Journal of Universal Computer Science, vol. 12, no. 6 (2006), 710-724
submitted: 31/5/06, accepted: 23/6/06, appeared: 28/6/06 © *J.UCS*

Testing Membership in Formal Languages Implicitly Represented by Boolean Functions

Beate Bollig
(Univ. Dortmund, Germany
beate.bollig@uni-dortmund.de)

Abstract: Combinatorial property testing, initiated formally by Goldreich, Goldwasser, and Ron in [Goldreich et al. (1998)] and inspired by Rubinfeld and Sudan in [Rubinfeld and Sudan 1996], deals with the relaxation of decision problems. Given a property P the aim is to decide whether a given input satisfies the property P or is *far* from having the property. A property P can be described as a language, i.e., a non-empty family of binary words. The associated property to a family of boolean functions $f = (f_n)$ is the set of 1-inputs of f. By an attempt to correlate the notion of testing to other notions of low complexity property testing has been considered in the context of formal languages. Here, a brief summary of results on testing properties defined by formal languages and by languages implicitly represented by small restricted branching programs is provided.

Key Words: binary decision diagrams (BDDs), boolean functions, branching programs (BPs), computational complexity, formal languages, property testing, randomness, sublinear algorithms
Category: F.1.3, F.2.2, G.3.1

1 Introduction

Property testing is a field in computational theory that deals with the information that can be deduced from the input, when the number of allowable queries (reads from the input) is significantly smaller than its size. Applications could be practical situations in which the input is so large that even taking linear time in its size to provide an answer is too much. Since any sublinear time algorithm can only view a small portion of the input for most problems the algorithm must give an answer which is in some sense approximate. Given a particular set, called property, and an input x one wants to decide whether x has the property or is *far* from it. Far usually means that many characters of the input have to be modified to obtain an element in the set. The definition of property testing is a relaxation of the standard definition of a decision problem in the sense that the tester is allowed arbitrary behavior when the object does not have the property and yet is close to an object having the property. In other words, property testing approaches approximation of decision problems by transforming languages into promise problems where certain inputs are excluded from consideration. Such an approximation might be used e.g. in applications where typical inputs are either good (have the property) or very bad (far from having the property) or to speed up a slow exact algorithm.

Property testing was first defined and applied by Rubinfeld and Sudan in the context of algebraic properties of functions [Rubinfeld and Sudan 1996]. They checked whether a given function computes a low-degree polynomial or is far from computing it. Inspired by this work the study of property testing for combinatorial objects was initiated in [Goldreich et al. (1998)]. These investigations were motivated by the notion of testing serving as a new notion of approximation and by some related questions that arise in computational learning theory. In [Goldreich et al. (1998)] the authors considered mainly graph properties. Since then property testing has become quite an active research area and has been applied to different classes of objects including graphs, hypergraphs, matrices, formal languages, boolean expressions, and point sets, see e.g., [Fischer 2001], [Ron 2001] for excellent surveys on the topic. A strong motivation for investigating property testing is that this area is abundant with fascinating combinatorial problems. One of the important questions is characterizing all properties that can be tested with a sublinear or even better with a constant number of queries into the input. Since a logical characterization of the properties testable with a constant number of queries is far from achieved, one goal is to identify whole classes of properties (instead of individual properties) that are testable and to formulate sufficient conditions for problems to be testable with a constant number of queries. One attempt is to correlate the traditional categorization of properties into computational complexity classes with the notion of testing. There are properties that are hard to decide but are easy to test with a constant number of queries into the input such as 3-colorability [Goldreich et al. (1998)] and properties that are easy to decide but are hard to test, e.g. in [Alon et al 2000] examples of NC^1 functions (functions representable by polynomial-size B_2-circuits of logarithmic depth) were presented that require $\Theta(n^{1/2})$ queries. By an attempt to correlate the notion of testing to other notions of low complexity property testing has been considered in the context of formal languages. In this survey we provide a brief summary of results on testing properties defined by formal languages and by languages implicitly represented by small restricted branching programs.

The rest of the paper is organized as follows. In Section 2 some preliminaries are given. Property testing is formally defined and representation models for boolean functions are introduced. In Section 3 some results concerning property testing in the context of formal languages are presented. Section 4 is devoted to results for properties described by boolean formulas in conjunctive forms of small size. The final Section 5 contains results for properties defined by branching programs of small size. To give the reader the opportunity to catch a glimpse of the lower bound techniques used in the property testing scenario, one result is presented in more detail.

