ELSEVIER

Discrete Applied Mathematics 122 (2002) 263–282

DISCRETE APPLIED MATHEMATICS

The complexity of minimizing and learning OBDDs and FBDDs ☆

Detlef Sieling

FB Informatik, LS 2, Univ. Dortmund, 44221 Dortmund, Germany

Received 30 August 1999; received in revised form 12 January 2001; accepted 30 April 2001

Abstract

Ordered Binary Decision Diagrams (OBDDs) and Free Binary Decision Diagrams (FBDDs) are data structures for Boolean functions. They can efficiently be manipulated if only OBDDs respecting a fixed variable ordering or FBDDs respecting a fixed graph ordering are considered. In this paper, it is shown that the existence of polynomial time approximation schemes for optimizing variable orderings or graph orderings implies NP = P, and so such algorithms are quite unlikely to exist. Similar hardness results are shown for the related problems of computing minimal size OBDDs and FBDDs that are consistent with a given set of examples. The latter result implies that size bounded OBDDs and FBDDs are not PAC-learnable unless NP = RP.

Keywords: Binary decision diagram; 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

☆ This work was supported by DFG grants We 1066/8 and We 1066/9. A preliminary version appeared in Proceedings of 24th International Symposium on Mathematical Foundations of Computer Science, Lecture Notes in Computer Science, Vol. 1672, Springer, Berlin, 1999, p. 251–261 [19].

E-mail address: sieling@ls2.cs.uni-dortmund.de (D. Sieling).

0166-218X/02/$ - see front matter
PII: S0166-218X(01)00324-9

are Ordered Binary Decision Diagrams (OBDDs), which were introduced by Bryant [4]. 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 OBDDs and on a particular extension of OBDDs, namely Free BDDs (FBDDs).

In order to define and motivate the problems considered in this paper, we recall the definitions of OBDDs and FBDDs. A Binary Decision Diagram (BDD) for the representation of Boolean 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. An example of an FBDD is shown on the left side of Fig. 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. In OBDDs we have the extra condition that on all paths the variables are tested at most once and according to a fixed ordering, which is called the variable ordering. The size of a BDD is the number of internal nodes.

Each node v of a BDD 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. The chosen path from v to the terminal node is called the computation path for a. In a BDD for the representation of $f_1,\ldots,f_m$ for each function f_j there is a pointer to a node representing f_j. In the example on the left of Fig. 1 for the input $(x_1,\ldots,x_6)=(1,0,1,0,1,0)$ the computation path indicated by a dotted line is chosen. Since a terminal node labeled by 1 is reached, $f(1,0,1,0,1,0)=1$.

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 [25] and Žák [26], and in the paper of Simon and Szegedy [21] most previous approaches are handled in a unified way. Already in the early paper of Fortune, Hopcroft and Schmidt [5] 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 [20] 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.

A graph ordering describes for each input a permutation of the variables. Hence, graph orderings are a generalization of variable orderings. 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

