Electronic Colloquium on Computational Complexity - Reports Series 1997 - available via:

ECCC
TR97-060

FTP: ftp.eccc.uni-trier.de:/pub/eccc/
WWW: http://www.eccc.uni-trier.de/eccc/
Email: ftpmail@ftp.eccc.uni-trier.de with subject 'help eccc'

The Satisfiability Problem for Probabilistic Ordered Branching Programs

Manindra Agrawal *
Dept. of Computer Science
Indian Institute of Technology
Kanpur 208016, India

Thomas Thierauf
Abt. Theoretische Informatik
Universität Ulm
89069 Ulm, Germany

Abstract

We show that the satisfiability problem for bounded error probabilistic ordered branching programs is NP-complete. If the error is very small however (more precisely, if the error is bounded by the reciprocal of the width of the branching program), then we have a polynomial-time algorithm for the satisfiability problem.

1 Introduction

Branching programs are an interesting computational model to investigate. One reason for this is the tight relationship of the size of a branching program to the space needed by (nonuniform) Turing machines [Mas76] (see [BS90]). Another reason is the use of restricted kinds of branching program in applications as for example circuit verification (see Bryant [Bry92] for a broad overview).

Definition 1.1 *A* (deterministic) branching program B in n variables $x_1, \ldots, x_n$ *is a directed acyclic graph with the following type of nodes. There is a single node of indegree zero, the* initial node of B. *All nodes have outdegree two or zero. A node with outdegree two is an* internal node of B *and is labelled with a variable* x_i, *for some* $i \in \{1, \ldots, n\}$. *One of its outgoing edges is labelled with* 0, *the other with* 1. *A node with outdegree zero is a* final node of B. *The final nodes are labelled either with* accept *or* reject. *The* size *of a branching program is the number of its nodes.*

A branching program B in n variables defines an n-ary Boolean function from $\{0,1\}^n$ to $\{0,1\}$ in the obvious way: for an assignment $\boldsymbol{a} = (a_1, \ldots, a_n) \in \{0,1\}^n$, *we walk through B, starting at the initial node, always following the (unique) edge labelled a_i when the node has label x_i, until we reach a final node. If the final node is an accepting node, we define $B(\boldsymbol{a}) = 1$, and $B(\boldsymbol{a}) = 0$ otherwise.*

The restrictions on branching programs usually considered bound the number of times a variable can be tested.

Definition 1.2 *A branching program is called* read-k-times, *if, on each path from the initial node to a final node, every variable occurs at most k times as a node label.*

*Research done in part while visiting the university of Ulm, Germany. Supported in part by an Alexander von Humboldt fellowship.

Of particular interest are read-once branching program where the variables are read in a certain fixed order.

Definition 1.3 *A* ordered branching program *(also called* ordered binary decision diagram, OBDD *for short) is a read-once branching program such that there is a permutation* π *on* $\{1,\ldots,n\}$ *such that if an edge leads from a node labelled* x_i *to a node labelled* x_j*, then* $\pi(i) < \pi(j)$.

Ordered branching programs have the advantage that one can efficiently manipulate them. For example, given two ordered branching programs, one can easily construct a new one computing the conjunction of the given ones. Also, there are fast, in fact, linear time algorithms to check the equivalence of two ordered branching programs.

The main drawback of ordered branching programs is their limited computational power. For example, multiplication requires exponential size ordered branching programs [Bry91]. (For more lower bounds see for example [BRS93, BHR95, Juk89, KMW91, Pon95, Weg88].) It is therefore of great interest whether one can find some less restrictive model in order to be able to compute more functions within small size, but, at the same time, to maintain all the nice properties ordered branching programs have.

A bad choice, for example, seems to be read-once branching programs. Not only many of the lower bound proofs for ordered branching programs work as well in the read-once model, also in general one cannot combine them according to Boolean operations: there are examples of functions that have small read-once branching programs, but their conjunction requires exponential size.

Gergov and Meinel [GM96] proposed *parity ordered branching programs,* $\oplus$*-OBDDs* for short. This model is based on *nondeterministic branching programs.*

Definition 1.4 Nondeterministic branching programs *are defined as deterministic ones but that can have additional unlabelled nodes with unbounded fan-out.*

On some input, when the evaluation reaches such a nondeterministic node*, one can choose an arbitrary edge to proceed. A nondeterministic branching program* accepts *an input, if there exists a path that leads to the accepting node. Otherwise the input is* rejected.

Parity ordered branching programs, $\oplus$*-OBDDs, are nondeterministic branching programs that accept a given input iff there is an odd number of accepting paths.*

Some of the functions where ordinary ordered branching programs need exponential size have small $\oplus$-OBDDs. Gergov and Meinel [GM96] show that they have Boolean closure properties and give an efficient probabilistic equivalence test. Very recently, Waack [Waa97] exhibited even a polynomial-time equivalence test for $\oplus$-OBDDs.

In this paper we consider *probabilistic branching programs* introduced by Ablayev and Karpinski [AK96].

Definition 1.5 Probabilistic branching programs *are defined as nondeterministic ones but the nondeterministic nodes are now called* probabilistic nodes.

On some input, when we reach a probabilistic node, the edge to proceed is chosen under uniform distribution out of all outgoing edges. A probabilistic branching program accepts its input if the probability of reaching the accepting node is at least 1/2. Otherwise the input is rejected.

