# Testing of Function that have small width Branching Programs

Ilan Newman
Department of Computer Science, University of Haifa, Haifa, Israel
E-mail: ilan@cs.haifa.ac.il

## Abstract

*Combinatorial property testing, initiated formally by Goldreich, Goldwasser and Ron in [11], and inspired by Rubinfeld and Sudan [13], deals with the following relaxation of decision problems: Given a fixed property and an input $x$, one wants to decide whether $x$ has the property or is being 'far' from having the property.*

*The main result here is that if $\mathcal{G} = \{g : \{0,1\}^n \rightarrow \{0,1\}\}$ is a family of Boolean functions that have read-once branching programs 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 $g(x) = 1$, and rejects it with hight probability if at least $\epsilon n$ bits of $x$ should be modified in order for it to be in $g^{-1}(1)$. The algorithm queries $(\frac{2^w}{\epsilon})^{O(w)}$ many queries. In particular, for constant $\epsilon$ and $w$, the query complexity is $O(1)$.*

*This generalizes the results of Alon et. al. [2] asserting that regular languages are efficiently $(\epsilon, O(1))$-testable.*

## 1. Introduction

Combinatorial property testing, initiated formally by Goldreich, Goldwasser and Ron in [11], and inspired by Rubinfeld and Sudan [13], deals with the following relaxation of decision problems: Given a fixed property and an input $x$, one wants to decide whether $x$ has the property or is being 'far' from having the property. A property here is a set of binary strings (these inputs that have the 'property') and is being equated with its characteristic function (that is '1' on all inputs that have the property and '0' elsewhere). Being 'far' is measured by the number of bits that needs to be changed for an input $x$ in order for it to have the property (i.e- the hamming distance). A property is said to be $(\epsilon, q)$-testable if there is a randomized algorithm that for every input $x \in \{0,1\}^n$ queries at most $q$ bits of $x$ and with probability $2/3$ distinguishes between the case that $x$ has the property and the case that $x$ is $\epsilon n$-far from having the property. When a property $P$ is $(\epsilon, q)$-testable with $q = q(\epsilon)$ (i.e $q$ is a function of $\epsilon$ only, and is independent of $n$) then we say that $P$ is $\epsilon$-testable. Sometimes, algorithms that don't err on inputs that have the property are considered (one-sided error). In fact, here as well as in [11, 2] and most applications, the algorithms have 1-sided error.

Apart of being a natural relaxation of the standard decision problem, combinatorial property testing emerges naturally in the context of PAC learning, program checking [9, 6, 13], probabilistically checkable proofs [3] and approximation algorithms [11].

In [11], the authors mainly consider graph properties and show (among other things) the quite surprising fact that testing bipartiteness can be done by randomly testing a polynomial in $1/\epsilon$ number of edges of the graph, answering the above question with constant probability of failure. They also raise the question of obtaining general results identifying classes of properties that are $\epsilon$-testable, for every constant $\epsilon > 0$. Some interesting examples are given in [11], several additional ones can be obtained by applying the Regularity Lemma [1]. Alon et. al. [2], proved that any regular language is $(\epsilon, O(1))$-testable for $\epsilon = \Omega(1)$, hence obtaining a general result identifying a non trivial class of properties that are $\epsilon$-testable. Here we further pursue this direction: We prove that if a language has (non-uniform) read-once branching program of width $w$ then it is $(\epsilon, (\frac{2^w}{\epsilon})^{O(w)})$-testable. In particular this shows that every family of function that has (non-uniform) constant width, read-once branching program is $(\epsilon, O(1))$-testable for $\epsilon = \Omega(1)$. This identifies many other non trivial families of Boolean functions that are $\epsilon$-testable. In particular this generalizes and gives an alternative proof (and algorithm) for the result of [2], as regular languages have constant width read-once branching programs.

A Branching Program (BP) of width $w$ is a deterministic **leveled** BP in which every level contains at most $w$ vertices. In the sequel we will be interested in BP of width $w$ that have the further restriction of being **read-once**, namely every level is associated with a variable (all nodes in a level query the same variable) and each variable appears in at most one level. Branching Programs have been extensively studied as a model of computation for Boolean functions ( [7] contains a survey text, see also [4, 5, 10] for a very partial list of different aspects involving BP and read-once BP).

The size of BP (and read-once BP) is tightly connected to the space complexity of the function it computes: If a language is in $SPACE(s)$ and time $t$ then it has BP of width $2^{O(s)}$ (and length $t$), and also read-once branching program of width $2^{O(2^s)}$. However, the inverse of the last assertion is not true (even for computable languages). The result of [2] and the result here, in its uniform manifestation, may be viewed as asserting that 'very small' space functions are efficiently testable: All regular languages are in $SPACE(O(1))$. They also have read-once BP of $O(1)$ size. What happens for $SPACE(\omega(1))$-functions? It is known that $SPACE(O(1)) = SPACE(o(\log\log n))$=Regular. Hence the above question is interesting for $SPACE(s)$ with $s = \Omega(\log\log n)$. The result here says nothing directly for $s = \Omega(\log\log n)$. However, we get rid of the strong 'uniformity' of the DFA's used in [2]. Thus, in particular, we show that functions that have small width, read-once BP, are $(\epsilon, f(n))$-testable for some small growing function $f(n)$, although they might require $\omega(\log\log n)$-space. This include trivially the family of $O(1)$-terms DNF, $O(1)$-clauses CNF and some other interesting examples (see section 4).

