^*.

Afterwards, we may replace the z^1-variables by the constant 1 and may proceed by determining the relative ordering of x_2 and x'_2, and so on. In the same way we may determine the relative orderings of the variables of the functions corresponding to the clauses. After computing all these orderings we can construct an FBDD for F as shown in Figure 1, apply the reduction rules to this FBDD and count the number of its internal nodes. In the same way as in the last section we decide whether (U, C) is satisfiable. We remark that in the case that (U, C) is satisfiable this algorithm does not necessarily compute a satisfying assignment since it is sufficient that the

algorithm obtains a graph ordering so that the size of the corresponding FBDD is bounded by $(1+\delta)(5n+2m+L+1)$. Such a graph ordering does not necessarily correspond to a satisfying assignment but merely to an assignment for which less than $\varepsilon|C|$ clauses are not satisfied.

5. The Complexity of Minimum Consistent FBDD

In order to prove Theorem 3 we again provide a reduction from εRob3SAT-b. Assume that there is a polynomial time approximation scheme A for MCFBDD. Let (U, C) be an instance for εRob3SAT-b that fulfills the promise. The proof works in the following way. We first show how to construct a set of examples from (U, C). To this set of examples we apply A and from the size of the resulting FBDD G' we can decide whether (U, C) is satisfiable. We shall use several ideas of the proof of Theorem 1: The examples are defined over the set V of variables given in the proof of Theorem 1. All examples are consistent with the function F. We shall show that we can modify G' without increasing its size so that it afterwards represents the function F. Hence, we can use the analysis of the FBDD size for F given in Section 3.

The examples constructed for (U, C) are listed in Tables I and II. For each $j \in \{1, \ldots, n+m\}$ there is a copy of Table I. Let V_j denote the set of variables that f_j essentially depends on. Hence $|V_j| \in \{2, 3\}$. In Table I the term $z^j = 1$ means that the variables z_1^j, z_2^j and z_3^j are set to 1, while $z^j = 0$ is an abbreviation for the seven assignments to z_1^j, z_2^j and z_3^j, where at least one variable is equal to 0. Similarly, the entry in row 5a means all assignments to x-variables, where at least one $x \in V_j$ is set to 0 (and the remaining variables $x \in V_j$ to 1) and at most two variables $x \notin V_j$ are set to 1 (and the remaining variables $x \notin V_j$ to 0). Obviously there are $O(n^2)$ such assignments. For each row the number of corresponding examples is given in the last column. Since there are $n + m$ copies of the first table, there are altogether $O((n + m)^3)$ examples. It is easy to see that this set of examples can be constructed in polynomial time. Obviously, all examples are consistent with the function F.

We choose δ as in the proof of Theorem 1 and apply the polynomial time approximation scheme A for δ to the constructed set of examples. The result is an FBDD G' that is consistent with all examples. G' represents some function F', which may be different from F. We shall prove that we can compute an FBDD G from G' so that G represents F and $|G| \leq |G'|$. Hence, the analysis of the FBDD size of F in the proof of Theorem 1 implies the following.

1. If (U, C) is satisfiable, there is an FBDD for F with at most $5n + 2m + L + 1$ nodes. Since all examples are consistent with F, there is a solution of the constructed instance of MCFBDD of this size. Hence, A computes a solution G' with at most $(1 + \delta)(5n + 2m + L + 1)$ nodes.

2. If (U, C) is not satisfiable, each FBDD for F has more than $(1 + \delta)(5n + 2m + L + 1)$ nodes. Since, as shown in the following, from each solution G' we can construct an FBDD G for F with $|G| \leq |G'|$, it follows $|G'| > (1 + \delta)(5n + 2m + L + 1)$.

Hence, (U, C) is satisfiable iff the result G' of A has at most $(1 + \delta)(5n + 2m + L + 1)$ nodes, and we obtain a polynomial time algorithm for εRob3SAT-b in contradiction to Theorem 4.