A probabilistic branching program has bounded error*, if there is an* $\epsilon > 0$ *such that the acceptance probability is either at most* $1/2 - \epsilon$ *or at least* $1/2 + \epsilon$ *on all inputs.*

We use BP-OBDD *as a short hand for* bounded error probabilistic ordered branching programs.

Ablayev and Karpinski [AK96] exhibit a function f that requires exponential-size read-once branching programs, whereas f can be computed by polynomial-size BP-OBDDs. Later on, Ablayev [Abl97] extended the gap to *nondeterministic* read-once branching programs.

Another example is PERMUTATION-MATRIX, the problem to decide whether a given $n \times n$ 0-1-matrix is a permutation matrix. That is, whether there is precisely one 1 in every row and every column. PERMUTATION-MATRIX requires exponential-size nondeterministic read-once branching programs [Juk89, KMW91], whereas it can be computed by polynomial-size BP-OBDDs [Sau97].

We add a further example to this list: the CLIQUE-ONLY function. Given a graph G with n nodes and a $k \leq n$. One has to determine whether G has a k-clique and the clique edges are the only edges of G. CLIQUE-ONLY requires exponential-size nondeterministic read-once branching programs [BRS93]. We show that it can be computed by polynomial-size BP-OBDDs.

On the other hand, the INDIRECT-STORAGE-ACCESS and the HIDDEN-WEIGHTED-BIT function require exponential-size BP-OBDDs [Sau97] (see [Abl97, Sau97] for more lower bounds).

It is easy to see that bounded error probabilistic ordered branching program are closed under Boolean combinations. So the most interesting open question with respect to this model is to ask for *efficient satisfiability- or equivalence tests*. In this paper, we solve this open problem. However, we give a negative answer with respect to the most interesting cases in Section 4: the satisfiability problem for bounded error probabilistic ordered branching programs is NP-complete. Only if the error of the branching program is bounded by the reciprocal of its width we have a polynomial-time algorithm for the satisfiability problem. This is shown in Section 5. Because the equivalence problem is reducible to the satisfiability problem, this also provides an efficient equivalence test for probabilistic ordered branching program with small error.

We start by providing some basic facts about probabilistic branching programs in the next section.

2 Basic Properties

Ordered branching programs, OBDDs, are somehow similar to finite automata, with the difference that branching programs are a *nonuniform* model and that the input might be read in a different order than just from left to right. However, many of the construction done with finite automata can be adapted to ordered branching programs. For example they have a canonical form [Bry86]: for any ordered branching program there is a uniquely determined minimal equivalent one with respect to this order. Since the minimization process can be carried out efficiently, this also provides a polynomial-time equivalence test. Another example is the construction of the *cross product* of two such programs that obey the same order. This essentially allows to combine ordered branching programs according to Boolean operations [Bry86, Bry92].

We sketch the construction for two BP-OBDDs B_0 and B_1 that obey the same order. Assume that B_0 and B_1 are *layered* such that there are alternating probabilistic and deterministic nodes, that furthermore *all* variables appear on every path, so that the same variable is tested in every deterministic layer. Edges go only from one level to the next. This can easily be achieved by introducing *redundant nodes*. These are nodes which, in case of a deterministic node, have both its edges going to the same node. In case of a probabilistic node there is only one edge that goes to some node with probability 1. Now we define program B. B has the same layers as B_0 and B_1, the nodes of each layer are the cross product of the nodes of B_0 and B_1 at the corresponding layer. Edges are defined such that B simulates B_0 and B_1 in parallel. That is, there is an edge from node (u, v) to (u', v') in succeeding levels of B, if there is an edge from u to u' in B_0 and from v to v' in B_1. The size of B is bounded by $|B_0||B_1|$ and the number of paths

that reach a node multiply: if some input x is accepted by B_0 with probability p_0 and by B_1 with probability p_1, then B on input x reaches the (accept,accept)-node with probability p_0p_1, the (accept,reject)-node with probability $p_0(1-p_1)$, the (reject,accept)-node with probability $(1-p_0)p_1$, and the (reject,reject)-node with probability $(1-p_0)(1-p_1)$.

Using the cross product, one can achieve *probability amplification* of BP-OBDDs. Let B be such a program in n variables that computes some function f with error $1/2-\epsilon$. That is,

$$\mathbf{Prob}[B(\boldsymbol{x}) = f(\boldsymbol{x})] \quad \geq \quad \frac{1}{2} + \epsilon.$$

We apply the above cross product construction t times with $B = B_0 = B_1$. This yields program B^t that consists of t factors B. The acceptance of B^t is defined according to a *majority vote* on its t factors. The size of B^t is bounded by $|B|^t$. Since t is in the exponent, we have to choose t constant in order to keep B^t within size polynomial in $|B|$. Therefore, by standard arguments (for example using Chernoff-bounds), we can amplify the correctness of B from $\frac{1}{2}+\epsilon$ to $1-\delta$, for any constant $0 < \delta < 1/2$.

Lemma 2.1 *Let B be a BP-OBDD that computes some function f with error $1/2-\epsilon$ and let $0 < \delta < 1/2$. Then there is BP-OBDD of size polynomial in $|B|$ that computes f with error δ.*

As a first application we show that BP-OBDDs can be combined according to Boolean operations.