We end up this discussion by noting that certainly $SPACE(O(\log n))$ functions are not efficiently testable in general; ([2, 11] contain lower bounds for such functions). However, the situation of $SPACE(s)$ for $\log\log n \leq s << \log n$ is wide open. In particular we don't have any candidate for a $SPACE(O(\log\log n))$ function that its $\epsilon$-testing requires $n^{\Omega(1)}$ many queries for every constant $\epsilon > 0$.

## 2. Definitions and Notations

A leveled branching program (BP) is a directed graph with one source $s$, two sinks; $A$ (accept) and $R$ (reject). Each internal node has two outgoing edges one of which is labeled by '0' and the other by '1'. Nodes are partitioned to consecutive 'levels' so that edges that go out from a level point to vertices in the next level. In addition, levels are labeled by members of $[n]$ (Boolean variables). Hence each input $x \in \{0,1\}^n$ naturally defines a path from $s$ to either $A$ or $R$ (by following at each step, the edge with the label identical to the current variable's value). A leveled BP defines naturally a Boolean function $g : \{0,1\}^n \mapsto \{0,1\}$ in the following way: $g(x) = 1$ if the path that $x$ defines reaches $A$.

A *read-once* BP for $g : \{0,1\}^n \mapsto \{0,1\}$ is a leveled BP computing $g$ with the additional property that distinct levels are labeled with distinct variables. We number the levels of the BP from 0 (containing the source $s$) and on. We may assume that the last level (containing $A$ and $R$) is numbered by $n$. A leveled BP is said to have width $w$ if the its largest level contains $w$ nodes.

From now and on we consider only read-once BP's.

For a given BP, $B$, computing a function $g : \{0,1\}^n \mapsto \{0,1\}$, and two nodes $u, v$, we define $B[u : v]$ the (sub) branching program for which its start node is $u$ and its accept node is $v$. Obviously $B[u : v]$ computes a Boolean function on at most $n$ variables.

Let $x, y \in \{0,1\}^n$, we define $dist(x,y) = hamming(x,y) = |\{i|\ x_i \neq y_i\}|$. Let $g : \{0,1\}^n \mapsto \{0,1\}$ we define $dist(x,g) = min\{dist(x,y)|\ y \in g^{-1}(1)\}$. For two nodes $u, v$ in levels $i, j$ respectively, let $dist(x, B[u:v]) = dist(x[i,j], g')$ where $g'$ is the function computed by $B[u : v]$ on input $x[i,j] = x_i, x_{i+1}, ..., x_j$.

We will be interested in BP's for which the start and accept node are not always defined. However, the reject node $R$ will be defined. Given such BP, if all paths from $v$ end in the reject node, we say that $v$ is an $R$-**node**. Let $p < q$ be two distinct levels in a branching program $B$. We denote by $B[p : q]$ the sub BP of $B$ containing all nodes in levels $p, p+1, ..., q$. $B[p : q]$ has undefined source and sink, but has the $R$-nodes of $B$. For each $0 \leq l \leq n$ we denote by $V(l)$ the set of all non $R$-nodes in level $l$. We may assume that every level contains at most one $R$-node as different $R$-nodes in the same level may be identified.

For an integer $q$ and a node $v$ in a BP, we denote by

$$A_q(v) = \{u|\text{ there is a path of length } q \text{ from u to v}\}$$

See Figure 1.

## 3. Results

**Theorem 1** *Let $g : \{0,1\}^n \mapsto \{0,1\}$ be computed by a read-once BP, $B$ of width $w$. Then testing $g$ requires at most $(\frac{2^w}{\epsilon})^{O(w)}$.*

**Corollary 3.1** *If $g : \{0,1\}^n \mapsto \{0,1\}$ has a read-once BP of width $w = O(1)$ then $g$ is $(\epsilon, O(1))$-testable for every $\epsilon = \Omega(1)$.*

The proof of Theorem 1 uses several reduction steps in order to reduce testing of a $w$-width BP to testing of $(w-1)$-width BP's. We start by analyzing the following simple case which will also be the basis for our recursion process.

**Proposition 3.1** *If $g : \{0,1\}^n \mapsto \{0,1\}$ is computable by a BP of width $w = 2$, then $g$ is $(\epsilon, O(\frac{1}{\epsilon}))$-testable.*

**Proof:** Let $B$ be a BP of width $w = 2$, computing the function $g$ and let $x$ be such that $dist(g,x) \geq \epsilon n$ (assuming that there is such $x$). We look at the path $x$ defines in $B$, and every time it reaches an $R$-node, flip the corresponding bit of $x$ and mark this place. Note that if at level $i$ there is an $R$-node, then every following level $l > i$ must contain an $R$-node too (as an R-node can only point to another R-node). Obviously, there must be at least $\epsilon n$ marked places. On the other hand for $x$ for which $g(x) = 1$ there are obviously no marked places at all. Hence sampling $O(\frac{1}{\epsilon})$ bits of an input $x$ and rejecting if a level is found in which the corresponding bit leads from the (unique) non $R$-node to an $R$-node is guaranteed to succeed with probability $2/3$ for $\epsilon n$-far inputs (and with probability 1 for other inputs). ■.

### 3.1. The general case - Proof of the main Theorem

Before we prove Theorem 1 we want to indicate the overall description of the algorithm and try to outline the intuition behind the definitions.