2 Preliminaries

2.1 Property Testing

We introduce some basic definitions and notations and make the idea of property testing for formal languages a little bit more precise. Let P be a property, i.e., a non-empty family of binary words. A word w of length n is called ϵn-far from satisfying P if no word w' of the same length, which differs in at most ϵn places (Hamming distance), satisfies P. The distance parameter ϵ plays a role similar to that of an approximation factor in standard approximation algorithms. An ϵ-test for P is a randomized algorithm, which queries the quantity n, has the ability to make queries about the value of any desired bit of an input word w of length n, and distinguishes with probability $2/3$ between the case of $w \in P$ and the case of w being ϵn-far from satisfying P. (Obviously the choice of the success probability $2/3$ is not crucial since any constant strictly greater than $1/2$ is sufficient because of probability amplification.) Sometimes an ϵ-test is simply called a testing algorithm. An ϵ-test for P has one-sided error if it always accepts inputs that have the property P. An ϵ-test is non-adaptive if its queries do not depend on the answers to previous queries, otherwise the ϵ-test ist adaptive. In addition to possible practical significance, one-sided error and non-adaptivity are useful theoretical tools for obtaining hardness results. A property P is said to be $(\epsilon, q(\epsilon, n))$-testable if there is an ϵ-test that for every input x of size n queries at most $q(\epsilon, n)$ bits of the input string. If a property P is $(\epsilon, q(\epsilon, n))$-testable with $q = q(\epsilon)$ (i.e., q is a function of ϵ only, and is independent of n), P is said to be ϵ-testable. Finally, we say that property P is testable if P is ϵ-testable for every fixed $\epsilon > 0$. Otherwise P is non-testable. The query complexity of a testing algorithm is the minimal number of its queries to an input string.

Properties can be identified with the collection of their characteristic boolean functions, i.e., a property $P \subseteq \{0,1\}^*$ is identified with a family of boolean function $f = (f_n)$, where $f_n : \{0,1\}^n \to \{0,1\}$ so that $f_n(x) = 1$ iff $x \in P$ and $|x| = n$. Let $x, y \in \{0,1\}^n$. If $g : \{0,1\}^n \to \{0,1\}$ and g is not the constant function 0, we define $dist(x, g) = \min\{H(x, y) | y \in g^{-1}(1)\}|$, where $H(x, y)$ is the Hamming distance. An input x is ϵn-close to a function g iff $dist(x, g) \leq \epsilon n$. Otherwise the input x is ϵn-far.

A property is linear if it forms a vector space.

2.2 Representations for Boolean Functions

2.2.1 Branching Programs and Binary Decision Diagrams

Branching programs (BPs), in applications also called binary decision diagrams (BDDs), are considered as a nonuniform model of computation in complexity theory and as a data structure for boolean functions in several applications,

in particular, in verification and for CAD problems. (See [Wegener 2000] for a survey.)

A *branching program* (BP) on the variable set $X_n = \{x_1, \ldots, x_n\}$ is a directed acyclic graph with one source and two sinks labeled by the constants 0 and 1. Each non-sink node (or decision node) is labeled by a boolean variable and has two outgoing edges, one labeled by 0 and the other by 1.

An input $a \in \{0,1\}^n$ activates all edges consistent with a, i.e., the edges labeled by a_i which leave nodes labeled by x_i. The computation path for an input a in a BP G is the path of edges activated by a which leads from the source to a sink. A computation path for an input a is called accepting if it reaches the 1-sink.

Let B_n denote the set of all boolean functions $f : \{0,1\}^n \to \{0,1\}$. The BP G represents the function $f \in B_n$ for which $f(a) = 1$ iff the computation path for the input a is accepting. The size of a branching program G is the number of its nodes. The branching program size of a boolean function f is the size of the smallest BP representing f. The length of a branching program is the maximum length of a path.

A branching program is called (syntactically) *read k times* (BPk) if each variable is tested on each path at most k times.

A branching program is called *s-oblivious*, for a sequence of variables $s = (s_1, \ldots, s_l)$, $s_i \in X_n$, if the set of decision nodes can be partitioned into disjoint sets V_i, $1 \leq i \leq l$, such that all nodes from V_i are labeled by s_i and the edges which leave V_i-nodes reach only V_{i+1}-nodes. The level i, $1 \leq i \leq l$, contains the V_i-nodes. The level $l+1$ contains the 0- and the 1-sink. If the sequence of variables is unimportant we call an s-oblivious branching program an oblivious branching program. An oblivious branching program is of width w if its largest level contains w nodes.

An oblivious read k times branching program for a sequence $s = (s_1, \ldots, s_{kn})$ is called k-IBDD if s can be partitioned into k subsequences $(s_{(i-1)n+1}, \ldots, s_{in})$, $1 \leq i \leq k$, for which $\{s_{(i-1)n+1}, \ldots, s_{in}\} = X_n$.

2.2.2 Conjunctive Forms

A boolean formula is in conjunctive form (CF) if it is a conjunction of clauses, where every clause is a disjunction of literals. (A literal is a boolean variable or a negated boolean variable.) The size of a CF is the number of its clauses. If all clauses contain at most k literals, the formula is a kCF. A boolean function f is said to have $O(1)$ size 0-witnesses if it can be represented by a kCF, where $k = O(1)$.