Lemma 2.2 *BP-OBDDs (with the same order) can be combined in polynomial time according to any Boolean operation.*

Proof. BP-OBDDs can be complemented by exchanging accepting and rejecting states. Therefore it remains to show how to construct the conjunction of two BP-OBDDs B_0 and B_1 with n variables. Let the error of both be at most $1/4$ (applying Lemma 2.1).

The idea to get a BP-OBDD that computes $B_0 \wedge B_1$ is the same as for deterministic OBDDs: assume that B_0 and B_1 are layered such that deterministic and probabilistic node alternate and that all variables occur on every path. Then we can build the cross product $B = B_0 \times B_1$ and define the (accept,accept) node as the accepting node of B and the other leafs as rejecting nodes.

Let $\boldsymbol{a} \in \Sigma^n$. If $\boldsymbol{a}$ is accepted by both, B_0 and B_1, then B accepts $\boldsymbol{a}$ with probability at least $(3/4)\ (3/4) = 9/16$. On the other hand, If $\boldsymbol{a}$ is rejected by either B_0 or B_1, then B accepts $\boldsymbol{a}$ with probability at most $1/4$. □

3 The Computational Power

As already mentioned, there are some functions that can be computed by small BP-OBDDs but require exponential size ordered (in fact, read-once) branching programs. In this section we give some examples to demonstrate how branching programs can use randomization.

Although polynomial size BP-OBDDs cannot multiply [AK97] they can nevertheless *verify* multiplication. That is, given x, y, and z they can check if $xy = z$.

Theorem 3.1 *BP-OBDDs can verify multiplication with one-sided error and within polynomial-size.*

Proof. Given n-bit numbers x and y and $2n$-bit number z. Small branching programs cannot handle such numbers. Instead, we do computations *modulo some small prime* p.

For example it is easy to construct an ordered branching program that computes $(x \bmod p)$ in the sense that there are p final nodes numbered $0, \ldots, p-1$ such that the program ends up in node $(x \bmod p)$. Based on this, we construct an ordered branching program $B_p(x, y, z)$ that checks whether

$$xy \equiv z \pmod{p}.$$

B_p starts by computing $(x \bmod p)$. Then we read the bits of $y = y_{n-1} \cdots y_0$. Since

$$(x \bmod p)\ y \equiv \sum_{i=0}^{n-1} (x \bmod p)\ 2^i y_i \pmod{p},$$

we can also compute $(xy \bmod p)$. Now it remains to compute $(z \bmod p)$ and to compare it with $(xy \bmod p)$. The size of B_p is $O(p^2 n)$. Note that B_p is ordered.

If indeed $xy = z$, then B_p will accept independently of p. On the other hand, if $xy \neq z$ then B can accept anyway for some prime p, because we could still have that $xy \equiv z \pmod{p}$ in this case. However, since these numbers are bounded by 2^{2n}, there are at most $2n$ primes where our test can fail.

Our final program B therefore probabilistically branches to programs $B_{p_1}, \ldots, B_{p_{4n}}$, where $p_1, \ldots, p_{4n}$ are the first $4n$ prime numbers. Each of those checks whether $xy \equiv z \pmod{p_i}$. If $xy = z$, then $\mathbf{Prob}[B(X) \text{ accepts}] = 1$. Otherwise $\mathbf{Prob}[B(X) \text{ accepts}] \leq 1/2$.

By the Prime Number Theorem p_{4n} is polynomially bounded in n. Therefore B has polynomial size. □

Another example is provided by the CLIQUE-ONLY function. Recall that on input of a graph G and a k, we have to decide whether G has a k-clique and no other edges outside the clique.

Theorem 3.2 CLIQUE-ONLY *has polynomial-size BP-OBDDs with one-sided error.*

Proof. Let A be an adjacency matrix of a graph G with n nodes, and $k \leq n$. G has only a k-clique iff

(i) there exist k rows such that each contains precisely $k-1$ ones and the remaining rows are all zero, and

(ii) any two nonzero rows must be identical except for the positions where they intersect the main diagonal.

Condition (i) is easy to check, even for deterministic OBDDs. The variable order is *row-wise*, i.e., $x_{1,1} < x_{1,2} < \cdots < x_{n,n-1} < x_{n,n}$. Therefore it remains to check condition (ii) with an BP-OBDD that has the same order, and then apply Lemma 2.2.

Suppose we add a 1 at the diagonal positions of the nonzero rows of A. Then condition (ii) says that the resulting nonzero rows must be identical. Let $r_1, \ldots, r_n$ denote the rows of A and interpret them as binary numbers. Introducing a one at the diagonal position of nonzero row r_j corresponds to adding 2^{n-j} to r_j. Therefore it suffices to check that for any two consecutive nonzero rows r_j and r_k, we have

$$r_j + 2^{n-j} \quad = \quad r_k + 2^{n-k}. \tag{1}$$

We construct a deterministic OBDD B_p that verifies equation (1) modulo some small prime p. B_p looks for the first nonzero row, say j and computes $s = (x_j + 2^{n-j} \bmod p)$ by doing a binary count modulo p as in the previous theorem. Then B_p checks for each forthcoming nonzero row k that $x_k + 2^{n-k} = s$. (again by counting modulo p to determine the value $(x_k + 2^{n-k} \bmod p)$). The size of B_p is $O(n^2p^2)$.