The algorithm for $\epsilon$-testing will be recursive on the width $w$. The main phase of the algorithm (for a restricted sort of branching program) is the following: We look at $p = O(\frac{1}{\epsilon^2})$ levels that are equally spaced in $B$. Our aim is to test for each sequence of vertices ($s = v_0, v_1, ..., v_p, v_{p+1} = accept$) taken from these $p$ levels, whether the input $x$ might pass from $v_i$ to $v_{i+1}$ on its way from $s$ to *accept*. Clearly, for any $x \in \{0,1\}^n$, $g(x) = 1$ if and only if there is such sequence for which $dist(x, B[v_i : v_{i+1}]) = 0$ for every $i$. On the other hand if $dist(x, g) \geq \epsilon n$ then clearly, for every such sequence there is an $i$ for which $dist(x, B[v_i : v_{i+1}]) = \Omega(\epsilon^3 \cdot n)$. Hence, our test will be as follows: For each such sequence $s$ we try to find an $i$ so that $dist(x, B[v_i : v_{i+1}]) = \Omega(\epsilon^3 \cdot n)$. If we succeed, we answer 'no'. Otherwise, (i.e if for a certain sequence $s$ there is no such $i$) we answer 'yes'.

The key point will be to show that, under some restrictions, testing these shorter BP's, $B[v_i : v_{i+1}]$, may be reduced to testing *narrower* BP's which will lead to the recursive algorithm yielding the final result.

A key notion are that of *q-full* levels and decomposable BP's that are defined below.

**Definition 1** *Let $v$ be a non $R$-vertex in level $l$ of a branching program. We say that $v$ is $q$-**full** if $A_q(v)$ contains all non $R$-nodes of level $l - q$. If every non $R$-vertex in level $l$ is q-full, then we say that level $l$ is q-full.*

Namely, a vertex $v$ is $q$-full if $v$ is reachable from every non R-vertex of level $l - q$, See Figure 1.

**Proposition 3.2** *If $v$ is q-full for a certain q, then it is $q'$-full for every $q' > q$.* ■

**Definition 2** *Let $\delta < 1$, $n' \leq n$. We say that a BP, $B$ is $(\delta, n')$-**decomposable** if it contains a level $\ell < n'$ that is q-full for $q \leq \frac{\delta \ell}{10}$.*

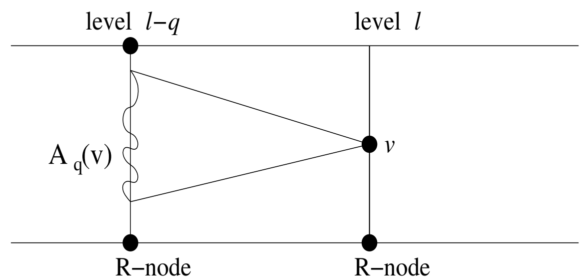


**Figure 1.** $A_q(v)$ is the set of nodes in level $l - q$ that can reach $v$. Here $v$ is **not** $q$-full as not all non R-nodes in level $l - q$ can reach $v$.

We first show that if $B$ is not $(\epsilon, n')$-decomposable (for some appropriate $n'$) then testing $B$ can indeed be reduced to testing 'narrower' BP's as described above. Then we show how a general BP, $B$, can be decomposed into disjoint sub programs, so that each is not decomposable and so that testing $B$ can be reduced to testing one of the non decomposable parts of it.

We first start with our main Lemma showing how to test non-decomposable BP's.

**Lemma 3.1** *Let $n' \geq n - \frac{\epsilon n}{40}$, $\delta \leq \epsilon$ and let $g : \{0,1\}^n \mapsto \{0,1\}$ be computed by a non $(\delta, n')$-decomposable BP, $B$ of width $w$. Then $\epsilon$-testing $g$ requires at most $q_1(\epsilon, w) \leq \frac{w^4}{\delta^3} q(0.8\epsilon, w-1)$, where $q(\epsilon, w)$ is the query complexity of $\epsilon$-testing BP's of width $w$.*

**Proof:** The idea of the proof is the following: We fix $O(\frac{1}{\delta^2})$ levels that are equally spaced in $B$, leaving out enough space in the beginning and in the end of $B$. The assumption that $B$ is not $(\delta, n')$-decomposable will imply that each of these levels is not $(2m)$-full where $m$ is the distance between two consecutive levels. This, in turn, will imply that testing the subprogram in between two levels may be reduced to testing a 'narrower' BP.

Formally: Let $m = \frac{\delta^2 n}{400}$. Let $\{l_0, ..., l_p\}$ be the set of levels that are $m$ apart, starting from level $\frac{20m}{\delta}$ and ending before level $n'$. Namely, $l_i = \frac{20m}{\delta} + i \cdot m,\ i = 0, 1, ..., p = \frac{n' - l_0}{m} = O(\frac{1}{\delta^2})$. Let $\mathcal{S} = V(l_0) \times ... \times V(l_p)$. Each $\bar{v} = (v_0, ..., v_p) \in \mathcal{S}$ defines a sequence of possible 'stopovers' in which an input $x \in g^{-1}(1)$ might pass in its way from the start vertex to $A$. The following Proposition relates testing of $B$ to testing of sub programs of it that are defined by $\mathcal{S}$:

**Proposition 3.3**

1. Let $x \in g^{-1}(1)$ then for some $\bar{v} \in \mathcal{S}$, $dist(x, B[v_i : v_{i+1}]) = 0$ for $i = 0, ..., p-1$. ■

2. Let $dist(x,g) \geq \epsilon n$ then for each $\bar{v} \in \mathcal{S}$, $\Sigma_{i=0}^{i=p-1} dist(x, B[v_i : v_{i+1}]) \geq \epsilon n - l_0 - (n - l_p) \geq 0.9\epsilon n$. ■

Our intent in the rest of the proof is to show that testing $B[v_i : v_{i+1}]$ can be reduced to testing narrower BP's.