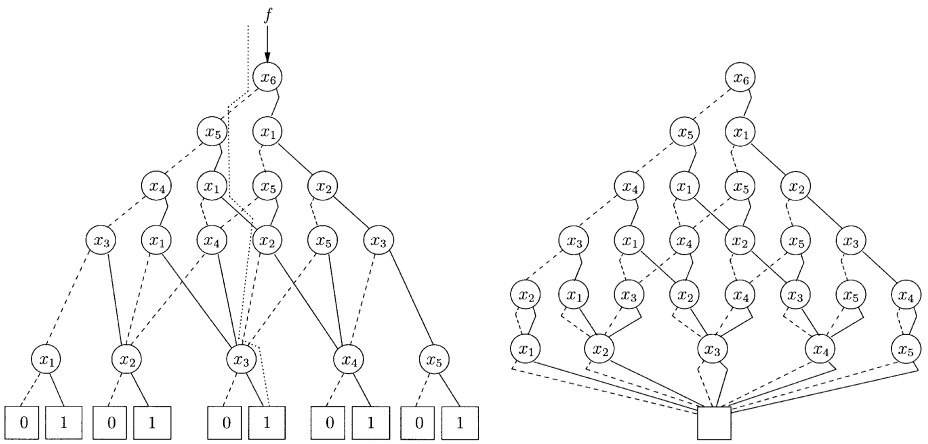

Fig. 1. An example of an FBDD H and of a graph ordering G such that H is a G-FBDD. In H the computation path for the input $(x_1,\ldots,x_6)=(1,0,1,0,1,0)$ is indicated by a dotted line.

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. An example of a graph ordering is shown on the right of Fig. 1. The FBDD on the left of Fig. 1 is driven by this graph ordering.

It is well-known that the OBDD size for functions may strongly depend on the chosen variable ordering [4]. Similarly the size of a G-FBDD for a particular function may strongly depend on the chosen graph ordering G. So the applicability of OBDDs and FBDDs depends on the ability to choose good variable orderings or good graph orderings. For this reason a large number of heuristics for the variable ordering problem has been proposed (e.g. [11,7,15]). An algorithm for finding an optimal variable ordering was presented, e.g., by Friedman and Supowit [6]. This algorithm has an exponential worst-case run-time. For the problem of computing good graph orderings only a heuristic for computing graph orderings of a tree-like shape has been proposed [2]. Günther and Drechsler [10] presented an algorithm for minimizing FBDDs with a double exponential worst-case run time. This algorithm can also be used to optimize graph orderings. In this paper, we explore the complexity of the corresponding optimization problems, which are defined in the following.

Minimum OBDD (MinOBDD)
Instance: A Boolean function f described by an OBDD G.
Problem: Compute an OBDD for f which has minimal size.

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^* such 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 OBDD or FBDD where the function to be represented is specified by a set of examples.

Minimum Consistent OBDD (MCOBDD)
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 OBDD of minimal size that represents a function f that is consistent with all examples in E, i.e., for which $\forall \langle x, \beta\rangle \in E: f(x) = \beta$.

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 f that is consistent with all examples in E, i.e., for which $\forall \langle x, \beta\rangle \in E: f(x) = \beta$.

These problems occur 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 OBDD or FBDD of size $|E| \cdot N$ that is consistent with the set E of examples. This OBDD or 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 terminal node labelled by β. However, such an OBDD or 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 [24]. 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 OBDD or 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 OBDDs or FBDDs, i.e. of the efficient construction of size bounded OBDDs or FBDDs well approximating a hidden concept, is determined by the complexity of MCOBDD or MCFBDD, resp. This follows from the results of Pitt and Valiant [14] 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 $\mathrm{NP} \neq \mathrm{RP}$.

The main results of this paper are the proofs of the following hardness results. The definition of approximation schemes is given in Section 2.

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

Theorem 2. *If there is a polynomial time approximation scheme for MinFBDD, then* $NP = P$.

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

Theorem 4. *If there is a polynomial time approximation scheme for MCOBDD or 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 [14] Theorem 4 also implies that for some function $s:\mathbb{N}\to\mathbb{N}$, OBDDs and FBDDs with $s(n)$ nodes are not learnable in the PAC-learning model unless NP = RP.

We review some known results about the considered problems. The NP-hardness of MinOBDD for OBDDs for multi-output functions was shown by Tani et al. [23] and for OBDDs for single-output functions by Bollig and Wegener [3]. Theorem 1 has already been proven in Sieling [17,18]. The proof in the present paper is much simpler than the proof in [17,18]. In Sieling [18] even the stronger result is proven that the existence of a polynomial time approximation algorithm with any constant performance ratio for MinOBDD implies NP = P. It remains an open problem whether MinFBDD, OptGraphOrdering, MCOBDD and MCFBDD can be approximated in polynomial time up to some constant factor or whether a similar nonapproximability result can be proven.