3 Testing Membership in Formal Languages

Motivated by the desire to understand in what sense the complexity of testing properties of strings is related to the complexity of formal languages, Alon et al [Alon et al 2000] showed that membership in any regular language is testable, hence obtaining a general result identifying a non-trivial class of properties each being testable. The presented algorithm tries to find evidence to an input string not belonging to the given language in the form of so-called infeasible subwords. Every word w that is $\epsilon|w|$-far from the given language contains many such short infeasible subwords. The query complexity of the algorithm is dependent on the size of the (smallest) finite automaton accepting the given language, but this size is a fixed constant with respect to the length of the input word. The authors also discussed the testability of more complex languages and proved that the query complexity required for testing context-free languages cannot be bounded by any function that only depends on the distance parameter ϵ. One important subclass of the context-free languages are the Dyck languages which includes strings of properly balanced parantheses. Formally, the Dyck language D_m contains all balanced strings that contain at most m types of parantheses. In [Alon et al 2000] it was shown that membership in D_1 can be tested with a number of queries that only depends on the distance parameter ϵ, whereas membership in D_2 cannot be tested with less than a logarithmic number of queries with respect to the input length. A lower bound of $\Omega(n^{1/11}/\log n)$ on the query complexity of any algorithm for testing D_m, $m > 1$, was shown in [Parnas et al. 2003]. Moreover, an algorithm with query complexity $O(n^{2/3}\log^3(n/\epsilon)\epsilon^{-3})$ was presented testing whether an input string belongs to D_m.

It is still not known whether there exists a sublinear testing algorithm for all context free languages.

In [Chockler and Kupferman 2004] the result of Alon et al was extended to ω-regular languages and so-called lasso-shaped words. A reduction from the ϵ-test of infinite words to a constant number of ϵ-tests for finite words was described.

4 Testing CF Properties

Testing a property represented by a CF can be viewed as testing whether a given assignment to boolean variables satisfies the given conjunctive form or is far from any satisfying assignment. For a long time all properties known to be hard for two-sided error testing were functions whose 0-witnesses were large, e.g., the linear lower bound of Bogdanov, Obata, and Trevisan [Bogdanov et al. 2002] capitalizes on the existence of inputs that are far from having the property, yet any local view of a constant fraction of them can be extended to an input having the property. If the property is defined by a kCF, $k = O(1)$, this cannot happen.

For each input that does not satisfy the property, there exists a set of k queries that witnesses the fact that the input does not have the property.

In [Fischer et al. 2002] the authors described some properties that are equivalent to properties representable by formulas in 2CF with respect to the number of queries required for testing. The equivalent properties include being a vertex cover in a graph fixed in advance, beeing a clique in a graph fixed in advance and monotonicity of graph labelings over the binary alphabet. These proofs of equivalence are so-called gap-preserving local reductions that tranform instances of one problem into another, such that the testing algorithms for the second problems could be used to solve the first one (see also [Bogdanov et al. 2002]). Therefore, the reductions transform positive instances into positive instances and ϵ-instances into instances that are reasonably far from having the property, i.e., the distance is preserved and is changed by at most a constant factor. Each query for the new problem should be computable by a constant number of queries for the original problem. These requirements ensure that a (one-sided error) testing algorithm for the new problem yields a (one-sided error) testing algorithm for the original problem with asymptotically the same query complexity. Furthermore, in [Fischer et al. 2002] a one-sided error testing algorithm for monotonicity of labelings of graphs with n nodes with query complexity $O((n/\epsilon)^{1/2})$ was presented.

Testing properties representable by kCFs becomes hard for $k \geq 3$, i.e., there cannot exist sublinear testing algorithm since Ben-Sasson, Harsha, and Raskhodnikova [Ben-Sasson et al. 2005] showed the existence of linear properties representable by 3CFs that require a linear number of queries. First, they demonstrated that every adaptive two-sided error test for checking membership in a vector space can be converted to a non-adaptive one-sided error test with the same query complexity and essentially identical parameters. Afterwards, the authors presented sufficient conditions for a vector space to be hard to test, and proved that random (c, d)-regular Low Density Parity Check Codes (LDPC codes) satisfy these conditions (see [Gallager 1963] for the definition of these codes). In the last step, they have demonstrated how to convert the resultant codes to vector spaces representable by boolean formulas in 3CF. To obtain the results the following techniques have been used:

- Yao's minimax principle [Yao 1977] providing a general technique for proving lower bounds for randomized algorithms;
- expansion in random graphs (and applications to random LDPC codes);
- linear algebra.

Fischer, Newman, and Sgall [Fischer et al. 2004] proved that there exists a property with $O(1)$ size 0-witnesses, and therefore with boolean formulas in kCF,

$k = O(1)$, that can be represented by a width 3 oblivious read-twice branching program but for which a $5/8 \cdot 10^{-7}$-test requires $\Omega(n^{1/10})$ queries. Like the result in [Ben-Sasson et al. 2005] the existence of the property involves a probabilistic argument and the proof is not constructive.

5 Testing Properties Representable by Small Size Restricted Branching Programs

In [Bollig and Wegener 1998] it was shown that some monotone functions representable by boolean formulas in disjunctive form with prime implicants of length 2 need size $2^{\Omega(n^{1/2})}$ if they are represented by read-once branching programs. It is not difficult to modify this function to obtain a function with $2^{\Omega(n^{1/2})}$ read-once branching program size that is representable by boolean formulas in 2CFs. On the other hand the parity function can only be represented by a boolean formula in conjunctive form with 2^{n-1} clauses but the OBDD size is $2n+1$. Therefore, the classes of boolean functions representable by kCFs and by oblivious branching programs of polynomial size are incomparable.