Now we can again use the same technique as in Theorem 3.1 to obtain a polynomial-size BP-OBDD that checks condition (ii). □

4 NP-Complete Satisfiability Problems

In this paper we are mainly interested in satisfiability and equivalence problems. Note that the satisfiability problem is at most as hard as the equivalence problem, since satisfiability asks for (not) being equivalent with the all-zero function.

Consider a read-once branching program. Here, the satisfiability problem is trivial since it is enough to check that there is a path from the initial to the accepting node. It is well known that already for the extension to *read-twice* branching programs, the satisfiability problem is NP-complete. Only in the restricted case that there are a constant number of layers of ordered branching programs, all respecting the same order (so called k-OBDDs), the satisfiability problem stays in P [BSSW94].

The above argument still works for *nondeterministic* read-once branching programs. (Also the argument in [BSSW94] for k-OBDDs goes through.) However, this is not clear for *probabilistic* read-once branching programs, not even for ordered ones. The task here is to find an input that is accepted with high probability by a such a program B. What we *can* do is the following: for every given input $\boldsymbol{a} \in \{0,1\}^n$ we can determine how many paths in B lead to the accepting, respectively rejecting node. That is, we can compute $\mathbf{Prob}[B(\boldsymbol{a}) \text{ accepts}]$ in polynomial time. The satisfiability problem for probabilistic read-once branching programs (with unbounded error) can be stated as

$$\exists \boldsymbol{a} : \ \mathbf{Prob}[B(\boldsymbol{a}) \text{ accepts}] \geq \frac{1}{2}.$$

Therefore it is in NP. It is also NP-complete:

Proposition 4.1 *The satisfiability problem for probabilistic ordered branching programs (with unbounded error) is* NP*-complete.*

Proof. We reduce CNF-SAT. Let $F = \bigwedge_{i=1}^{m} C_i$ be a CNF-formula with m clauses $C_1, \ldots, C_m$. We construct a probabilistic ordered branching program B_F such that

$$F \in \text{SAT} \quad \Longleftrightarrow \quad \exists \boldsymbol{a} : \ \mathbf{Prob}[B_F(\boldsymbol{a}) \text{ accepts}] \geq \frac{1}{2}.$$

Let B_i be a deterministic ordered branching program that accepts if clause C_i is satisfied on a given input. B_F is constructed as follows. The initial node of B_F is a probabilistic node that branches $2m$ times. For m of these edges, each leads to the initial node of exactly one program B_i. The remaining m edges go directly to the rejecting node.

It follows that B_F accepts input $\boldsymbol{a}$ if and only if all the programs B_i accept (recall that these are deterministic), which is only possible when $\boldsymbol{a}$ satisfies F. □

When considering the case of *bounded error*, there is a subtlety on how to define the satisfiability problem precisely: let B be a probabilistic ordered branching program and lets fix the

error bound $\epsilon = 1/4$. Then B accepts an input $\boldsymbol{a}$, if $\mathbf{Prob}[B_F(\boldsymbol{a})$ accepts$] \geq 3/4$. Additionally we also would have to check that B has in fact bounded error on *all* inputs. However, already this latter problem is coNP-complete.

Proposition 4.2 *Given a probabilistic ordered branching program B and an $\epsilon > 0$. The problem to decide whether B is of bounded error ϵ is* coNP*-complete.*

Proof. The argument is essentially the same as for Proposition 4.1. Consider the case $\epsilon = 1/4$. Construct B_F as above but with $4m-4$ edges leaving the initial node and $3m-4$ of them going directly to the rejecting node. Then we have

$$F \in \text{SAT} \quad \Longleftrightarrow \quad \exists \boldsymbol{a} : \ \frac{1}{4} < \mathbf{Prob}[B_F(\boldsymbol{a}) \text{ accepts}] < \frac{3}{4}.$$

□

Hence, efficient satisfiability algorithms can only exist for the *promise version* of the problem: given B and $\epsilon > 0$, we take as a promise that B is in fact a probabilistic ordered branching program *with error* ϵ. With this assumption we want to decide whether $\exists \boldsymbol{a} : \ \mathbf{Prob}[B_F(\boldsymbol{a})$ accepts$] \geq 1/2 + \epsilon$. If the promise is not true, then we can give an arbitrary answer.

However, (unfortnately, from a practical point of view) even the promise version of the satisfiability problem for BP-OBDDs is NP-complete.

Theorem 4.3 *The satisfiability problem for BP-OBDDs is* NP*-complete.*

Proof. Manders and Adleman [MA78] have shown that some specific Diophantine equations so called *binary quadratics*, are NP-complete. More precisely, the following set Q defined over the natural numbers is NP-complete:

$$Q = \{ (a,b,c) \mid \exists x, y : \ ax^2 + by = c \}.$$

As a slight generalization of Theorem 3.1, BP-OBDDs can verify such binary quadratics. That is, the set

$$Q' = \{ (a,b,c,x,y) \mid ax^2 + by = c \}$$

can be accepted by a polynomial-size BP-OBDD, call it B.