One may also define the analog of the problem OptGraphOrdering for OBDDs: OptVarOrdering is the problem to compute an optimal variable ordering for a function given by an OBDD. However, OptVarOrdering and MinOBDD are polynomially related and, therefore, usually not explicitly distinguished. On the other hand it is not clear whether OptGraphOrdering and MinFBDD are polynomially related, since it is not known whether a polynomial time algorithm for OptGraphOrdering implies a polynomial time algorithm for MinFBDD.

The learnability of size bounded OBDDs under the PAC-learning model was considered by Takenaga and Yajima [22]. They proved 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. The difference to our result concerning MCOBDD is that the variable ordering is fixed, while we do not assume anything about the variable ordering.

The paper is organized as follows. In the following section, we recall some properties of OBDDs and FBDDs and definitions of approximation algorithms. In Section 3 Theorems 1–3 are proven. Then we discuss the adaptation of this proof to the problems MCOBDD and MCFBDD (Section 4). Finally, we prove two technical lemmas in Section 5 and conclude the paper with some open problems.

2. Preliminaries

We recall some properties of OBDDs and FBDDs. There are two reduction rules for decreasing the size of OBDDs and FBDDs without changing the represented function:

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 OBDD or FBDD is called reduced if neither of the reduction rules is applicable. Bryant [4] showed that reduced OBDDs with a fixed variable ordering are unique up to isomorphism. Sieling and Wegener [20] proved a similar result for G-FBDDs. Hence, we may talk about *the* (reduced) OBDD or G-FBDD for some function f.

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 $OPT(I)$ 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 $1+\varepsilon$. If the run time of A is bounded by some polynomial in ε^{-1} and the input length, A is called fully polynomial time approximation scheme. Theorems 1–4 even hold, if the run time of the approximation schemes arbitrarily depends on ε.

3. The complexity of OBDD and 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 recall 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.

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, 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 such 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 MinOBDD and 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 [1,13]. In the first paper a reduction from the PCP-Theorem implies that there is some $\varepsilon > 0$ such 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 presented. 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 5. *There are constants $b \in \mathbb{N}$ and $\varepsilon > 0$ such that εRob3SAT-b is NP-hard.*

Proof of Theorems 1 and 2. We assume that there is a polynomial time approximation scheme A for MinOBDD or MinFBDD and construct a polynomial time algorithm for εRob3SAT-b where b and ε are the constants ensured by Theorem 5. 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 OBDD H (which is simultaneously an FBDD). 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 need not to consider this case.

We construct the OBDD H for a function F which is defined over the set

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

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

$$f_i = x_i \wedge x'_i \qquad \text{for } 1 \leqslant i \leqslant n,$$

$$f_{n+i} = \bigwedge_{u_j \in C_i} x_j \wedge \bigwedge_{\overline{u}_j \in C_i} x'_j \quad \text{for } 1 \leqslant i \leqslant m.$$

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 we define

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

We fix the following variable ordering: $z_1^1, z_2^1, z_3^1, z_1^2, \ldots, z_3^{n+m}, y, x_1, x'_1, \ldots, x_n, x'_n$. The shape of an OBDD for F is shown in the left side of Fig. 2. We see that the OBDD consists of a switch that chooses which of the functions f_i has to be evaluated.