Table I:

	x-variables	y	z-variables except z^j	z^j	function value	number of examples
1a	1	0	1	0	1	7
1b	1	0	1	1	0	1
1c	1	1	1	0	1	7
	For all $x^* \in V_j$:					
2a	1 except $x^* = 0$	1	1	0	0	$7 \cdot 3$
2b	1 except $x^* = 0$	1	1	1	1	3
2c	1 except $x^* = 0$	0	1	1	0	3
	For all $l > j$ and for *some* $x^* \in V_j \setminus V_l$					
3a	1 except $x^* = 0$	0	1 except $z^l = 0$	0	0	$7 \cdot 7 \cdot (n+m)$
3b	1 except $x^* = 0$	0	1 except $z^l = 0$	1	1	$7 \cdot (n+m)$
	For all x-variables $x^* \in (V_{j+1} \cup \cdots \cup V_{n+m}) \setminus V_j$ and the maximal l where $x^* \in V_l$					
4a	1 except $x^* = 0$	0	1 except $z^l = 0$	0	1	$7 \cdot 7 \cdot 2 \cdot n$
4b	1 except $x^* = 0$	0	1 except $z^l = 0$	1	0	$7 \cdot 2 \cdot n$
4c	1	0	1 except $z^l = 0$	1	1	$7 \cdot 2 \cdot n$
5a	0 for some $x \in V_j$ and 1 for at most two $x \notin V_j$	0	1	0	0	$O(n^2)$
5b	1 for all $x \in V_j$ and 1 for at most two $x \notin V_j$	0	1	0	1	$O(n^2)$

Table II:

	x-variables	y	z-variables	function value	number of examples
6a	0	0	1	0	1
6b	0	1	1	1	1

In the following we show how to construct G. The FBDD G' is transformed into G in the following three steps. We note that all intermediate results are FBDDs that are not larger than G' and that represent functions that are consistent with all examples.

Step 1: Reordering of z_1^1-, z_2^1- and z_3^1-nodes to z^1-blocks.

As in the proof of Theorem 1 let a_1, a_2 and a_3 be the numbers of z_1^1-, z_2^1- and z_3^1-nodes. W.l.o.g. let a_1 be the minimum of those numbers. We replace z_2^1 and z_3^1 by the constant 1 and, afterwards, we replace each z_1^1-node by a z^1-block. By the same arguments as in the proof of Theorem 1 the size of the FBDD does not increase, but now the represented function may change, since a function F' different from F may be represented. For example, $F'_{|z_1^1=0,z_2^1=1,z_3^1=1}$ and $F'_{|z_1^1=0,z_2^1=1,z_3^1=0}$ may be different, while after the reordering of the z^1-nodes to z^1-blocks the corresponding subfunctions coincide. However, this does not matter, since the resulting function is consistent with all examples. This follows from the fact that for all assignments that differ only in the z^1-variables and where at least one of the z^1-variables takes the value 0 the examples prescribe the same value of the function.

In the same way we reorder the z^2-nodes to z^2-blocks and so on. We call the resulting FBDD again G'. Since the z-variables are arranged as blocks, we consider the z-blocks as ordinary variables in the following. E.g., we may talk about replacing z^j by 0, which means that the z^j-variables are replaced by constants, where at least one of those variables gets the value 0.

Step 2: Reordering of the z-nodes to a "switch" as shown in the left of Figure 1.

As in the proof of Theorem 1 we say that G' has the Property $P(j)$ iff the z^1-, ... , z^j-blocks are arranged as shown in Figure 1. Let j be the number for which G' has the Property $P(j-1)$ but not the Property $P(j)$. We show how to construct from G' an FBDD that is consistent with all examples and has the Property $P(j)$. Hence, we may iteratively apply this construction in order to obtain an FBDD with the Property $P(n+m)$.

We perform in G' the replacement $z^1 = 1, \ldots, z^{j-1} = 1$ and obtain the SubFBDD G^* of G' whose source is the 1-successor of the z^{j-1}-block in the switch in G'. Hence, we may only use examples, where $z^1 = \cdots = z^{j-1} = 1$. Since G' does not have the Property $P(j)$, at the source of G^* there is not a z^j-block.

If at the source of G^* an x-variable $x^* \notin V_j \cup \cdots \cup V_{n+m}$ is tested, we may replace x^* by an arbitrary constant. The represented function may change, but it remains consistent with all examples since there are no two examples prescribing different function values for $z^1 = \cdots = z^{j-1} = 1, x^* = 0$ and $z^1 = \cdots = z^{j-1} = 1, x^* = 1$ (and identical assignments to the remaining variables). This is iterated until at the source of the SubFBDD there is an x-variable $x^* \in V_j \cup \cdots \cup V_{n+m}$ or y or a z-variable. We call the resulting SubFBDD again G^*. If now at the source of G^* there is a z^j-block, nothing more is to show. Otherwise, we prove that there are at least two z^j-blocks in G^*.