By our choice of parameters and the fact that $B$ is not $(\delta, n')$-decomposable we have:

**Fact 3.1** *Let $l$ be a level such that $l_0 \leq l$, then level $l$ is not $(2m)$-full* ■

For each level $l$ such that $l_i < l \leq l_{i+1}$, let $F(l)$ be the set that contains each $(l - l_i)$-full vertex $v$ in level $l$. In other words, $v \in F(l)$ if it is in level $l$ and it is reachable from every non-R vertex of level $l_i$. By our assumption on $B$, $F(l) \neq V(l)$ as otherwise $l$ would be $(l - l_{i-1})$-full in contradiction with Fact 3.1.

Hence the above implies the following Claim:

**Claim 3.1** *Let $u, v$ be non-R vertices in levels $l_i, l_{i+1}$ respectively and let $l$ be a level such that $l_i \leq l \leq l_{i+1}$.*

- *Let $u'$ be in level $l$ and assume that $u' \notin F(l)$ then $B[u : u']$ is of width $w' \leq w - 1$.*
- *Let $v'$ be in level $l$ and assume that $v' \in F(l)$ then $B[v' : v]$ is of width $w' \leq w - 1$.*

**Proof:** Let $u' \notin F(l)$ be in level $l$. As $u' \notin F(l)$, $u'$ is not $(l - l_i)$-full, then, by Proposition 3.2, it is also not $(l - l')$-full for every $l' > l_i$. Namely, for every intermediate level $l_i < l' < l$ there are (non R) vertices that cannot reach $u'$ and hence can be taken as $R$-nodes in $B[u; u']$. This proves the first part.

Assume now that $v'$ is in level $l > l_i$ and $v' \in F(l)$. Let $t$ be any node at level $l' > l$ that is reachable from $v'$. As $v' \in F(l)$ it follows that $t$ is $(l' - l_i)$-full. Hence not all vertices in level $l'$ can be reachable for $v'$ as otherwise level $l'$ will be $(l' - l_i)$-full. As this is true for every $l < l' \leq l_{i+1}$ this implies that $B[v' : v]$ is of width $w' \leq w - 1$. If $v$ is in level $l = l_i$ then the same argument for $t$ will work except that $t$ will be $(l' - l_{i-1})$-full. Again this implies that level $l'$, such that $l_i < l' \leq l_{i+1}$ cannot have all its non R-nodes reachable from $v'$ as otherwise it would be $(l' - l_{i-1})$-full (which is a contradiction, as no level is $2m$-full). ■

Claim 3.1 asserts that $B[v_i : v_{i+1}]$ is indeed of width of at most $(w - 1)$, unless $v_i \notin F(l_i)$ and $v_{i+1} \in F(l_{i+1})$. We still need to deal with the case for which $v_i \notin F(l_i)$ and $v_{i+1} \in F(l_{i+1})$ which might be of width $w$. The key observation here is that any path from $v_i$ to $v_{i+1}$ must start at $V(l_i) - F(l_i)$ (as $v_i$ is such), and end in $F(l_{i+1})$. Hence this path must intersect $F(l)$ for some intermediate level $l_i < l \leq l_{i+1}$. In addition, (by Proposition 3.2) once it intersects $F(l)$ it intersects $F(l')$ for every $l' > l$ (see Figure 2).

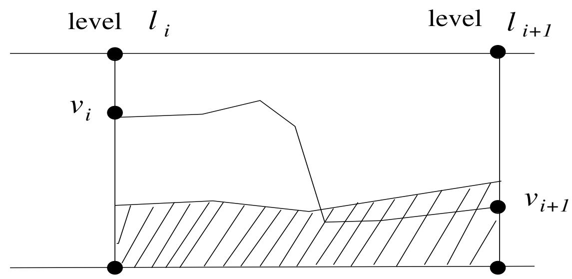


**Figure 2.** Vertices in shadowed area are in $F()$. If path from $v_i$ to $v_{i+1}$ intersects $F(l)$ at some intermediate level $l$. Then it intersects $F()$ for every following level.