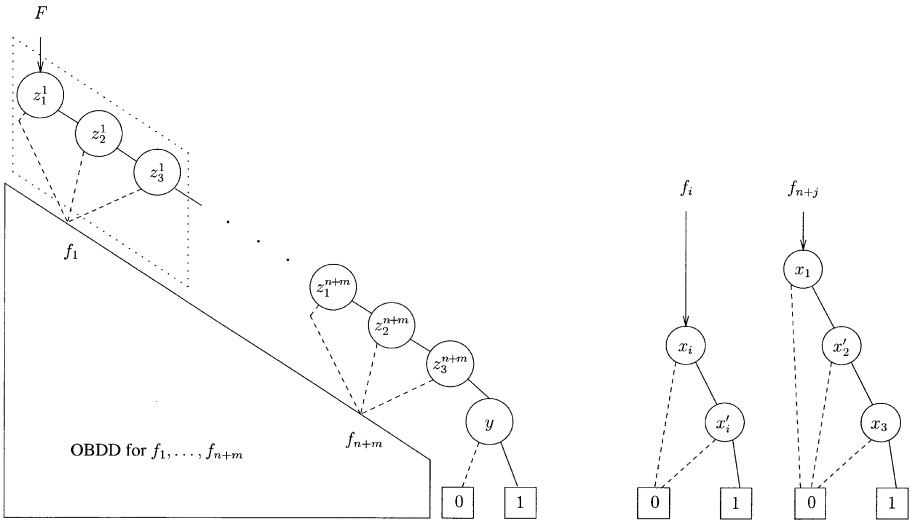

Fig. 2. The shape of an OBDD for the function F and OBDDs for $f_i = x_i \wedge x_i'$ and for $f_{n+j} = x_1 \wedge x_2' \wedge x_3$ (the function corresponding to the clause $C_j = \{u_1, \bar{u}_2, u_3\}$).

Fig. 2 shows that it is easy to construct OBDDs for $f_1, \ldots, f_{n+m}$ in polynomial time and to combine these OBDDs to the OBDD H for F in polynomial time.

Let L denote the total number of literals in the clauses in C. Let $\delta = \varepsilon/(21b)$. We shall prove the following two lemmas.

Lemma 6. *If (U, C) is satisfiable, the minimum OBDD size and the minimum FBDD size for F are both bounded above by $5n + 2m + L + 1$.*

Lemma 7. *If (U, C) is not satisfiable, the minimum OBDD size and the minimum FBDD size for F are both larger than $(1 + \delta)(5n + 2m + L + 1)$.*

Hence, the existence of a polynomial time approximation algorithm with a performance ratio of at most $1 + \delta$ and, in particular, the existence of a polynomial time approximation scheme for MinOBDD or MinFBDD implies that we can distinguish satisfiable and nonsatisfiable instances of εRob3SAT-b in polynomial time, which implies P=NP. Hence, Theorems 1 and 2 follow from Theorem 5 and Lemmas 6 and 7.

Proof of Lemma 6. We assume that (U, C) is satisfiable. Let σ be a satisfying assignment. We show that the size of a minimal OBDD for F is bounded above by $5n + 2m + L + 1$ by presenting an OBDD of this size.

We define $X_0 = \{x_i \mid \sigma(u_i) = 0\} \cup \{x_i' \mid \sigma(u_i) = 1\}$, i.e., the set of x-variables corresponding to the literals that take the value 0 in σ. Similarly, let $X_1 = \{x_i \mid \sigma(u_i) = 1\} \cup$

$\{x_i' \mid \sigma(u_i) = 0\}$, i.e., the set of x-variables corresponding to the literals that take the value 1 in σ. We choose a variable ordering π that starts with $z_1^1, z_2^1, z_3^1, z_1^2, \ldots, z_3^{n+m}, y$. After that the variables in X_0 follow in an arbitrary order, and finally, the variables in X_1 in an arbitrary order.

It is easy to construct an OBDD for $f_1, \ldots, f_n$ and π with $2n$ internal nodes. Then 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 OBDD consisting of $|C_i|$ internal nodes. Since σ is satisfying, by the choice of π the last variable is a variable corresponding to a literal of C_i that is satisfied by σ. If we join the constructed OBDDs for $f_1, \ldots, f_{n+m}$, the last internal node of the OBDD for f_{n+i} can be merged with a node of one of the OBDDs for $f_1, \ldots, f_n$. Hence, the OBDD for all these functions consists of at most $2n + \sum_{i=1}^{m}(|C_i| - 1) = 2n + L - m$ internal nodes. Finally, we construct from this OBDD for $f_1, \ldots, f_{n+m}$ an OBDD for F as outlined in Fig. 2. Then the number of nodes labeled by y and the z-variables is $3n + 3m + 1$ and the total number of internal nodes is $5n + 2m + L + 1$. □

Proof of Lemma 7. 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.