For fixed a, b, c, we can construct a BP-OBDD B_{abc} from B that computes the subfunction of B with a, b, and c plugged in as constants. Recall that B is deterministic except for the root node. Therefore we can obtain B_{abc} by reducing B appropriately. For example, to fix variable x_1 to a_1 in B, we construct $B_{x_1=a_1}$ as follows: eliminate all nodes labelled x_1 in B and redirect edges to such a node w to the node that follows the a_1-edge of w.

For all natural numbers a, b, c, we have that

$$(a,b,c) \in Q \quad \Longleftrightarrow \quad B_{abc} \text{ is satisfiable.}$$

This proves the theorem. □

Corollary 4.4 *The equivalence problem for BP-OBDDs is* coNP*-complete.*

5 An Efficient Satisfiability Test for BP-OBDDs with Small Error

Ordered branching programs can be *layered*: by introducing redundant nodes we can achieve that every variable occurs on every path of the program. Then all nodes that test the same variable have the same distance to the root, they form a *layer* of the program. The maximum number of nodes in a layer is called the *width* of the program.

We can extend these notions to BP-OBDDs: here we also have *probabilistic layers* that contain probabilistic nodes only. Then we require that deterministic and probabilistic layers alternate. The *width* is again the maximum size of a layer.

The main result in this section is an efficient satisfiability test for BP-OBDDs that have small error, namely error bounded by $1/(\text{width}+2)$. That is, we consider the following problem:

BOUNDED-WIDTH-BP-OBDD-SAT
Given a BP-OBDD B with error ϵ and width W such that $\epsilon < 1/(W+2)$.
Decide whether B is satisfiable.

Theorem 5.1 BOUNDED-WIDTH-BP-OBDD-SAT $\in$ P.

Proof. Let B be some BP-OBDD with n variables $x_1, \ldots, x_n$, width W and error $\epsilon < 1/(W+2)$. W.l.o.g. we can assume that B is layered, so that probabilistic and deterministic layers alternate. We number the layers according to their distance to the root. The root layer (which is a single node) has number 0.

We want to find out whether B is satisfiable, i.e., whether there exists an input such that B accepts with high probability. We traverse B layer by layer, starting at the initial node, i.e. with layer $\ell = 0$ and $B_0 = B$. In each level we transform the branching program. Also the probabilities that a probabilistic node branches to its successors will change. In the beginning, all probabilities have the form $1/p$ if a node has p successors. Since we will also get other rational numbers as probabilities, we generalize the BP-OBDD model and write the probabilities on the edges, maybe as pairs of integers in binary representation.

Suppose we have reached layer $\ell \geq 0$ and let B_ℓ denote the branching program constructed so far. B_ℓ has the following properties:

1) B_ℓ is *deterministic* up to layer $\ell - 1$, and identical to B from layer $\ell + 1$ downwards,

2) the error of B_ℓ is bounded by ϵ,

3) the width of B_ℓ is at most $W + 1$, and

4) B_ℓ is satisfiable iff B is satisfiable.

In general, B_ℓ accepts only a subset of the strings accepted by B. Nevertheless, we ensure property (4) which is enough to check the satisfiability of B. In particular, the resulting branching program after we have processed the last level is a *deterministic* ordered branching program. Hence this will solve our problem since the satisfiability problem for ordered branching programs is trivial: it is a graph reachability problem.

We now describe how to process layer $\ell \geq 0$. Let ℓ be the smallest probabilistic layer of B_ℓ. (If layer ℓ is deterministic then define $B_{\ell+1} = B_l$ and proceed to the next layer.) We consider three consecutive layers:

layer ℓ consists of k probabilistic nodes u_1, u_2, ..., u_k. We can inductively assume that the part of B_l from the initial node to the u-nodes is deterministic and that $k \leq W + 1$.

layer $\ell + 1$ consists of $l \leq W$ deterministic nodes v_1, v_2, ..., v_l which all query the variable x_r.

layer $\ell + 2$ consists of $m \leq W$ probabilistic nodes w_1, w_2, ..., w_m.

These nodes are connected as follows.

u-nodes with v-nodes: each node u_i is connected to all the nodes v_j with the edge between them having probability p_{ij} (in case u_i is not connected to some node v_j, we take the probability p_{ij} to be zero). We have $\sum_j p_{ij} = 1$ for each $1 \leq i \leq k$.

v-nodes with w-nodes: node v_j is connected to nodes $w_{e(j,0)}$ and $w_{e(j,1)}$ via edges labelled 0 and 1 respectively.

Our first step is to make the k u-nodes deterministic. For this we introduce $2k$ new nodes at layer $\ell + 1$, call them v_1^0, v_1^1, v_2^0, v_2^1, ..., v_k^0, v_k^1 (these nodes will be probabilistic) that replace the old v-nodes. The u-nodes get label x_r, the variable queried by the old v-nodes, and we put the b-edge of u_i to node v_i^b, for $b \in \{0, 1\}$.

Next, we introduce an edge from node v_i^b to the node w_t and assign probability $q_{b,i,t}$ to it such that $q_{b,i,t}$ is precisely the probability to reach w_t from u_i in B_ℓ if x_v has value b. This is achieved by defining

$$q_{b,i,t} \quad = \quad \sum_{\substack{j \\ e(j,b)=t}} p_{ij}.$$

Finally, delete all the old v-nodes v_j and edges adjacent to them. Let B'_ℓ be the branching program constructed so far. Clearly B'_ℓ is equivalent to B_ℓ. In fact, the probability to reach a node w_t has not changed. So in particular, the error probability of B'_ℓ is the same as in B_ℓ, i.e., at most ϵ.