Newman [Newman 2002] extended the result described in [Alon et al 2000] asserting that regular languages are testable by considering oblivious read-once branching programs of constant width as a non-uniform counterpart of the notion of a finite automaton. He proved the following result. If $g = (g_n)$ is a family of boolean functions representable by OBDDs of width w then for every n and $\epsilon > 0$ there is a randomized algorithm that always accepts every $x \in \{0,1\}^n$ if $g_n(x) = 1$ and rejects it with high probability if at least ϵn bits of x should be modified to obtain some $x' \in g_n^{-1}(1)$. His algorithm makes $(2^w/\epsilon)^{O(w)}$ queries. Therefore, for constant ϵ and w the query complexity is $O(1)$.

Fischer, Newman, and Sgall [Fischer et al. 2004] put a bound on the branching program complexity of boolean functions that still guarantees testability by presenting a property identified with an oblivious read-twice branching program of width 3 for which at least $\Omega(n^{1/3})$ queries are necessary. Furthermore, Newman's result can be generalized to nonoblivious read-once branching programs of constant width if on all paths from the source to the sinks all variables are tested exactly once.

Bollig and Wegener [Bollig and Wegener 2003] showed that functions representable by read-once branching programs of quadratic size are not necessarily testable. Nevertheless, the presented lower bound on the query complexity is very small for the investigated boolean function. In [Bollig 2005b] an improved lower bound of $\Omega(n^{1/2})$ was presented. As a corollary a boolean function was provided representable by (oblivious) read-once branching programs of almost linear size but not testable with a constant number of queries. Until now it is an open question whether there exists a sublinear testing algorithm for all functions representable by read-once branching programs of polynomial size.

The linear lower bound on the query complexity in [Ben-Sasson et al. 2005] was proven for a function representable by boolean formulas in 3CFs where each literal appears in at most d clauses, $d = O(1)$. Therefore, using standard techniques it is not difficult to prove the existence of a boolean function representable by 2IBDDs of width 3 that need query complexity $\Omega(n)$. Like the other results in [Ben-Sasson et al. 2005] the existence of the property involves a probabilistic argument and the lower bound proof is not constructive.

To give the reader the opportunity to catch a glimpse of the lower bound techniques used in the property testing scenario, in the following a smaller lower bound is presented for an explicitly defined boolean function. One key idea are gap-preserving local reductions already mentioned in Section 4.

Our aim is to construct a property identified with a family of boolean functions $g = (g_n)$ that can be represented by 2IBDDs of constant width and by CFs of linear size, where even most of the clauses have constant length, but for which any ϵ-test requires n^δ queries for some $0 < \epsilon < 1$ and $0 < \delta < 1$. First, we reinvestigate a function $f_n : \{0,1,2\}^{n^2} \to \{0,1\}$ already considered by Fischer and Newman [Fischer and Newman 2001]. Here, we use a different proof to present a larger lower bound on the query complexity. Afterwards we consider a boolean encoding of that function. Already Fischer and Newman encoded the $\{0,1,2\}$-labeling to present a non-testable $\forall\exists$-poset property. Since our aim is to construct a boolean function that can be represented by 2IBDDs of small size and by CFs of linear size, we have to use a different approach.

For the rest of this section we consider 2-dimensional matrices and use the notion of rows and columns in the usual matrix sense. The property *symmetric permutation* uses the alphabet $\{0,1,2\}$. We say that a matrix satisfies the property *symmetric permutation* if it is a row/column permutation of a symmetric matrix with all 2's on its primary diagonal, and no 2's anywhere else. Obviously, this requirement is equivalent to the following two conditions:

1. In every row and in every column there exists exactly one 2-entry.
2. The matrix contains none of the following 2×2 matrices as a submatrix (to ensure that the original matrix was symmetric):
$\begin{pmatrix} 2 & 0 \\ 1 & 2 \end{pmatrix}, \begin{pmatrix} 2 & 1 \\ 0 & 2 \end{pmatrix}, \begin{pmatrix} 0 & 2 \\ 2 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 2 \\ 2 & 0 \end{pmatrix}$.

The property *symmetric permutation* is identified with its characteristic function $f = (f_n)$, where f_n is defined on $n \times n$ matrices M on the variables m_{ij}, $1 \le i, j \le n$.

Proposition 1. *Let U be the uniform distribution on all boolean $n \times n$ matrices. For each $\epsilon \le 1/4 - \delta$, δ a constant, the probability that an instance generated according to U is ϵn^2-close to f_n is $o(1)$.*

Proof. Let $m := (n^2 - n)/2$. A pair $(m_{ij}, m_{i',j'})$, $i \neq i'$ and $j \neq j'$, is called a σ-pair for a permutation $\sigma \in S_n$ if $\sigma(i) = j'$ and $\sigma(i') = j$. If an instance M generated according to U is ϵn^2-close to f_n, then there exists a permutation $\sigma \in S_n$ such that at most ϵn^2 σ-pairs $(m_{ij}, m_{i',j'})$ differ, i.e., $m_{ij} = 1 - m_{i',j'}$. If we bound this probability for each σ by $2^{-\Omega(n^2)}$, then the probability that there exists some permutation with this property is bounded above by $2^{n \log n} \cdot 2^{-\Omega(n^2)} = 2^{-\Omega(n^2)}$. For a fixed permutation σ the number of σ-pairs $(m_{ij}, m_{i'j'})$ where $m_{ij} \neq m_{i'j'}$ is binomially distributed for the parameters m and $1/2$. Hence, by Chernoff bounds, the probability of at most $(1/2 - \alpha)m$ indices where $m_{ij} \neq m_{i'j'}$ is bounded by $2^{-\Omega(m)} = 2^{-\Omega(n^2)}$ for each constant $\alpha > 0$. Hence, we are done if $\epsilon n^2 \leq (1/2 - \alpha)m$. Since $n^2 = (2 + o(1))m$ and $\epsilon \leq 1/4 - \delta$ we get

$$\epsilon n^2 \leq (1/4 - \delta)(2 + o(1))m \leq (1/2 - \alpha)m$$

for some appropriate $\alpha > 0$ and m large enough.