Hence to test $B[u : v]$ for $u \in V(l_i) - F(l_i)$ and $v \in F(l_{i+1})$ we do the following: Let $k = \frac{10}{\delta}$. We chose in the range $\{l_i, ..., l_{i+1}\}$ $k+1$ levels, $p_0, ..., p_k$, that are $\frac{m}{k}$ apart: $p_j = l_i + j \cdot \frac{m}{k}$, $j = 0, ..., k$

**Claim 3.2** *Let $u \in V(l_i) - F(l_i)$ and $v \in F(l_{i+1})$*

- *Let $x \in \{0,1\}^n$ s.t $dist(x, B[u, v]) = 0$, then for some $j \in \{1, ..., k\}$ there are some $u' \in V(p_{j-1}) - F(p_{j-1})$, $v' \in F(p_j)$ so that $dist(x, B[u : u']) = dist(x, B[v' : v]) = 0$* ■
- *Let $x \in \{0,1\}^n$ s.t $dist(x, B[u, v]) \geq 0.9\epsilon m$, then for every $j \in \{1, ..., k\}$ and for every $u' \in V(p_{j-1}) - F(p_{j-1})$ and $v' \in F(p_j)$,*

$$dist(x, B[u : u']) + dist(x, B[v' : v]) \geq 0.9\epsilon m - \frac{m}{k} \geq 0.8\epsilon m$$ ■

Now we formally end the proof of Lemma 3.1 by presenting a $\epsilon$-testing algorithm that reduces the testing of a BP, $B$ of width $w$ that is not $(\delta, n')$-decomposable to $(0.8\epsilon)$-testing of narrower BP's:

> **Algorithm** $A(\epsilon, w)$:
> Let $B$ be a $w$-width BP that computes $g : \{0,1\}^n \mapsto \{0,1\}$ and so that $B$ is not $(\delta, n')$-decomposable for some $n' \geq n - \frac{\epsilon n}{40}$, $\delta \leq \epsilon$. Let $m, \mathcal{S}$, as above.
>
> For all $\bar{v} \in \mathcal{S}$, for all $i = 0, ..., p-1$ perform $(0.9\epsilon)$-test of $B[v_i : v_{i+1}]$ by calling algorithm $B(0.9\epsilon, w, B[v_i : v_{i+1}])$ below.
>
> If there is such $\bar{v}$ so that for every $i = 0, ...p-1$ the test on $B[v_i : v_{i+1}]$ passes - answer $g(x) = 1$.
> Otherwise, if for every $\bar{v}$ there is a $i \in \{0, ..., p-1\}$ for which the test of $B[v_i : v_{i+1}]$ fails, then answer $g(x) = 0$.

Testing $B[v_i : v_{i+1}]$ is done as follows:

**Algorithm** $B(0.9\epsilon, w, B[v_i : v_{i+1}])$:

1. If $v_i \in F(l_i)$ or $v_{i+1} \notin F(l_{i+1})$ then by Claim 3.1, $B[v_i : v_{i+1}]$ is already of width $w' \leq w - 1$. This test is done by calling the general $\epsilon$-testing procedure for width $(w-1)$ BP's.

2. Otherwise, if $v_i \notin F(l_i)$ and $v_{i+1} \in F(l_{i+1})$ let $k = \frac{10}{\delta}$. We chose in the range $\{l_i, ..., l_{i+1}\}$ $k+1$ levels, $p_0, ..., p_k$, that are $\frac{m}{k}$ apart: $p_j = l_i + j \cdot \frac{m}{k},\ j = 0, ..., k$

   For $j = 1, ..., k$ and every $u' \in V(p_{j-1}) - F(p_{j-1})$, $v' \in F(p_j)$, a $0.8\epsilon$-test is performed twice on $B[v_i : u']$ and $B[v' : v_{i+1}]$, by calling the general procedure for $\epsilon$-testing width $(w-1)$ BP's. For each of these two tests, a 'pass' is considered if at least one of the two repetitions passes.

   If for some $j$, $u' \in V(p_{j-1}) - F(p_{j-1})$ and $v' \in F(p_j)$, both test pass, algorithm $B$ returns 'pass' for the test on $B[v_i : v_{i+1}]$. Otherwise, if for every $j$, $u' \in V(p_{j-1}) - F(p_{j-1})$ and $v' \in F(p_j)$ at least one of the two tests fails algorithm $B$ returns 'fail' for the test on $B[v_i : v_{i+1}]$.

**Claim 3.3** *For $x$ such that $g(x) = 1$, algorithm $A$ ends up with the right answer with probability 1. For $x$ such that $dist(g, x) \geq \epsilon n$ algorithm $A$ ends up with the right answer with error probability bounded by 1/3*

**Proof:** The proof follows directly from the description, Proposition 3.3, Fact 3.1 and Claims 3.1, 3.2. Repeating the test on $B[v_i : u']$ and $B[v' : v_{i+1}]$ twice in item 2 of algorithm $B()$, guarantees error probability $\leq 1/9$ for the result on each. This, in turn, implies that the error probability on the event "for $j, u', v'$ the test pass but the result is wrong" id bounded by $2/9 < 1/3$ as required. We avoid further details here. ■

Let analyze the overall query complexity of algorithm $A$: There are $p = \frac{n - l_0}{m} = O(\frac{1}{\delta^2})$ levels that are $m$-apart, and we need to test for every legitimate $\bar{v} \in \mathcal{S}$, $B[v_i : v_{i+1}]$. There are $O(pw^2) = O(\frac{w^2}{\delta^2})$ such BP's. Each of these may be further divided into $k = O(\frac{1}{\delta})$ levels in algorithm $B$, for which we test twice $B[u : u'], B[v' : v]$ for possibly all pairs $(u', v') \in V(p_i) \times V(p_{i+1})$. Hence there are a total of $O(\frac{w^2}{\delta^2})O(\frac{w^2}{k}) = O(\frac{w^4}{\delta^3})$ such tests. Each of these tests is a call to an $0.8\epsilon$-test of a $(w-1)$-width BP, hence we get: $q_1(\epsilon, w) \leq O(\frac{w^4}{\delta^3})q(0.8\epsilon, w - 1)$.

