Testing membership in languages 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 [16], 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 'far' from having the property.

The main result here is that if $\mathcal{G} = \{g_n : \{0,1\}^n \to \{0,1\}\}$ is a family of Boolean functions that have oblivious 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 $g_n(x) = 1$, and rejects it with high probability if at least ϵn bits of x should be modified in order for it to be in $g_n^{-1}(1)$. The algorithm makes $(\frac{2^w}{\epsilon})^{O(w)}$ queries. In particular, for constant ϵ and w, the query complexity is $O(1)$.

This generalizes the results of Alon et. al. [2] asserting that regular languages are ϵ-testable for every $\epsilon > 0$.

1 Introduction

Combinatorial property testing, initiated formally by Goldreich, Goldwasser and Ron in [11], and inspired by Rubinfeld and Sudan [16], 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 'far' from having the property. A property here is a set of binary strings (those inputs that have the 'property') and is identified 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 need 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 (ϵ, 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 ϵn-far from having the property. Varying ϵ and n may result in different algorithms with different query complexity $q = q(\epsilon, n)$, that may depend on both ϵ and n. If for a fixed $\epsilon > 0$ and every large enough n, a property

*A preliminary version of this paper appeared in the Proceedings of the 41^{th} Symposium on Foundation of Computer Science (FOCS'00), IEEE Press 2000.

P is (ϵ, q)-testable with number of queries q that is independent of length of the input, n, then we say that P is ϵ-testable. If for every $\epsilon > 0$ P is ϵ-testable then P is said to be *testable.*

Apart from being a natural relaxation of the standard decision problem, combinatorial property testing emerges naturally in the context of PAC learning, program checking [10, 6, 16], 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 the graph property of being bipartite is testable. They also raise the question of obtaining general results identifying classes of properties that are testable. 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 membership in any regular language is testable, hence obtaining a general result identifying a non trivial class of properties each being testable. Here we further pursue this direction: We prove that if a language has a (non-uniform) oblivious 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 functions that can be defined by a non-uniform collection of constant width oblivious read-once branching programs is testable. This also generalizes and gives an alternative proof and algorithm for the result of [2], as regular languages can be represented by constant width oblivious read-once branching programs. We note however, that the dependence of the query complexity here is worse than in [2].

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's of width w that have the further restriction of being *oblivious 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 (citebopp-sip contains a survey text, see also [4, 5, 13] for a very partial list of different aspects involving BP's and read-once BP's).

The size of a BP (and a read-once BP) is tightly related to the space complexity of the function it computes: If a language is in $SPACE(s)$ then it has a BP of total size of at most $n \cdot 2^{O(s)}$ [8], and also a read-once branching program of width $2^{O(2^s)}$ [12]. 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 $SPACE(O(1))$, hence 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 [12]. 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]. In regular languages the same finite automaton is used to test all the words, even of different lengths. On the other hand, when represented by a family of BP's, each BP computes the characteristic function of the property for a given input length. There are languages of arbitrary complexity that can be represented by $O(1)$-width oblivious BP's. Our result apply to such cases as well. This includes the family of $O(1)$-terms DNF, $O(1)$-clauses CNF and some other interesting examples (see section 4).

Finally we note that $SPACE(O(\log n))$ functions are not testable in general; [2, 11, 15] contain lower bounds showing that some functions in $SPACE(O(\log n))$ are not ϵ-testable, and sometimes not even (ϵ, n^δ)-testable for some fixed $\epsilon, \delta < 1$. However, the question of whether properties in $SPACE(s)$ for $\log\log n \leq s << \log n$ are 'efficiently' testable is open. In particular we don't have any candidate for a $SPACE(O(\log\log n))$ function whose ϵ-testing requires $n^{\Omega(1)}$ queries for some fixed $\epsilon > 0$.

2 Definitions and Notations

We identify properties with the collection of their characteristic Boolean functions, namely: A property $\mathcal{P} \subseteq \{0,1\}^*$ is identified with $\{f : \{0,1\}^n \longrightarrow \{0,1\}\}$ so that $f(x) = 1$ if and only if $x \in \mathcal{P}$.