We start with an arbitrary FBDD G for F. If this FBDD does not have the shape of the OBDD in Fig. 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 such that afterwards the z-variables are arranged as shown in Fig. 2. 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.

We always assume that the considered FBDD does not contain redundant nodes. We call a node labeled by x_i redundant, if the function represented at this node does not essentially depend on x_i. A redundant node can be deleted by redirecting the incoming edges to one of the successors. 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 and, in particular, for the detection of redundant nodes.

In Fig. 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. 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_2^1- and z_3^1-nodes, respectively. W.l.o.g. 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 edge leading to a node labeled by z_2^1 or z_3^1 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

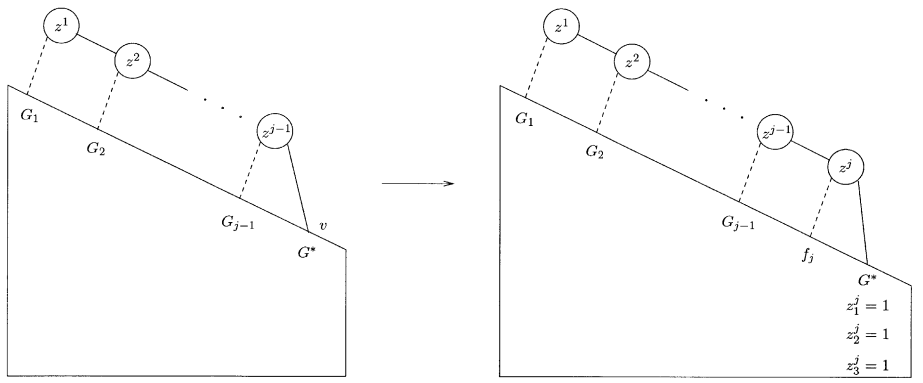

Fig. 3. The construction of an FBDD with the Property $P(j)$ from an FBDD with the Property $P(j-1)$.

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 Fig. 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. For $j \in \{1,\ldots,n+m\}$ the FBDD G has the Property $P(j)$ if the z^1-block,$\ldots,z^j$-block in G are arranged as shown in Fig. 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. In order to simplify the notation we say that G always has the (empty) Property $P(0)$.

Lemma 8. *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 5 where we prove a more general result. Now we can rearrange G in the following way. We search for the smallest j such that G does not have the Property $P(j)$. Then the z^1-,$\ldots,z^{j-1}$-blocks are arranged as shown on the left side of Fig. 3. In order to simplify the figure the 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 8 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. (If v is the source,

we define the first node of the created z^j-block as the new source.) 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 Fig. 3. 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 Property $P(j)$. Hence, we may iterate this procedure until the FBDD has the shape shown in Fig. 2. 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 topologically last internal node, 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 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 set of 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 $L \leqslant 3m$, $n \leqslant 3m$ 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 Lemma 7. □

Altogether, Theorems 1 and 2 are proven. Finally, we prove Theorem 3.