This completes the proof of Lemma 3.1. ■

We note that if $B$ contains $w$ nodes at its start level and at its end level, then with the above complexity one can actually test $B[u, v]$ for any start and end nodes $u, v$ that belong to the start and end levels respectively.

In order to test the general case what remains now is to reduce testing of decomposable BP's into non-decomposable ones. This will follow from the following:

**Proposition 3.4** *Let $u$ $t_1, t_2$ be non $R$-vertices and assume that $t_1, t_2$ are $q$-full vertices on the same level. Then for every $x \in \{0,1\}^n$*

$$|dist(x, B[u : t_2]) - dist(x, B[u : t_1])| \leq q$$

.

**Proof:** Assume that $t_1$ and $t_2$ are at level $t$. We may assume that $u$ is at level $l$ so that $l < t - q$ for otherwise the proposition is trivial. Then, the closest $x'$ to $x$ that traverses $B$ from $u$ to $t_1$ must intersect $A_q(t_2)$. Hence by changing only the $q$ last bits of $x'$ we get an $x''$ that traverses $B$ from $u$ to $t_2$. ■

**Definition 3** *Let $x \in \{0,1\}^n$ and $p < q$ two levels, we define:*

$$dist(x, B[p : q]) = \min\{dist(x, B[u : v])|\ u \in V(p), v \in V(q)\}$$

**Claim 3.4** *Assume that there is a sequence of levels $l_0 = 1, ..., l_k = n$ and a sequence of numbers $r_1, ..., r_k$ so that the level $l_i$ is $r_i$-full for each $l_i \in \{l_1, ..., l_k\}$. Then for every $x \in \{0,1\}^n$, $\Sigma_i dist(x, B[l_i : l_{i+1}]) \geq dist(x, g) - \Sigma_i r_i$*

**Proof:** Let $x \in g^{-1}(1)$ then certainly $dist(x, B[l_i : l_{i+1}]) = 0$ for every $i = 1, ..., k-1$. On the other hand, for every $x$ for which $dist(x, g) \geq d$, $\Sigma_1^k dist(x, B[l_i : l_{i+1}]) \geq d - \Sigma_i r_i$ by Proposition 3.4 (for otherwise, a word $x'$ close to $x$ can be 'patched up' from the individual $k$ 'close' subwords, with an additional pay of $\Sigma r_i$ changes in order to patch each consecutive pair). ■

We are ready now to prove Theorem 1.

**Proof:** Let $B$ be a BP of width $w$ that computes $g$. Let $a_0 = 0$, let $a_1 = l_1$ be the first level which is $r_1$-full for $r_1 \leq \frac{\epsilon l_1}{20}$. Let $l_2$ be the smallest for which level $a_2 = a_1 + l_2$ is $r_2$-full for $r_2 \leq \frac{\epsilon l_2}{20}$ etc. This defines a sequence of BP's (with start and end that are undefined), $B_1, ..., B_k$, $B_i = B[a_{i-1} : a_i]$ of length $l_i$.

Let $\epsilon_1 = 0.5\epsilon$. Testing $B$ is done by sampling $i \in [k]$ proportionally to the length $l_i$, and $\epsilon_1$-testing $B[u, v]$ for all pairs $(u, v)$ for which $u$ is in levels $a_{i-1}$ and $v$ in level $a_i$. If any of these tests pass then the answer is 'yes', otherwise (if all tests fails) we answer 'no'.

By our choice, for every $i$, either $l_i = O(1)$ or $B_i$ does not contain an $q$-full level $l$ with $q \leq \frac{\epsilon_1 l_i}{10}$ (except for the very last level) and hence it is not $(\epsilon_1, l_i - 1)$-decomposable. Hence every $B_i$ can be trivially tested (in the former case), or tested by using algorithm $A$ as asserted by Lemma 3.1.

Let us check the error probability of this algorithm: If $x \in g^{-1}(1)$ then for every $i$, $dist(x, B[u,v]) = 0$ for some $u, v$. Hence the answer will be yes with probability 1.

For $x$ s.t $dist(x,g) \geq \epsilon n$, Claim 3.4 implies that $dist(x, B[a_i : a_{i+1}]) \geq \epsilon n - \Sigma_i r_i$. However, as $r_i \leq \frac{\epsilon \cdot l_i}{20}$ we conclude that $\Sigma r_i \leq \frac{\epsilon n}{20}$ and hence $\Sigma dist(x, B[a_i : a_{i+1}]) \geq \frac{95}{100}\epsilon n$

Thus by sampling as above we get that with probability at least 0.4 we land on a $B_i = B[a_i : a_{i+1}]$ for which $dist(x, B[a_i : a_{i+1}]) \geq \frac{1}{2}\epsilon l_i = \epsilon_1 l_i$ (by simple application of Markov inequality). For this $i$ and for every $u \in V(a_{i-1}), v \in V(a_i)$ we have that $dist(x, B[u : v]) \geq \epsilon_1 l_i$. Thus, for a wrong answer, the test will have to fail for each of the pairs. On each pair the test will succeed with probability of at least $0.4\frac{2}{3}$. Repeating this test 4 independent times (or just once if there are at least 4 pairs) will reduce the error probability to below $\frac{1}{3}$ again.