Lemma 6: *Let G' be an FBDD that is consistent with all examples and has the Property $P(j-1)$, but not the Property $P(j)$. Furthermore, let at the 1-successor v of the z^{j-1}-block of the switch some x-variable in $V_j \cup \cdots \cup V_{n+m}$ or y or a z-variable be tested. Then from v at least two z^j-blocks are reachable.*

We prove Lemma 6 in Section 6. Now we modify G' in a similar way as in Section 3 and shown in Fig. 2: In G' we replace z^j by the constant 1. (Different from the proof in Section 3 there may be z^j-blocks in the FBDD reached by the 0-edges leaving the z^1-, ... , z^{j-1}-block of the switch. Furthermore, this part of the FBDD may share such blocks with G^*. Hence, also the z^1-, ... , z^{j-1}-blocks in that part of the FBDD are replaced by 1.) Afterwards, we create a z^j-block, whose 0-successor is an FBDD for f_j (consisting of at most three nodes) and whose 1-successor is G^* after the replacement. In G' we redirect the 1-edge leaving the z^{j-1}-block of the switch to the new z^j-block. We call the resulting FBDD again G'. Obviously, the size does not increase by these modifications since at least six nodes are deleted by the replacement and most six nodes are inserted. It remains to show that the represented function is still consistent with all examples.

For inputs with $z^1 = \cdots = z^j = 1$ the function does not change. The replacement $z^1 = \cdots = z^{j-1} = 1$ and $z^j = 0$ now yields the subfunction f_j, which is consistent with all examples. For all other inputs, i.e. inputs where for some $i < j$ the block z^i takes the value 0, the represented function may change since z^j was replaced by 1. However, the function is still consistent with all examples since there are no examples prescribing a value of the function essentially depending on z^j if for some $i < j$ the block z^i takes the value 0.

Step 3: Modification of the FBDD so that at the edges leaving the switch the functions $f_1, \ldots, f_{n+m}$ and y are computed.

Let G'_i be SubFBDD at the 0-edge leaving the z^i-block of the switch. The function represented by G'_i may be different from f_i, for example, it may depend on y, on z^j for some $j > i$, or on $x \notin V_i$. W.l.o.g. let $V_i = \{x_1, x_2, x_3\}$. We run through G'_i on the computation path for the assignment where all variables in V_i and all z^j-variables, where $j \neq i$, take the value 1 and all other variables take the value 0. Because of example 5b the value 1 is computed. If we change that assignment by replacing one of the V_i-variables by 0, because of example 5a the value 0 is computed. Hence, on the considered computation path all variables in V_i are tested. We construct an FBDD for $f_i = x_1 \wedge x_2 \wedge x_3$ that consists of three internal nodes and tests the variables in the same ordering as on the considered computation path. We replace the edge to G'_i by an edge to that FBDD. This is done for all $i \in \{1, \ldots, n+m\}$. Similarly we replace the 1-edge leaving the z^{n+m}-block of the switch by an edge to an FBDD for the function y that consists of only one y-node. Afterwards, the reduction rules are applied. We call the resulting FBDD G. Obviously, G represents the function F. It remains to show that $|G| \leq |G'|$.

From the consideration of the computation paths in the last paragraph it followed that G'_i contains at least one x-node for each $x \in V_i$. Hence, G can only be larger than G', if for some $x \in V_i \cap V_j$ the SubFBDDs G'_i and G'_j share an x-node, while the corresponding SubFBDDs in G have different x-nodes. W.l.o.g. let $V_i = \{x_1, x_2, x_3\}$ and $V_j = \{x_1, x_4, x_5\}$. Let G''_i be the FBDD that we obtain from G'_i by replacing all x-variables except $x_1, \ldots, x_5$ by 0, the variable y and the z^i-block by 0, and all z^l, where $l \neq i$, by 1. Similarly, let G''_j be the FBDD that we obtain from G'_j by replacing all x-variables except $x_1, \ldots, x_5$ by 0, the variable y and the z^j-block by 0, and all z^l, where $l \neq j$, by 1. Because of the examples 5a and 5b the FBDDs G''_i and G''_j compute the functions $f_i = x_1 \wedge x_2 \wedge x_3$ and $f_j = x_1 \wedge x_4 \wedge x_5$. The FBDDs G''_i and G''_j can only share an x_1-node if in G''_i the variables x_2 and x_3 are tested before x_1 and in G''_j the

variables x_4 and x_5 are tested before x_1. Hence, only in this situation G'_i and G'_j can share an x_1-node. But then also the x_1-nodes in the constructed FBDDs for f_i and f_j are merged. Hence, G does not contain more x-nodes than G'. Because of the examples 6a and 6b the FBDD G' contains at least one y-node so that also the number of y-nodes in G is not larger than in G'. Also the number z-nodes does not increase, since z-nodes may be removed but no new z-nodes are inserted when constructing G from G'.