Proof of Theorem 3. In order to prove Theorem 3 we assume that there is a polynomial time approximation scheme A for OptGraphOrdering. We try to adapt the proof of Theorems 1 and 2. This means we construct for the instance (U,C) of εRob3SAT-b an FBDD for the function F as described above and apply A to this FBDD. Now we get the problem that the result of A is a graph ordering G instead of an FBDD, while by Lemmas 6 and 7 we only know of a relation between the size of a G-FBDD for F and the satisfiability of (U,C). Hence, we would like to compute a G-FBDD for F in order to determine its size. In general, for the computation of a G-FBDD from a

graph ordering G and a function given by an FBDD no polynomial time algorithm is known. However, here we have a slightly different situation, because the given FBDD for the function F is simultaneously an OBDD. Hence, an algorithm for computing a G-FBDD for F from a graph ordering G and an OBDD for F is sufficient, if the run time of this algorithm is bounded by some polynomial with respect to the input and output size. Using such an algorithm we can compute the G-FBDD for F and in the same way as in the proof of Theorems 1 and 2 we can decide in polynomial time whether (U, C) is satisfiable.

We do not describe in detail how the algorithm for computing a G-FBDD for a function F given by an OBDD and for a graph ordering G works, since this algorithm is an adaptation of a result in Sieling [16], which is also mentioned without proof in Meinel and Slobodová [12]. The result is the following statement: If there is a polynomial time algorithm for the equivalence test of FBDDs, there is an algorithm for the computation of a G-FBDD for a function given by an FBDD and for a graph ordering G, whose run time is bounded by some polynomial with respect to the input and output size. The algorithm given in Sieling [16] uses the oracle for the equivalence test of FBDDs for the equivalence test of subfunctions of F. This equivalence test is used in order to avoid the creation of multiple nodes representing the same function. In our situation an OBDD for F is given and we can use the equivalence test for OBDDs in order to test subfunctions of F for equivalence. Hence, the oracle for the equivalence test of FBDDs is no longer needed if the input is an OBDD, and we obtain a polynomial time algorithm for the construction of a G-FBDD for F from the OBDD for F and the graph ordering G. Altogether, Theorem 3 follows. □

4. The complexity of minimum consistent OBDD and FBDD

In order to prove Theorem 4 we again provide a reduction from εRob3SAT-b. Assume that A is a polynomial time approximation scheme for MCOBDD or MCFBDD, resp. 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 OBDD or FBDD G' we can decide whether (U, C) is satisfiable. We shall use several ideas of the proof of Theorems 1 and 2: The examples are defined over the set V of variables given in the proof of Theorems 1 and 2. All examples are consistent with the function F. We shall show that we can modify G' without increasing its size such that it afterwards represents the function F. Hence, we can use the analysis of the FBDD size for F given in Lemmas 6 and 7.

The examples constructed for (U, C) are listed in the Tables 1 and 2. For each $j \in \{1, \ldots, n+m\}$ there is a copy of Table 1. Let V_j denote the set of variables that f_j essentially depends on. Hence $|V_j| \in \{2, 3\}$. In Table 1 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. The quantifier "for all $l > j$ and for a fixed $x^* \in V_j \setminus V_l$" means that for each $l > j$ exactly one variable $x^* \in V_j \setminus V_l$ is chosen and the examples 3a and 3b are constructed for this

Table 1

	x-variables	y	z-variables except z^j	z^j	Function value	Upper bound on the 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	7×3
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 a fixed $x^* \in V_j \setminus V_l$						
3a	1 except $x^* = 0$	0	1 except $z^l = 0$	0	0	$7 \times 7\ (n+m)$
3b	1 except $x^* = 0$	0	1 except $z^l = 0$	1	1	$7\ (n+m)$
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	$7 \times 7 \times 2 \times n$
4b	1 except $x^* = 0$	0	1 except $z^l = 0$	1	0	$7 \times 2 \times n$
4c	1	0	1 except $z^l = 0$	1	1	$7 \times 2 \times n$
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 2

	x-variables	y	z-variables	Function value	Number of examples
6a	0	0	1	0	1
6b	0	1	1	1	1

choice of x^*. The entry in row 5a describes 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 an upper bound on 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 Theorems 1 and 2 and apply the polynomial time approximation scheme A for MCOBDD or MCFBDD, resp., for the performance ratio $1+\delta$ to the constructed set of examples. The result is an OBDD or FBDD G' for some function that is consistent with all examples. We remark that formally the names of the variables and the distinction of x-, y- and z-variables is not part of the input for the problem MCOBDD and MCFBDD. However, in the definition of MCOBDD