Now we have two consecutive layers of probabilistic nodes. We can merge the layer of w-nodes into the v-nodes and change the probabilities on the edges going out from v-nodes appropriately such that the previous probabilities to reach a node in the layer below the w-nodes is not altered. However, the number of v-nodes we obtain that way is $2k$ (recall that k is the number of probabilistic u-nodes in B_ℓ). Therefore, if we just use this process to eliminate all the probabilistic nodes from B_ℓ, we might end up with an exponential number of nodes in the final branching program. So we have to reduce the number of v-nodes in B'_ℓ before merging them with the w-nodes.

Now we use the fact that we only want to know whether B_ℓ *is satisfiable*. Therefore we can throw out some v-nodes from B'_ℓ as long as we can guarantee that the reduced branching program is still satisfiable if B'_ℓ is satisfiable.

Below we give a method to reduce the number of v-nodes to at most $m + 1$ (recall that m is the number of w-nodes). Then we obtain program $B_{\ell+1}$ by merging the remaining v-nodes with the w-nodes, as described above, so that $B_{\ell+1}$ has at most $m + 1$ probabilistic v-nodes at layer $\ell + 1$ and no w-nodes. Since $m \leq W$, the width of $B_{\ell+1}$ is bounded by $W + 1$. Hence this will suffice to complete the algorithm.

A v-node v_i^b in B'_ℓ is connected to a w-node w_t via an edge with probability $q_{b,i,t}$. Notice that *there are no probabilistic nodes above the layer of v-nodes.* Therefore, B'_ℓ is satisfiable iff there is a v-node v_i^b such that the acceptance probability of that node on some input, i.e., the

probability of reaching the accepting node from v_i^b on the input, is at least $1-\epsilon$. Since the error of B'_ℓ is bounded by ϵ, each v-node has acceptance probability of either at least $1-\epsilon$ or at most ϵ on any input.

With each v-node v_i^b we associate a point $\boldsymbol{q}_{b,i} \in \boldsymbol{Q}^m$, where the t-th coordinate is given by $q_{b,i,t}$. Let $\boldsymbol{a}$ be some input to B'_ℓ that reaches node v_i^b and let $\boldsymbol{y_a} = (y_1, \ldots, y_m) \in \boldsymbol{Q}^m$ be the acceptance probabilities of $\boldsymbol{a}$ when starting at nodes $w_1, \ldots, w_m$. Then $\boldsymbol{a}$ is accepted by B'_ℓ with probability $\boldsymbol{q}_{b,i} \cdot \boldsymbol{y_a}$.

We are seeking for an $\boldsymbol{a}$ that is accepted by B'_ℓ with probability at least $1-\epsilon$. The trick now is to relax the condition a bit: instead of $\boldsymbol{a}$, we try to find $\boldsymbol{y} \in [0,1]^m$ such that $\boldsymbol{q}_{b,i} \cdot \boldsymbol{y} \geq 1-\epsilon$, even though $\boldsymbol{y}$ might not occur as a probability vector $\boldsymbol{y_a}$ at the w-nodes for any input $\boldsymbol{a}$ that reaches node v_i^b. More precisely, we take $\boldsymbol{y}$ as an unknown and try to find out whether

there exists a $\boldsymbol{y}$ *such that* v_i^b *is* the only *v-node that has acceptance probability* $1-\epsilon$.

If this is *not* the case, then

- either no v-node has acceptance probability $1-\epsilon$ (including v_i^b) *on any input*,
- or there is an input on which both v_i^b *and some other v-node* have acceptance probability $1-\epsilon$.

In both cases we can safely delete the node v_i^b from B'_ℓ and still maintain the property that the resulting branching program is satisfiable if and only if B'_ℓ is.

v_i^b is the only v-node that has acceptance probability $1-\epsilon$ if the following system of linear inequalities has a solution $\boldsymbol{y} \in \boldsymbol{Q}^m$:

$$\begin{aligned} \boldsymbol{q}_{b,i} \cdot \boldsymbol{y} &\geq 1-\epsilon, \\ \boldsymbol{q}_{a,j} \cdot \boldsymbol{y} &\leq \epsilon, \quad \text{for } (a,j) \neq (b,i), \text{and} \\ 0 \leq y_t &\leq 1, \quad \text{for } 1 \leq t \leq m. \end{aligned}$$

After deleting a v-node for which this set of inequalities has no solution, we repeat the above process again for other v-nodes until the above set of inequalities has a solution for every remaining v-node. In the ideal case we are left with just one v-node (if B is satisfiable). But since y can take values other than the actually occuring probabilities, there might be more v-nodes left. We now show that the number of v-nodes can be at most $m+1$.

Let v_1, v_2, ..., v_r be the remaining v-nodes and let $\boldsymbol{q}_j \in \boldsymbol{Q}^m$ be the point associated with v_j and $Q = \{\boldsymbol{q}_1, \boldsymbol{q}_2, \ldots, \boldsymbol{q}_r\}$. Let furthermore $\boldsymbol{y}_j$ be a vector that satisfies the above set of inequalities for v_j.