Now, every individual test is of a non $\epsilon_1$-decomposable $BP$ hence by Lemma 3.1 (with $\delta = \epsilon_1$) we get that in order to test $B$ we need $w^2$ many $(0.5\epsilon)$-tests of non $(0.5\epsilon)$-decomposable BP's (for each start and and end point). However, the complexity of testing a non-decomposable BP already accounts for testing each start and end pairs (as we test for every $B[v_i : v_{i+1}]$). Therefore the total complexity is

$$q(\epsilon, w) \leq \frac{8w^4}{\epsilon^3} q(0.4\epsilon, w-1)$$

which implies that $q(\epsilon, w) = (\frac{2^w}{\epsilon})^{O(w)}$ $\blacksquare$.

Finally, we end up with a note on the total running time of the algorithm. In the setting of $\epsilon$ testing this is not of prime importance, but still might be of interest. Time complexity here could be considered in two different models: The first in which the BP as well as the input $x$ are considered as inputs for the algorithm. The other, is to use preprocessing of the fixed BP, $B$ which is independent of the input, and then use it for every input $x$ of length $n$. We will analyze here the first and more restrictive model (which clearly bounds the complexity of the other).

To analyze the time complexity when the BP, $B$ is part of the input, we note that the complexity is contributed by following factors: The query complexity, decomposing $B$ into non-decomposable parts, computing $F(l)$ for every level $l$ (which is needed by algorithm $B()$) and computing the outcome for algorithm $A()$ and $B()$ form the answers they get back from lower recursion level calls.

we may start the algorithm (before actually recursing on a specific input $x$) by first computing for all vertices $u, v$ whether $v$ is reachable from $u$. This is done just once and takes $O(n^2)$ steps.

At each recursion level, decomposing the branching program can be done trivially in $O(w^2 n)$ steps. Computing $F(l)$ for each $l$ is done in time bound $O(w^2 n)$. Now the time for algorithm $A$, $t_A(0.5\epsilon, w, n)$ (as we call $A$ with $0.5\epsilon$) is contributed by $\frac{w^2}{0.25\epsilon^2}$ calls to algorithm $B()$ plus computing from these $\frac{w^2}{0.25\epsilon^2}$ answers the outcome of $A()$. Computing this outcome is done by checking whether there is a sequence $\bar{v} = (v_0, ..., v_p) \in V(l_0) \times ... \times V(l_p)$ for which all the answers of algorithm $B()$ on $B[v_i : v_{i+1}]$ are 'yes'. This can be trivially reduced to a connectivity problem on a graph containing every node in the levels $l_0, ..., l_p$ and with an edge $(v_i, v_{i+1})$ for each pair for which the test on $B[v_i : v_{i+1}]$ is 'yes'. Hence this is done in $O(\frac{w^2}{\epsilon^2})$ steps (which is the number of edges in this graph). This yields $t_A(0.5\epsilon, w, n) = O(\frac{w^2}{\epsilon^2} + \frac{w^2}{\epsilon^2} t_B(0.45\epsilon, w, n))$ where $t_B()$ is the time of algorithm $B$. By the same reasoning we get that $t_B(0.45\epsilon, w, n) = O(\frac{w^2}{\epsilon} + \frac{w^2}{\epsilon} t(0.4\epsilon, w-1, n))$, where $t()$ is the time for testing a general BP (when the data of the first step is already given).

Hence we get a recurrence $t(\epsilon, w, n) = O(w^2 n + \frac{w^2}{\epsilon^2} \cdot \frac{w^2}{\epsilon} t(0.4\epsilon, w-1, n)$ which yields $t(\epsilon, w, n) = (\frac{2^w}{\epsilon})^{O(w)} n$. Adding the $O(n^2)$ contribution of the first step and realizing that the algorithm is reasonable only for $w = o(\log \log n)$ we get a total of $O(n^2)$ running time for the whole algorithm.

In the case that the BP, $B$ is given in advance, and hence a preprocessing is allowed, one can show that the running time will be bounded by $(\frac{2^w}{\epsilon})^{O(w)} + O(\log n)$. We don't elaborate further on this point here.

## 4. Examples of some interesting functions and open problems

We present here some examples of functions that are efficiently testable and have narrow width, read-once BP's (Some times, a direct efficient testing algorithm is obvious). The first non trivial such family is of all regular languages, with a direct testing algorithm by [2]. We remark here, that for this case, our algorithm gives a conceptually different algorithm then of [2]. The dependence of the query complexity on $w$ in this case is similar to what would come out of [2]. The dependence on $\epsilon$ is worse.

Other very simple families are $k$-term-DNF, and $k$-clauses-CNF for small $k$ (each having $k+1$ width read-once BP). A function $g : \{0,1\}^n \mapsto \{0,1\}$ is $k$-term-DNF if it has a DNF representation (a disjunction of terms, each is a conjunction of literals) with at most $k$ terms. Analogously, a $k$-clause CNF is define. Clearly, these languages are efficiently tested (as long as $k$ is small enough) as union (intersection) of small number of efficiently testable languages is efficiently testable ([11]). It is interesting to note that these includes examples of functions that are, say, in uniform $SPACE(\log\log n)$ but not regular and hence not in $SPACE(o(\log\log n))$. On the other hand, since they have a constant width read-once BP (in fact 1-term DNF), they can be $\epsilon$-tested.