and MCFBDD it is implicitly assumed that the internal nodes of the output OBDD or FBDD are labeled by variables and that the correspondence of these variables and the input of the examples is known. Hence, in the following presentation we may assume that in the output OBDD or FBDD the same names of the variables as in Tables 1 and 2 are used. The computed OBDD or FBDD G' represents some function F', which may be different from F. We shall prove that we can compute an FBDD G from G' such that G represents F and $|G| \leqslant |G'|$. Hence, the analysis of the OBDD and FBDD size of F in Lemmas 6 and 7 implies the following.

1. If (U, C) is satisfiable, there is an OBDD (and therefore also 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 MCOBDD or MCFBDD, resp., 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 OBDD and 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 $|G| \leqslant |G'|$, 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. Together with Theorem 5 the existence of a polynomial time approximation scheme for MCOBDD or MCFBDD implies P = NP.

In the following we show how to construct G. The OBDD or FBDD G' is transformed into the FBDD G in the following three steps. We note that all intermediate results are FBDDs that are not larger than G' and that they 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 Lemma 7 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 Lemma 7 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 Fig. 2.

As in the proof of Lemma 7 we say that G' has the Property $P(j)$ iff the z^1-,...,z^j-blocks are arranged as shown in Fig. 2. 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 9. *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 9 in Section 5. Now we modify G' in a similar way as in the proof of Lemma 7 and as shown in Fig. 3. In G' we replace z^j by the constant 1. (Different from the proof of Lemma 7 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 1-successor is G^* after the replacement. For the 0-successor of the z^j-block we create an FBDD for f_j consisting of at most three internal nodes as shown on the right of Fig. 2. 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 such that at the edges leaving the switch the functions $f_1,\ldots,f_{n+m}$ and y are computed.

Let G_i' be the 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 $|G| \leqslant |G'|$.

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 such that also the number of y-nodes in G is not larger than in G'. Also the number of z-nodes does not increase, since z-nodes may be removed but no new z-nodes are inserted when constructing G from G'.

Altogether, we have shown how to construct G from the output of the approximation scheme. This completes the proof of Theorem 4.

5. Proof of Lemmas 8 and 9

First we note that Lemma 9 implies Lemma 8. By the assumptions of Lemma 8 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 1 and 2. Hence, by Lemma 9 the FBDD G contains two z^j-blocks. This implies the statement of Lemma 8.

Now we prove Lemma 9. We use the following notation. Let $c \in \{0,1\}$. If ξ is an assignment to some subset Z of the set of variables and x is a variable that does not get a value by ξ (i.e., $x \notin Z$), let $[\xi, x = c]$ denote the assignment that we obtain by extending ξ by the assignment $x = c$. 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 9. 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 la. 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 9. □

6. Conclusion and open problems

We proved that polynomial time approximation schemes for MinFBDD, OptGraphOrdering, MCOBDD and MCFBDD are unlikely to exist and presented a simplified proof for the corresponding statement for MinOBDD. It remains an open problem whether there are polynomial time approximation algorithms with some constant performance ratio for MinFBDD, OptGraphOrdering, MCOBDD and MCFBDD 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 such 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 [6], while in the case of FBDDs the run time of the algorithm of Günther and Drechsler [10] may be still exponential.

Acknowledgements

I thank Petr Savický for drawing my attention to the learnability questions and Ingo Wegener for fruitful discussions on the proofs in this paper.

References

[1] S. Arora, C. Lund, R. Motwani, M. Sudan, M. Szegedy, Proof verification and hardness of approximation problems, in: Proceedings of 33rd Symposium on Foundations of Computer Science, 1992, pp. 14–23.