Proposition 2. *Let S be a set of $d = o(n)$ variables from $\{m_{11}, \ldots, m_{nn}\}$ and M be an instance generated according to P, the uniform distribution on $f_n^{-1}(1)$. Let σ_M be the corresponding permutation to M. Let $A(S)$ be the event that S contains no σ_M-pair and no variable m_{ij} for which $\sigma_M(i) = j$. Then* $\mathrm{Prob}_P(A(S)) \geq 1 - o(1)$.

Proof. The permutation σ_M is chosen according to the uniform distribution. The d variables in S lead to at most $(1/2)d(d-1)$ different pairs for at most $(1/2)d(d-1) \cdot (n-2)!$ permutations and to at most d variables m_{ij} for at most $(n-1)!$ permutations σ each, where $\sigma(i) = j$. Therefore, the probability that S contains a σ_M-pair or a variable m_{ij}, where $\sigma_M(i) = j$, is at most $(1/2)d(d-1) \cdot (n-2)!/n! + d(n-1)!/n!$ which is $o(1)$ if $d = o(n)$ and n large enough. Hence, the proposition follows.

Theorem 3. *For any ϵ-test for f_n, $\epsilon \leq 1/4 - \delta$ and δ a constant, $o(n)$ queries are insufficient, where n^2 is the number of variables of f_n.*

Proof. To prove the lower bound, we apply Yao's *minimax* principle [Yao 1977] which says that to show a lower bound on the complexity of a randomized test, it is enough to present an input distribution on which any deterministic test with that complexity is likely to fail. Therefore, we define two distributions, P for the positive instances and N for the negative instances, and show that in order to distinguish between these distributions $o(n)$ queries are insufficient.

P is the uniform distribution on $f_n^{-1}(1)$ and can be realized as follows. First, a symmetric matrix M' with 2's on the primary diagonal and 0's and 1's everywhere else is uniformly chosen. The input matrix M is constructed from M' by permuting its rows according to a permutation which is chosen uniformly in

random. N is the uniform distribution on the set of all boolean $n \times n$ matrices that are ϵn^2-far from f_n. Moreover, let U be the uniform distribution on all boolean $n \times n$ matrices. The probability distribution D over all inputs is now defined by choosing with probability $1/2$ positive instances according to P and with probability $1/2$ negative instances according to N.

We prove that even an adaptive deterministic algorithm that queries at most $d = o(n)$ bits has an error probability of more than $1/6$ on inputs taken from the distribution D. By probability amplification we can conclude that every adaptive deterministic algorithm that queries at most $d/9 = o(n)$ bits must have an error probability of more than $1/3$.

Let $\mathcal{T}$ be an adaptive algorithm that queries at most d bits. Every leaf in the decision tree that represents $\mathcal{T}$ is labeled by either accept or reject. We may assume that $\mathcal{T}$ queries each input bit at most once for a randomly chosen input according to D and that the decision tree is a complete ternary tree of depth d, because we can transform each decision tree in such a tree without increasing the error probability. Let L be the set of all leaves that are labeled by reject. Let $B(L)$ be the event to reach a leaf from the set L. We assume that $\mathrm{Prob}_N(B(L)) \geq 2/3$ as otherwise the algorithm errs on inputs which are ϵn^2-far from f_n with probability of more than $1/6$. Our aim is to prove that using the assumption it follows that $\mathrm{Prob}_P(B(L)) \geq (1 - o(1))2/3 > 1/3$ which implies that the algorithm errs by rejecting positive inputs with probability of more than $1/6$.

Every leaf $\alpha \in L$ corresponds to a set of variables S_α that were queried along the way to α and an assignment b_{S_α} to these variables. The algorithm reaches for an input w the leaf α if the assignment to the variables in S_α is consistent with b_{S_α}. The assignment to variables that are queried in other branches of the decision tree is meaningless. We may assume that for every $\alpha \in L$ the assignment b_{S_α} to the variables in S_α contains only 0's and 1's as otherwise the leaf α may be changed to accept without reducing the success probability of $\mathcal{T}$ under D.