An oblivious leveled branching program (BP), is a directed graph B, in which the nodes are partition into levels $L_0, ..., L_m$. There are two special nodes; a 'start' node belonging to L_0 and an 'accept' node belonging to L_m. Edges are going only from a level to nodes in the consecutive level. Each node has at most two out-going edges one of which is labeled by '0' and the other by '1'. In addition, all edges in between two consecutive levels are associated with a member of $\{1, ..., n\}$ (a Boolean variable). An input $x \in \{0,1\}^n$ naturally defines a path starting at the *start*-node: At each step, if the edges are associated with i then the edge with the label identical to the value of x_i is chosen. A leveled BP defines a Boolean function $g : \{0,1\}^n \longrightarrow \{0,1\}$ in the following way: $g(x) = 1$ if the path that x defines reaches *accept*. This definition of branching programs is essentially equivalent to what is sometimes called 'deterministic' branching programs (as each input defines at most one path from each node). However, note that this definition is slightly different from the standard definition of deterministic BP's in which every vertex has exactly two outgoing edges, one that is labeled by '1' and the other by '0'. Here instead, an input x can be 'stuck' at an internal node w due to the fact that w has just one outgoing edge that is associated with i and is labeled by a value that is opposite to that of x_i (this cannot happen in the standard definition). A leveled BP is of width w (w-width) if its largest level contains w nodes.

An *oblivious read-once* BP computing $g : \{0,1\}^n \longrightarrow \{0,1\}$ is a leveled BP with the additional property that edges ending in distinct levels are labeled with distinct variables. This implies also that there are exactly $n+1$ levels (for a function that depends on all its n variables). We number the levels of the BP from 0 (containing the start s) and on, and associate to the edges in between levels the formal Boolean variables $X_1, ..., X_n$ consecutively (by possibly renaming the variables). We may assume that the last level is numbered by n.

Along the sequel we consider only oblivious read-once BP's. For a given BP, B, 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. If $u \in L_i$ and $v \in L_j$ then $B[u : v]$ computes a Boolean function on the variables $X_i, ..., X_j$. The length of $B[u : v]$ in this case is $\nu = j - i$. Such $B[u : v]$, as a subprogram of a read-once oblivious BP, is also read-once oblivious BP. When discussing such a BP $B[u : v]$, we renumber its levels so that its first level, which is level L_i in B, is denoted $L_0(B[u : v])$ and its last level is denoted by $L_\nu(B[u : v])$. When it is clear from the context which is the BP that is considered, we just refer to its first and last levels as L_0, L_ν respectively (where ν is the length of the corresponding BP).

We will be interested in BP's for which the start and accept nodes are not always defined. Namely, the BP B, might have multiple nodes in its first and last levels. For such a BP of length n, any choice of start and accept nodes $(s,t) \in L_0 \times L_n$ defines a different function on n variables. If no path from a node $w \in B$ reaches the last level, then deleting w from B will not change the function that B computes for any choice of start and accept nodes in the first and last level. Similarly, we may delete every vertex that can be reached from no vertex of the first level. Also, when we talk about $B[u:v]$ for some specific nodes u, v we may delete any node from $B[u:v]$ that is either not reachable from u or cannot reach v. In particular, this means that u is the only node in $L_0(B[u:v])$ and v is the only node in the last level of $B[u:v]$. All such nodes are called 'unnecessary nodes'. Along the sequel, we always assume that all BP under discussion contain no 'unnecessary nodes'.

For integers $a < b$, we denote by $B_{a:b}$ the sub program of B containing all nodes in levels $L_a, L_{a+1}, ..., L_b$. $B_{a:b}$ has undefined source and sink. Note that if B is an oblivious read-once BP of width w then for any two nodes u, v, and any two numbers a and b, $B[u:v]$ and $B_{a:b}$ are oblivious read-once BP's of width at most w. (The width can become smaller as nodes might become 'unnecessary'.)

Let $x, y \in \{0,1\}^n$, we define $dist(x,y) = hamming(x,y) = |\{i|\ x_i \neq y_i\}|$. Let $g : \{0,1\}^n \longrightarrow \{0,1\}$ such that $g^{-1}(1) \neq \phi$, we define $dist(x,g) = min\{dist(x,y)|\ y \in g^{-1}(1)\}$. For a BP B, and two nodes u and v in levels L_i, L_j respectively, let $dist(x, B[u:v]) = dist(x[i,j], g')$ where g' is the function computed by $B[u:v]$ on the formal variables $X_i, X_{i+1}, ..., X_j$.

Let B be an oblivious read-once BP with fixed start and accept nodes, that computes a Boolean function $g : \{0,1\}^n \longrightarrow \{0,1\}$. A randomize algorithm $\mathcal{A}$ is a 1-sided error ϵ-test for B (g), of query complexity $c(\mathcal{A})$, if for every input $x \in \{0,1\}^n$ it queries at most $c(\mathcal{A})$ queries and:

1. For every input $x \in g^{-1}$ the algorithm accepts.
2. For every input $x \in \{0,1\}^n$ for which $dist(x,g) \geq \epsilon n$ the algorithm rejects with probability at least 2/3.