[2] J. Bern, C. Meinel, A. Slobodová, Some heuristics for generating tree-like FBDD types, IEEE Trans. Comput. Aided Des. Integrated Circuits Systems 15 (1996) 127–130.

[3] B. Bollig, I. Wegener, Improving the variable ordering of OBDDs is NP-complete, IEEE Trans. Comput. 45 (1996) 993–1002.

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

[5] S. Fortune, J. Hopcroft, E.M. Schmidt, The complexity of equivalence and containment for free single variable program schemes, in: Proceedings of Fifth International Colloquium on Automata, Languages and Programming, Lecture Notes in Computer Science, Vol. 62, Springer, Berlin, 1978, pp. 227–240.

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

[7] M. Fujita, H. Fujisawa, N. Kawato, Evaluation and improvements of Boolean comparison method based on binary decision diagrams, in: International Conference on Computer-Aided Design, 1988, pp. 2–5.

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

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

[10] W. Günther, R. Drechsler, Minimization of free BDDs, in: Proceedings of Asia and South Pacific Design Automation Conference, 1999, pp. 323–326.

[11] S. Malik, A.R. Wang, R.K. Brayton, A. Sangiovanni-Vincentelli, Logic verification using binary decision diagrams in a logic synthesis environment, in: International Conference on Computer-Aided Design, 1988, pp. 6–9.

[12] C. Meinel, A. Slobodová, On the complexity of constructing optimal ordered binary decision diagrams, in: Proceedings of 19th International Symposium on Mathematical Foundations of Computer Science, Lecture Notes in Computer Science, Vol. 841, Springer, Berlin, 1994, pp. 515–524.

[13] C.H. Papadimitriou, M. Yannakakis, Optimization, approximation, and complexity classes, J. Comput. System Sci. 43 (1991) 425–440.

[14] L. Pitt, L.G. Valiant, Computational limitations on learning from examples, J. Assoc. Comput. Mac. 35 (1988) 965–984.

[15] R. Rudell, Dynamic variable ordering for ordered binary decision diagrams, in: Proceedings of International Conference on Computer-Aided Design, 1993, pp. 42–47.

[16] D. Sieling, Algorithmen und untere Schranken für verallgemeinerte OBDDs, Ph.D. Thesis, Universität Dortmund. Shaker, Aachen, 1995 (in German).

[17] D. Sieling, On the existence of polynomial time approximation schemes for OBDD-minimization (extended abstract), in: Proceedings of 15th Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Computer Science, Vol. 1373, Springer, Berlin, 1998, pp. 205–215.

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

[19] D. Sieling, The complexity of minimizing FBDDs (extended abstract), in: Proceedings of 24th International Symposium on Mathematical Foundations of Computer Science, Lecture Notes in Computer Science, Vol. 1672, Springer, Berlin, 1999, pp. 251–261.

[20] D. Sieling, I. Wegener, Graph driven BDDs—a new data structure for Boolean functions, Theoret. Comput. Sci. 141 (1995) 283–310.

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

[22] Y. Takenaga, S. Yajima, Hardness of identifying the minimum ordered binary decision diagram, Discrete Appl. Math. 107 (2000) 191–201.

[23] S. Tani, K. Hamaguchi, S. Yajima, The complexity of the optimal variable ordering problems of shared binary decision diagrams, in: Proceedings of Fourth International Symposium on Algorithms and Computation, Lecture Notes in Computer Science, Vol. 762, Springer, Berlin, 1993, pp. 389–398.
[24] L.G. Valiant, A theory of the learnable, Comm. ACM 27 (1984) 1134–1142.
[25] I. Wegener, On the complexity of branching programs and decision trees for clique functions, J. Assoc. Comput. Mac. 35 (1988) 461–461.
[26] S. Žák, An exponential lower bound for one-time-only branching programs, in: Proceedings of 11th International Symposium on Mathematical Foundations of Computer Science, Lecture Notes in Computer Science, Vol. 176, Springer, Berlin, 1984, pp. 562–566.