6. Proof of Lemmas 5 and 6

First we note that Lemma 6 implies Lemma 5. By the assumptions of Lemma 5 an FBDD G for the function F which has the Property $P(j-1)$ but not the Property $P(j)$ is given. Since G does not contain redundant nodes, at the 1-successor of the z^{j-1}-block of the switch an x-variable contained in $V_j \cup \cdots \cup V_{n+m}$ or y or a z-variable is tested. Since G represents F, it is consistent with all examples of the Tables I and II. Hence, by Lemma 6 the FBDD G contains two z^j-blocks. This implies the statement of Lemma 5.

Now we prove Lemma 6. We use the following notation. If ξ is an assignment to some set of variables and x is a variable that does not get a value by ξ, let $[\xi, x = 0]$ denote the assignment that we obtain by extending ξ by the assignment $x = 0$. Furthermore, we again consider $z^1, \ldots, z^{n+m}$ as ordinary variables and the z^i-blocks as ordinary nodes. We assume that $z^1, \ldots, z^{j-1}$ are replaced by the constant 1. The SubFBDD obtained from G' by this replacement is called G^*. Obviously, G^* is the SubFBDD of G' whose source is the 1-successor of the z^{j-1}-block of the switch. Let $X = \{x_1, \ldots, x_n, x'_1, \ldots, x'_n, y, z^j, \ldots, z^{n+m}\}$. Then the set of variables tested in G^* is some subset of X. In the following we only use examples where $z^1, \ldots, z^{j-1}$ take the value 1.

We outline the proof of Lemma 6: We first construct an assignment ξ to the variables in $X \setminus \{z^j\}$ and show that the represented function takes different values for the assignments $[\xi, z^j = 0]$ and $[\xi, z^j = 1]$. This implies that on the computation path for the partial input ξ in G^* a z^j-node v is reached. In a second step we construct an assignment σ of the variables in $X \setminus \{z^j\}$ and show in a similar way that on the computation path for σ some z^j-node w is reached. In order to prove $v \neq w$ we apply a cut-and-paste argument. Let Q be the computation path for ξ before the reached z^j-node. We construct an assignment π that takes for the variables tested on Q the same values as in ξ and for the other variables except z^j the same values as in σ. The variable z^j is undefined for π. We show that for some $c \in \{0, 1\}$ the represented function takes different values for $[\sigma, z^j = c]$ and $[\pi, z^j = c]$. This implies that for σ and π and, hence, also for σ and ξ not the same z^j-node can be reached, i.e., $v \neq w$. Therefore, there are two z^j-nodes.

For the assignment ξ all x-variables and all z-variables except z^j take the value 1 and y takes the value 0. Because of the examples 1a and 1b the represented function takes the value 1 for $[\xi, z^j = 0]$ and the value 0 for $[\xi, z^j = 1]$. Hence, there is a z^j-node v.

The choice of σ depends on the set of variables tested on Q. We distinguish the following cases.

Case 1: On Q some variable $x^* \in V_j$ is tested.

In σ the variable x^* gets the value 0, and all other x-variables, y and all z-variables except z^j

get the value 1. Because of the examples 2a and 2b the represented function takes the value 0 for $[\sigma, z^j = 0]$ and the value 1 for $[\sigma, z^j = 1]$. Hence, there is a z^j-node w.

Then by π all x-variables and all z-variables except z^j get the value 1. If in π the variable y gets the value 0, the represented function takes the value 1 for $[\pi, z^j = 0]$ because of example 1a. If y gets the value 1, the represented function takes the value 1 for $[\pi, z^j = 0]$ because of example 1c. Hence, $v \neq w$.

Case 2: On Q no variable of V_j, but z^l is tested.

Since G has the Property $P(j-1)$, we have $l > j$. Let $x^* \in V_j \setminus V_l$ be the variable, for which there are the examples 3. In σ the variables x^*, y and z^l get the value 0 and the remaining x-variables and z-variables except z^j get the value 1. Because of the examples 3a and 3b the represented function takes the value 0 for $[\sigma, z^j = 0]$ and the value 1 for $[\sigma, z^j = 1]$. Hence, there is a z^j-node w.