We claim that Q is *affinely independent* in $\boldsymbol{Q}^m$, that is, the points in Q span a $r-1$ dimensional subspace of $\boldsymbol{Q}^m$ (see [Grü67] for a reference). Since there can be at most $m+1$ affinely independent points in $\boldsymbol{Q}^m$, it follows that $r \leq m+1$.

By Lemma 5.2 below, to prove that Q is affinely independent, it suffices to show that for any $S \subseteq Q$ there exists an affine plane that separates S from $Q-S$ (i.e., the points in S lie on one side of the plane whereas the points in $Q-S$ lie on the other). We can assume that $|S| \leq r/2$ (otherwise replace S by $Q-S$).

The affine plane can be defined as the set of points $\boldsymbol{x} \in \boldsymbol{Q}^m$ that fulfill the equation

$$\begin{aligned} \boldsymbol{h}_S \cdot \boldsymbol{x} &= 1 - \frac{1}{W+2}, \text{ where} \\ \boldsymbol{h}_S &= \sum_{q_j \in S} \boldsymbol{y}_j. \end{aligned}$$

For any point $\boldsymbol{q}_i \in S$ we have:

$$\boldsymbol{h} \cdot \boldsymbol{q}_i \;\geq\; \boldsymbol{y}_i \cdot \boldsymbol{q}_i \;\geq\; 1-\epsilon \;>\; 1-\frac{1}{W+2}.$$

For any point $\boldsymbol{q}_i \in Q - S$ we have:

$$\begin{aligned}
\boldsymbol{h} \cdot \boldsymbol{q}_i &= \sum_{q_j \in S} \boldsymbol{y}_j \cdot \boldsymbol{q}_i \\
&\leq \epsilon * |S| \\
&\leq \epsilon * \frac{r}{2} \\
&\leq \epsilon * k \\
&\leq \epsilon * (W+1) \\
&< \frac{W+1}{W+2} \\
&= 1 - \frac{1}{W+2}.
\end{aligned}$$

This proves our claim.

To see that our algorithm runs in polynomial time, we note that the above system of linear inequalities can be solved in polynomial-time using Khachian's algorithm [Kha79], and the probabilities on the edges at any stage can be represented using polynomially many bits. Now Lemma 5.2 below completes the proof. □

Lemma 5.2 *Let $Q \subseteq \boldsymbol{Q}^m$ such that for every $S \subseteq Q$ there is an affine plane $\boldsymbol{h}_S$ that separates S from $Q - S$. Then Q is affinely independent.*

Proof. Let $Q = \{\boldsymbol{q}_1, \ldots, \boldsymbol{q}_r\}$. Q is affinely dependent if there exist $\lambda_1, \ldots, \lambda_r$ that are *not* all zero, such that $\sum_{i=1}^r \lambda_i * \boldsymbol{q}_i = \boldsymbol{0}$ and $\sum_{i=1}^r \lambda_i = 0$.

We embed Q in $\boldsymbol{Q}^{m+1}$ by mapping $\boldsymbol{q}_i$ to $\boldsymbol{q}'_i = (\boldsymbol{q}_i, 1)$. Let Q' denote the embedding of Q. Then we have that Q is affinely dependent iff Q' *linearly* dependent.

Corresponding to each affine plane $\boldsymbol{h}_S$, we now have a *hyperplane* that separates points in S' from points in $Q' - S'$. Namely, the affine plane $\boldsymbol{h}_S \cdot \boldsymbol{x} = d$ becomes the hyperplane $(\boldsymbol{h}'_S, -d) \cdot \boldsymbol{x} = 0$.

Now, suppose that Q' is linearly dependent. This implies that there are $\lambda_1, \ldots, \lambda_r \in \boldsymbol{Q}$ such that not all of them are zero and

$$\sum_{i=1}^{r} \lambda_i * \boldsymbol{q}'_i \;=\; \boldsymbol{0}. \tag{2}$$

Let S' be the set of those $\boldsymbol{q}'_i$ such that $\lambda_i \geq 0$. By equation (2) we have

$$\begin{aligned}
0 &= \boldsymbol{h}'_S \cdot (\sum_{i=1}^{r} \lambda_i * \boldsymbol{q}'_i) &(3)\\
&= \sum_{i=1}^{r} \lambda_i * (\boldsymbol{h}'_S \cdot \boldsymbol{q}'_i) &(4)
\end{aligned}$$

Now observe that each term in equation (4) is non-negative and at least one term is non-zero. To see this note that if $\lambda_i \geq 0$ then $\boldsymbol{q}'_i \in S'$ and hence $\boldsymbol{h}'_S \cdot \boldsymbol{q}'_i > 0$. On the other hand, if $\lambda_i < 0$

then $\boldsymbol{q}'_i \in Q' - S'$ and hence $\boldsymbol{h}'_S \cdot \boldsymbol{q}'_i < 0$. Therefore this latter sum cannot be zero and we have a contradiction. We conclude that Q must be affinely independent. □

Note that we cannot push a BP-OBDD with too large error into the range of Theorem 5.1 by the standard amplification technique. This is because there we use the cross-product construction which increases the width of the resulting BP-OBDD. In the specific BP-OBDDs presented in Section 3 and Theorem 4.3 there is an alternative way of amplification: we could choose more primes. But again this increases the width of the resulting BP-OBDD. More precisely, consider the BP-OBDDs that verify multiplication or binary quadratics. For each prime p, the sub-OBDD that verifies the equation modulo p has width p^2. So if we verify equations modulo the first k primes $p_1, \ldots, p_k$, then the width of the resulting BP-OBDD is

$$W \quad = \quad \sum_{i=1}^{k} p_i^2 \quad \geq \quad k^3.$$

The error is $\Theta(n/k)$. This is somewhat larger than $1/W^{1/3}$. This shows that neither the above polynomial-time algorithm can be generalized to significantly larger error bounds, nor can the NP-completeness proof be generalized to significantly smaller errors, unless P = NP. It remains an open question to settle the satisfiability problem for errors in the range $O(\frac{1}{W^{1/3}}) \cap \omega(\frac{1}{W})$.