Let $B(\alpha)$ be the event that the algorithm reaches the leaf α, $A(S_\alpha)$ the event as defined in Proposition 2, and M_ϵ the event that a uniformly chosen boolean $n \times n$ matrix is ϵn^2-far from f_n. The distribution N equals the uniform distribution U on all inputs with the restriction that the input is ϵn^2-far from f_n. Using Proposition 1 we obtain $\mathrm{Prob}_U(M_\epsilon) \geq 1 - o(1)$. Moreover, there exist 2^d leaves ℓ, where b_{S_ℓ} contains only 0's and 1's. A boolean matrix can only reach one of these leaves and under the uniform distribution each of these leaves is reached with the same probability. Therefore,

$$\begin{aligned}\mathrm{Prob}_N(B(\alpha)) &= \mathrm{Prob}_U(B(\alpha) \mid M_\epsilon) = \mathrm{Prob}_U(B(\alpha) \cap M_\epsilon)/\mathrm{Prob}_U(M_\epsilon) \\ &\leq \mathrm{Prob}_U(B(\alpha))/\mathrm{Prob}_U(M_\epsilon) \\ &\leq (1 + o(1))2^{-d}.\end{aligned}$$

Furthermore, using Proposition 2 we know that

$$\begin{aligned}\mathrm{Prob}_P(B(\alpha)) &\geq \mathrm{Prob}_P(B(\alpha) \cap A(S_\alpha)) = \mathrm{Prob}_P(B(\alpha) \mid A(S_\alpha)) \cdot \mathrm{Prob}_P(A(S_\alpha)) \\ &\geq (1 - o(1))2^{-d}.\end{aligned}$$

Now we can conclude that

$$\begin{aligned}\mathrm{Prob}_P(B(L)) &= \sum_{\alpha \in L} \mathrm{Prob}_P(B(\alpha)) \geq (1 - o(1)) \sum_{\alpha \in L} \mathrm{Prob}_N(B(\alpha)) \\ &= (1 - o(1))\mathrm{Prob}_N(B(L)) \geq (1 - o(1))2/3 \\ &> 1/3\end{aligned}$$

if n is large enough. This completes the proof of Theorem 3.

The function $f_n^{A,B} : \{0,1\}^{2n^2} \to \{0,1\}$ is a boolean encoding of the function f_n and is defined on two $n \times n$ boolean matrices A and B on the variables a_{ij} and b_{ij}, $1 \leq i, j \leq n$. The function $f_n^{A,B}$ outputs 1 iff the following conditions are fulfilled:

i) A is a permutation matrix, i.e., there exists exactly one 1-entry in each row and one 1-entry in each column.

ii) If a_{ij} and $a_{i'j'}$ are equal to 1 then $b_{ij'}$ is equal to $b_{i'j}$.

Proposition 4. *If there exists an ϵ-test with $q(\epsilon, 2n^2)$ queries for $f_n^{A,B}$ then there exists a 2ϵ-test with the same number of queries for f_n.*

Proof. For brevity we denote an ϵ-test with $q(\epsilon, n)$ queries by $(\epsilon, q(\epsilon, n))$-test in the following. Assume that there exists an $(\epsilon, q(\epsilon, 2n^2))$-test T for $f_n^{A,B}$. Our aim is to construct a $(2\epsilon, q(\epsilon, 2n^2))$-test for f_n. For this reason we define a mapping p from inputs M for f_n to inputs (A, B) for $f_n^{A,B}$. For every $M = (m_{ij})_{1 \leq i,j \leq n} \in \{0,1,2\}^{n^2}$ let $p(M)$ be defined as follows:

- $a_{ij} := m_{ij}(m_{ij} - 1)/2$,
- $b_{ij} := m_{ij}(m_{ij} - 1)/2 - (m_{ij} - 2)m_{ij}$.

If $m_{ij} = 0$ then $a_{ij} = b_{ij} = 0$, if $m_{ij} = 1$ then $a_{ij} = 0$ and $b_{ij} = 1$, and if $m_{ij} = 2$ then $a_{ij} = 1$ and $b_{ij} = 1$. Obviously, $M \in f_n^{-1}(1)$ implies $f_n^{A,B}(p(M)) = 1$. In order to obtain an input from $f_n^{-1}(1)$ there have to be at most as many bit positions in M to be changed as bit positions in $p(M)$ in order to get a 1-input from $f_n^{A,B}$. Therefore, if $dist(p(M), f_n^{A,B}) \leq \epsilon(2n^2)$ then $dist(M, f_n) \leq 2\epsilon n^2$.

To $(2\epsilon, q(\epsilon, 2n^2))$-test f_n on an input M, we perform the $(\epsilon, q(\epsilon, 2n^2))$-test T on $p(M)$. The only difference is that each time that a_{ij} (b_{ij}) is queried for $p(M)$ we just query m_{ij} and compute a_{ij} (b_{ij}). Since the reduction is distance preserving we can inherit the result of T without changing the error probability on 1-inputs and inputs that are $2\epsilon n^2$-far from f_n.

Now we transform the function $f_n^{A,B}$ to a boolean function g_n that can be represented by 2IBDDs of constant width and by CFs of linear size, where most of the clauses have constant length. The idea for the construction of g_n is the following one. We use the same number of copies for the variables in the matrices A and B. For every original variable a_{ij} (b_{ij}) we generate $(n-1)^2$ new variables $a_{ij}^{i'j'}$ ($b_{ij}^{i'j'}$), where $i \neq i'$ and $j \neq j'$. Then we add for each a_{ij} (b_{ij}) two new variables a_{ij}^r and a_{ij}^c (b_{ij}^r and b_{ij}^c).