By π all x-variables except x^* and all z-variables except z^j get the value 1, and x^* and y get the value 0. Because of example 2c the represented function takes the value 0 for $[\pi, z^j = 1]$. Hence, $v \neq w$.

Case 3: On Q the variable y is tested, but on Q neither a variable of V_j nor a z-variable are tested.

We choose some $x^* \in V_j$. For σ the variable x^* gets the value 0 and all other variables except z^j get the value 1. Because of the examples 2a and 2b the represented function takes the value 0 for $[\sigma, z^j = 0]$ and the value 1 for $[\sigma, z^j = 1]$. Hence, there is a z^j-node w.

By π all x-variables except x^* and all z-variables except z^j get the value 1, and x^* and y get the value 0. Because of example 2c the represented function takes the value 0 for $[\pi, z^j = 1]$. Hence, $v \neq w$.

Case 4: On Q neither y nor a z-variable nor a variable of V_j is tested.

Hence, at the source of G^* some x-variable $x^* \notin V_j$ is tested. Let l be the maximal number for which $x^* \in V_l$. Since by the assumptions of the lemma at the source of G^* only x-variables in $V_j \cup \cdots \cup V_{n+m}$ or y or some z-variable may be tested, and by the assumptions of this case V_j-variables, y and z-variables are excluded, we have $l > j$. We choose σ in the following way. The variables x^*, y and z^l get the value 0 and all remaining x-variables and all remaining z-variables except z^j get the value 1. Because of the examples 4a and 4b the represented function takes the value 1 for $[\sigma, z^j = 0]$ and the value 0 for $[\sigma, z^j = 1]$. Hence, there is a z^j-node w.

By π the variables y and z^l get the value 0, and all x-variables and all remaining z-variables except z^j get the value 1. Because of example 4c the represented function takes the value 1 for $[\pi, z^j = 1]$. Hence, $v \neq w$.

This completes the proof of Lemma 6.

7. Conclusion and Open Problems

We proved that polynomial time approximation schemes of MinFBDD, OptGraphOrdering and MCFBDD are unlikely to exist. It remains an open problem whether there are polynomial time

approximation algorithms with some constant performance ratio for these problems or whether it can be shown that such approximation algorithms imply NP = P. Another open problem is the complexity of computing a minimal size FBDD if the function to be represented is given by its (complete) truth table. Note that the size of the input is now exponential in the number of variables so that our proof of the hardness of MCFBDD does not work. In the case of OBDDs this problem can be solved in polynomial time by the algorithm of Friedman and Supowit [7], while in the case of FBDDs the run time of the algorithm of Günther and Drechsler [10] may be still exponential.

The last open problem concern the complexity of the construction of a minimum size OBDD consistent with some set of examples, where the variable ordering of the OBDD is not fixed in advance. The proof of our hardness result for MCFBDD is based on the possibility to find a set of representative examples of the function used in the proof of the hardness result for MinFBDD. For OBDDs a similar proof seems to be more involved since the proofs of the hardness results for MinOBDD (Tani, Hamaguchi and Yajima [18], Bollig and Wegener [4], Sieling [13, 14]) use more complicated functions. Hence, we also get the motivation to find easier proofs of the hardness of MinOBDD.

Acknowledgment

I thank Petr Savický for drawing my attention to the learnability question for FBDDs and 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. (1986). Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers* 35, 677–691.

[6] 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*, LNCS 62, 227–240.

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

[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. In *Proc. of Asia and South Pacific Design Automation Conference*, 323–326.

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

[12] Pitt, L. and Valiant, L.G. (1988). Computational limitations on learning from examples. *Journal of the Association for Computing Machinery* 35, 965–984.

[13] 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*, LNCS 1373, 205–215.

[14] Sieling, D. (1998). The nonapproximability of OBDD minimization. ECCC Report TR98-001, Revision 1 (available from www.eccc.uni-trier.de).

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

[16] 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.

[17] Takenaga, Y. and Yajima, S. (1993). NP-completeness of minimum binary decision diagram identification. Techn. Report of IEICE, COMP 92-99, 57–62.

[18] 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*, LNCS 762, 389–398.

[19] Valiant, L.G. (1984). A theory of the learnable. *Communications of the ACM* 27, 1134–1142.

[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 Mathematical Foundations of Computer Science*, LNCS 176, 562–566.

ECCC ISSN 1433-8092
http://www.eccc.uni-trier.de/eccc
ftp://ftp.eccc.uni-trier.de/pub/eccc
ftpmail@ftp.eccc.uni-trier.de, subject 'help eccc'