The equivalence problem for BP-OBDDs can be reduce to the satisfiability problem: let B_0 and B_1 be two BP-OBDDs, then $B_0 \not\equiv B_1$ iff $B = B_0 \oplus B_1$ is satisfiable. B can be constructed by Lemma 2.2. Therefore we also get an efficient algorithm for (the promise version of) the equivalence problem for BP-OBDDs of small error. Note that the width of B is bounded by the product of the widths of B_0 and B_1 and the errors sum up.

Corollary 5.3 *The equivalence problem for BP-OBDDs of width W with error ϵ is in* P*, provided that* $2\epsilon < 1/(W^2 + 2)$.

Acknowledgments

We would like to thank Somenath Biswas, Harry Buhrman, Lance Fortnow, Jochen Messner, Jacobo Toran, and Klaus Wagner for helpful discussions.

References

[Abl97] F. Ablayev. Randomization and nondeterminism are incomparable for ordered read-once branching programs. In *24rd International Colloquium on Automata Languages ans Programming*, Lecture Notes in Computer Science 1256, pages 195–202. Springer-Verlag, 1997.

[AK96] F. Ablayev and M. Karpinski. On the power of randomized branching programs. In *23rd International Colloquium on Automata Languages and Programming*, Lecture Notes in Computer Science 1099, pages 348–356. Springer-Verlag, 1996.

[AK97] F. Ablayev and M. Karpinski. Manuscript, 1997.

[BHR95] Y. Breitbart, H. Hunt, and D. Rosenkrantz. On the size of binary decision diagrams representing boolean functions. *Theoretical Computer Science*, 145:45–69, 1995.

[BRS93] A. Borodin, A. Razborov, and R. Smolensky. On lower bounds for read-k-times branching programs. *Computational Complexity*, 3:1–18, 1993.

[Bry86] R. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transaction on Computers*, 35(6):677–691, 1986.

[Bry91] R. Bryant. On the complexity of VLSI implementations and graph representation of boolean functions with applications to integer multiplication. *IEEE Transaction on Computers*, 40(2):205–213, 1991.

[Bry92] R. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992.

[BS90] R. Boppana and M. Sipser. The complexity of finite functions. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*. The MIT Press and Elsevier, 1990.

[BSSW94] B. Bollig, M. Sauerhoff, D. Sieling, and I. Wegener. On the power of different types of restricted branching programs. Technical Report TR94-026, Electronic Colloquium on Computational Complexity, http://www.eccc.uni-trier.de/eccc/, 1994.

[GM96] J. Gergov and C. Meinel. Mod-2-OBDDs - a data structure that generalizes EXOR-sum-of-products and odered binary decision diagrams. *Formal Methods in System Design*, 8:273–282, 1996.

[Grü67] B. Grünbaum. *Convex Polytopes*. Interscience Publishers (John Wiley & Sons), 1967.

[Juk89] S. Jukna. The effect of null-chains on the complexity of contact schemes. In *7th Fundamentals of Computation Theory*, Lecture Notes in Computer Science 380, pages 246–256. Springer-Verlag, 1989.

[Kha79] L. Khachian. A polynomial algorithm in linear programming. *Russian Academy of Sciences Doklady. Mathematics (formerly Soviet Mathematics–Doklady)*, 20:191–194, 1979.

[KMW91] M. Krause, C. Meinel, and S. Waack. Separating the eraser Turing machine classes L_e, NL_e, and P_e. *Theoretical Computer Science*, 86:267–275, 1991.

[MA78] K. Manders and L. Adleman. NP-complete decision problems for binary quadratics. *Journal of Computer and System Sciences*, 16:168–184, 1978.

[Mas76] W. Masek. A fast algorithm for the string editing problem and decision graphs complexity. Master's thesis, Massachusetts Institute of Technology, 1976.

[Pon95] S. Ponzio. *Restricted Branching Programs and Hardware Verification*. PhD thesis, Massachusetts Institute of Technology, 1995.

[Sau97] M. Sauerhoff. A lower bound for randomized read-k-times branching programs. Technical Report TR97-019, Electronic Colloquium on Computational Complexity, http://www.eccc.uni-trier.de/eccc/, 1997.

[Waa97] S. Waack. On the descriptive and algorithmic power of parity ordered binary decision diagrams. In *14th Symposium on Theoretical Aspects of Computer Science*, Lecture Notes in Computer Science 1200, pages 201–212. Springer-Verlag, 1997.

[Weg88] I. Wegener. On the complexity of branching programs and decision trees for clique functions. *Journal of the ACM*, 35:461–471, 1988.