One such interesting example is the following example of Papadimitriou [12]: Let $b$ a binary string without leading 0's. We denote by $n(b)$ the natural number whose binary representation is $b$. Let $L = \{b_1\$b_2\$...\$b_k|\ n(b_i) = i\}$. Clearly $L \in SPACE(\log\log n)$. It is also not hard to see that $L$ is not regular. However, as $L$ contains at most one word of each length, it obviously has a BP of width $w = 2$. (Note, that although we have here an alphabet of size 3, we may actually encode everything in binary by encoding each symbol with two bits).

Another example is of the following language that is not in $SPACE(\log\log n)$, but has 'narrow' BP's and hence efficiently testable:

For a (slowly growing function) $f : N \rightarrow N$ let $L_f = \{w = b_1\$b_2\$...\$b_k|\ n(b_i) \leq f(|w|)\}$, where $b_i$ are binary strings starting with 1, and $n(b_i)$ is as before. Clearly $L$ has read-once BP, B, of width $f(n)+1$. In addition $L$ is $\epsilon$-non-trivial, that is, for every $n$ there are inputs of length $n$ that are $\epsilon n$-far from $L$. Hence, by Theorem 1, $L$ is $(\epsilon, (\frac{2^{f(n)}}{\epsilon})^{f(n)}) - testable$. In particular for, say, $f(n) = o(\log\log n)$ we have that $L$ is $(\epsilon, o(n^\delta))$-testable for any $\delta > 0$. The query complexity can actually be improved, as one can chose the $m$ levels in the Proof of Lemma 3.1 such that their width is $w = 2$ and use a recursion of two folds only. This will result in an $(O(1), O(f(n)))$-testing algorithm. In fact, a direct testing algorithm of complexity $O(\log f(n))$ is quite easy to develop for this case. However, for the following, an efficient test is harder to obtain:

Let $L_1 = \{w = b_1\$b_2\$...\$b_k|\ |n(b_i) - n(b_{i-1})| \leq 1\}$, and let $L' = L \cap L_1$, where $L$ is as in the previous paragraph. It can be easily seen that $L'$ has read-once BP of width $O(f^2(n))$, hence again for small enough $f$ (say $f(n) = \log\log\log n$) it has efficient testing algorithm. A direct algorithm is a bit harder in this case (although, there is a direct and more efficient algorithm then of the consequences of Theorem 1).

In view of Theorem 1 and the examples above, one may ask what is the true dependence of the query complexity in $w$ and $\epsilon$. This remains open at this point. Another more puzzling question is whether $SPACE(\log\log n)$ can be efficiently testable (by this we mean with complexity less then $n^\delta$ for any $\delta > 0$). Currently we do not have any counter example to this.

Another issue is how far the current result may be generalized. One restriction here that may be considered is the 'read-once' restriction - can this be replaced by, say, polynomial total size? To this, the answer is false: By Barrington [4], every $NC^1$ function has a polynomial length leveled BP of width 5. However, In [2] examples of such functions that require $\theta(\sqrt{n})$ queries are presented. Hence, instead, one may ask whether constant width linear size BP's are efficiently testable. Recently [8] show that this is not the case either. They construct a function that has read-three-times BP and that is not $\epsilon$-testable (read-k-times BP is a BP where each variable appears in at most $k$ levels). This shows that the result here is essentially tight in this respect.

## References


[1] N. Alon, E. Fischer, M. Krivelevich and M. Szegedy, Efficient testing of large graphs, Proceedings of the $40^{th}$ IEEE FOCS, (1999), 656-666.

[2] N. Alon, M. Krivelevich, I. Newman and M. Szegedy, Regular Languages are Testable with a Constant Number of Queries, Proceedings of the $40^{th}$ IEEE FOCS, (1999), 645–655.

[3] S. Arora, C. Lund, R. Motwani, M. Sudan and M. Szegedy, Proof verification and the hardness of approximation problems, Journal of the ACM, 45(3) (1998), 501-555.

[4] D. M. Barrington, Bounded-Width Polynomial-Size branching programs recognize exactly those languages in $NC^1$, JCSS 38 (1989), 150-164.

[5] P. Beam, M. Saks and J.S Thathachar, Time-Space Tradeoffs for Branching Programs, Proceedings of the $39^{th}$ IEEE FOCS, (1998), 254-263.

[6] M. Blum, M. Luby and R. Rubinfeld, Self-testing/correcting with applications to numerical problems. J. Computer System Sciences 47 (1994), 549–595.

[7] R. B. Boppana and M. Sipser, The complexity of finite functions, In "Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity", 1990, 757-804.

[8] E. Fischer and I. Newman, manuscript, in preparation.

[9] P. Gemmell, R. Lipton, R. Rubinfeld, M. Sudan and A. Wigderson, Self-testing/correcting for polynomials and for approximate functions, Proceedings of the $23^{th}$ ACM STOC (1991), 32–42.

[10] S. Jukna, A. A. Razborov, P. Savick, I. Wegener, On P versus $NP \cap co-NP$ for Decision Trees and Read-Once Branching Programs. MFCS 1997, 319-326

[11] O. Goldreich, S. Goldwasser and D. Ron, Property testing and its connections to learning and approximation. To appear in J. of the ACM, also: Proceedings of the $37^{th}$ IEEE FOCS (1996), 339–348.

[12] C. Papadimitriou, Computational Complexity, Addison-Wesley 1994, exercise 2.8.11 and 1.8.12 pp. 54-55.

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