The function g_n outputs 1 iff the following conditions are fulfilled:

a) For $1 \leq i, j \leq n$, all variables $a_{ij}^{i'j'}$, $i \neq i'$ and $j \neq j'$, a_{ij}^r, and a_{ij}^c have the same value. The same holds for the b_{ij}-variables.

b) For each row i, $1 \leq i \leq n$, there exists exactly one variable a_{ij}^r, $1 \leq j \leq n$, set to 1.

c) For each column j, $1 \leq j \leq n$, there exists exactly one variable a_{ij}^c, $1 \leq i \leq n$, set to 1.

d) If $a_{ij}^{i'j'} = a_{i'j'}^{ij} = 1$ then $b_{ij'}^{i'j}$ and $b_{i'j}^{ij'}$ are equal.

Altogether the function g_n is defined on $N := 2n^2((n-1)^2 + 2)$ variables.

Proposition 5. *If there exists an $(\epsilon, q(\epsilon, N))$-test for g_n then there exists an $(\epsilon, q(\epsilon, N))$-test for $f_n^{A,B}$.*

The proof of Proposition 5 is similar to the proof of Proposition 4. The key idea is another gap-preserving local reduction.

Proposition 6. *The function g_n can be represented by* 2IBDDs *of width 3 and by CFs of linear size.*

Proof. In the first part of the 2IBDD we verify the requirements b), c), and d) one after another. Since the requirements are defined on different sets of variables we obtain an oblivious read-once branching program for these requirements by glueing the oblivious read-once branching programs together, i.e., the 1-sink of one branching program is replaced by the source of the next one and so on. The requirements b) and c) can be verified by an oblivous read-once branching program of width 3. The requirement d) can be checked for each group of four variables $a_{ij}^{i'j'}$, $a_{i'j'}^{ij}$, $b_{ij'}^{i'j}$, and $b_{i'j}^{ij'}$ by an oblivious read-once branching program of width 3 realising the function

$$\neg((a_{ij}^{i'j'} \wedge a_{i'j'}^{ij}) \wedge ((b_{ij'}^{i'j} \wedge \neg b_{i'j}^{ij'}) \vee (\neg b_{ij'}^{i'j} \wedge b_{i'j}^{ij'}))).$$

In the second part of the oblivious 2IBDD we just check the requirement a), i.e., if all copies of the same variable of the original function $f_n^{A,B}$ have the same

value. All copies of the same variable are tested one after another. Width 3 is sufficient.

The resulting CF is a conjunction of CFs checking the requirements a)-d) separately. The requirement that for each row i (column j) there exists exactly one variable a^r_{ij}, $1 \leq j \leq n$, (a^c_{ij} , $1 \leq i \leq n$) set to 1 is equivalent to the requirement that there exists for each row i (column j) at least one variable set to 1 and that there are not two variables in this row (column) set to 1. Hence, we obtain

$$(a_{i1} \vee a_{i2} \vee \cdots \vee a_{in}) \bigwedge_{1 \leq j_1 < j_2 \leq n} (\neg a_{ij_1} \vee \neg a_{ij_2}).$$

Altogether there are $2n \cdot n(n-1)/2 = n^3 - n^2$ clauses of length 2 and $2n$ clauses of length n to check the requirements b) and c). The requirement d) can be tested for each group of variables by a 4CF with 2 clauses

$$(\neg a^{i'j'}_{ij} \vee \neg a^{ij}_{i'j'} \vee \neg b^{i'j}_{ij'} \vee b^{ij'}_{i'j}) \wedge (\neg a^{i'j'}_{ij} \vee \neg a^{ij}_{i'j'} \vee b^{i'j}_{ij'} \vee \neg b^{ij'}_{i'j}).$$

Altogether there are $n^2 \cdot 2(n-1)^2$ clauses of length 4 to verify condition d). Finally, the test whether some variables have the same value can be done by pairwise checking that two of them are equal or in a more clever way by checking whether the first one is equal to the second one, the second one to the third one and so on. Therefore, the requirement a) can be tested by a 2CF with $2n^2 \cdot 2((n-1)^2+1)$ clauses. Summarizing, the function g_n has a CF representation with $4n^4 - 7n^3 + 7n^2$ clauses of length 2, $2n^4 - 4n^3 + 2n^2$ clauses of length 4 and $2n$ clauses of length n, where $N = \Theta(n^4)$ is the number of variables.

Combining Theorem 3 and Propositions 3-5 we have proved the following main result.

Theorem 7. *The function g_n can be represented by* 2IBDDs *of width 3 and by CFs of linear size but for any ϵ-test, $\epsilon \leq 1/8 - \delta'$, δ' a constant, $o(N^{1/4})$ queries are insufficient, where N is the number of variables of g_n.*

6 Concluding Remarks

An investigation that goes beyond the original definition of testable properties was initiated in [Parnas et al. 2004], concerning tolerant testers that must also accept inputs that are close enough to satify the given property. To be more precise, an input has to be accepted with high probability if at most an ϵ_1 fraction, of the input characters has to be modified to make it satisfy the property and it has to be rejected with high probability if at least an ϵ_2 fraction, $0 \leq \epsilon_1 < \epsilon_2 < 1$, of the input characters need modification. Recently, in [Fischer and Fortnow 2005] it was shown that not all testable properties are also tolerantly testable. Until

now it is open whether every testable boolean property admits a tolerant test with a sublinear number of queries.

In most of the cases in the literature, the distance used in the definition of being ϵn-far from satisfying a property is the Hamming distance between input strings. It may be interesting to consider other distance measures and to compare not only inputs of the same length to compute the distance of an input to a given property. The edit distance between two words is the minimal number of insertions, deletions, and substitutions of an input letter required to transform one word into the other. The edit distance with moves considers one additional operation, subwords can be moved to another position of the input word. In [Magniez and de Rougemont 2004, Fischer et al. 2006] property testing in the context of formal languages was investigated concerning the edit measure with moves.

References

[Alon et al 2000] Alon, N., Krivelevich, M., Newman, I., and Szegedy, M.: "Regular languages are testable with a constant number of queries"; SIAM Journal on Computing 30 (2000), 1842-1862.

[Blum et al. 1993] Blum, M., Luby, M., and Rubinfeld, R.: "Self-testing/correcting with applications to numerical problems"; Journal of Computer and System Sciences 47 (1993), 549-595.

[Ben-Sasson et al. 2005] Ben-Sasson, E., Harsha, P., and Raskhodnikova, S.: "Some 3CNF properties are hard to test"; SIAM Journal on Computing 35(1) (2005), 1-21.

[Bogdanov et al. 2002] Bogdanov, A., Obata, K., and Trevisan, L.: "A lower bound for testing 3-colorability in bounded-degree graphs"; Proc. of 43rd FOCS (2002), 93-102.

[Bollig 2005a] Bollig, B.: "A large lower bound on the query complexity of a simple boolean function"; Information Processing Letters 95 (2005), 423-428.

[Bollig 2005b] Bollig, B.: "Property Testing and the branching program size of boolean functions"; Proc. of FCT, Lecture Notes in Computer Science 3623, Springer (2005), 258-269.

[Bollig and Wegener 1998] Bollig, B. and Wegener, I.: "A very simple function that requires exponential size read-once branching programs"; Information Processing Letters 66 (1998), 53-57.

[Bollig and Wegener 2003] Bollig, B. and Wegener, I.: "Functions that have read-once branching programs of quadratic size are not necessarily testable"; Information Processing Letters 87 (2003), 25-29.

[Chockler and Kupferman 2004] Chockler, H. and Kupferman, O.: "ω-Regular languages are testable with a constant number of queries"; Theoretical Computer Science 329 (2004), 71-92.

[Fischer 2001] Fischer, E.: "The art of uninformed decisions: a primer to property testing"; Bulletin of the European Association for Theoretical Computer Science 75 (Oct. 2001), 97-126.

[Fischer and Fortnow 2005] Fischer, E. and Fortnow, L.: "Tolerant versus intolerant testing of boolean properties"; Proc. of 20th CCC (2005), 135-140.

[Fischer et al. 2002] Fischer, E., Lehman, E., Newman, I., Reskhodnikova, S., Rubinfeld, R., and Samorodnitsky, A.: "Monotonicity testing over general poset domains"; Proc. of 34th STOC (2002), 73-79.

[Fischer et al. 2006] Fischer, E., Magniez, F., and de Rougemont, M.: "Approximate satisfiability and equivalence"; to appear in Proc. of 21st Symposium on Logic in Computer Science (2006).

[Fischer and Newman 2001] Fischer, E. and Newman, I.: "Testing of matrix properties"; Proc. of 33rd STOC (2001), 286-295.

[Fischer et al. 2004] Fischer, E., Newman, I., and Sgall, J.: "Functions that have read-twice constant width branching programs are not necessarily testable"; Random Structures and Algorithms 24(2) (2004), 175-193.

[Gallager 1963] Gallager, R. G.: *Low Density Parity Check Codes*, MIT Press, Cambridge, (1963).

[Goldreich et al. (1998)] Goldreich, O., Goldwasser, S., and Ron, D.: "Property testing and its connection to learning and approximation"; Journal of the ACM 45 (1998), 653–750.

[Magniez and de Rougemont 2004] Magniez, F. and de Rougemont M.: "Property testing of regular languages"; Proc. of ICALP (2004), 932-944.

[Newman 2002] Newman, I.: "Testing membership in languages that have small width branching programs"; SIAM Journal of Computing 31(5) (2002), 1557-1570.

[Parnas et al. 2003] Parnas, M., Ron, D., and Rubinfeld, R.: "Testing membership in parenthesis languages"; Random Struct. Algorithms 22(1) (2003), 98-138.

[Parnas et al. 2004] Parnas, M., Ron, D., and Rubinfeld, R.: "Tolerant property testing and distance approximation"; ECCC Report TR04-010 (2004).

[Ron 2001] Ron, D.: "Property testing (a tutorial)"; *Handbook of Randomized Computing*, Rajasekaran, S., Pardalos, P.M., Reif, J.H., and Rolim, J.D., eds., Kluwer Academic Publishers (2001), 597-649.

[Rubinfeld and Sudan 1996] Rubinfeld, R. and Sudan, M.: "Robust characterization of polynomials with applications to program testing"; SIAM Journal of Computing 25(2) (1996), 252-271.

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

[Yao 1977] Yao, A. C.: "Probabilistic computation, towards a unified measure of complexity"; Proc. of 18th FOCS (1977), 222-227.