Let $\mathcal{B}_w^n$ be the set of all oblivious read-once BP's of width w and length n. For $B \in \mathcal{B}_w^n$ we denote by $\tilde{c}(\epsilon, B) = min\{c(\mathcal{A}) :\ \mathcal{A}\ is\ a\ 1\text{-}sided\ error\ \epsilon\text{-}test\ for\ B\}$. Namely, $\tilde{c}(\epsilon, B)$ is the query complexity of the best 1-sided error ϵ-test for B. Let $\tilde{q}(\epsilon, w) = max\{\tilde{c}(\epsilon, B) :\ B \in \mathcal{B}_w^n\}$. Namely, $\tilde{q}(\epsilon, w)$ is the worst query complexity needed to ϵ-test a w-width BP. Formally, $\tilde{q}(\epsilon, w)$ is a function of n too, however, as we shall see, asymptotically this is not the case.

Finally, in the sequel, for ease of notations we neglect taking $\lceil\rceil$ and $\lfloor\rfloor$ for numbers, even when they need to be integers, whenever this is clear from the context and has no bearing on the essence of proofs.

3 Results

Theorem 1 *Let $g : \{0,1\}^n \longrightarrow \{0,1\}$ be computed by an oblivious read-once BP of width w. Then there is an ϵ-test for g that makes $(\frac{2^w}{\epsilon})^{O(w)}$ queries.*

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

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. This approach has prospects since 1-width BP's are testable as asserted by the following proposition.

Proposition 1 *If $g : \{0,1\}^n \longrightarrow \{0,1\}$ is computable by an oblivious read-once BP of width $w = 1$, then g is $(\epsilon, O(\frac{1}{\epsilon}))$-testable by a 1-sided error algorithm.*

Proof: We assume that g is not identically '0' and not identically '1', as otherwise the test is trivial (with no queries at all). Let B be a BP of width $w = 1$, computing the non-zero function g. It is clear from the definition that g is a one term DNF. That is, written in formal variables $X_1, ..., X_n$, $g = \Pi_{i=1}^n t_i$ where t_i is either X_i or $\bar{X}_i$. g does not necessarily depend on all its variables. In this case we just look at the variables it does depend on. Let x be an input such that $dist(g, x) \geq \epsilon n$ (assuming that there is such x). It is easy to see that for at least $\epsilon \cdot n$ of the places $\{1, ..., n\}$ x_i is not consistent with t_i. Hence sampling $O(\frac{1}{\epsilon})$ bits of an input x and rejecting if x_i is inconsistent with t_i is guaranteed to succeed with probability 2/3 for ϵn-far inputs, and with probability 1 for any input x for which $g(x) = 1$. ■

For the proof of the case $w \geq 2$ we will need some machinery developed hereafter.

3.1 Main definitions and some intuition

The algorithm for ϵ-testing w-width BP's will be recursive on the width. Namely, our aim is to reduce ϵ-testing of a w-width BP to testing of $(w-1)$-width BP's with possibly a smaller ϵ. Key notions are that of *r-full* levels and decomposable BP's. They are defined below.

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

$$A_r(v) = \{u | \text{ there is a path of length } r \text{ from } u \text{ to } v\}.$$

See Figure 1.

Definition 3.2 *Let v be a vertex in level L_l of a branching program with start and accept nodes that are not necessarily defined. We say that v is r-**full** if $A_r(v)$ contains all nodes of level L_{l-r}. If every vertex in level L_l is r-full then L_l is said to be r-full.*

Namely, a vertex v is r-full if v is reachable from every vertex of level L_{l-r}, see Figure 1. Note that for every two nodes u and v of a BP B, v is always 1-full with respect to $B[u : v]$. This is due to the fact that $B[u : v]$ contains no 'unnecessary nodes'.

Fact 1 *Assume that $v \in L_l$ is r-full for a certain r and l, then:*

- *v is r'-full for every $r' > r$.*
- *If $u \in L_{l+1}$ is a neighbor of v then u is $(r+1)$-full.*

Proof: For the first part assume that $r' > r$ and $v' \in L_{l-r'}$. Then, as we assume that there are no 'unnecessary vertices', v' can reach some vertex $v'' \in L_{l-r}$. In turn v'' can reach v by the assumption that v is r-full. Hence v' can reach v.

For the second part; if v is r-full it can be reached from any $w \in L_{l-r}$. Since u is a neighbor of v, it can also be reached by every $w \in L_{l-r}$. ■

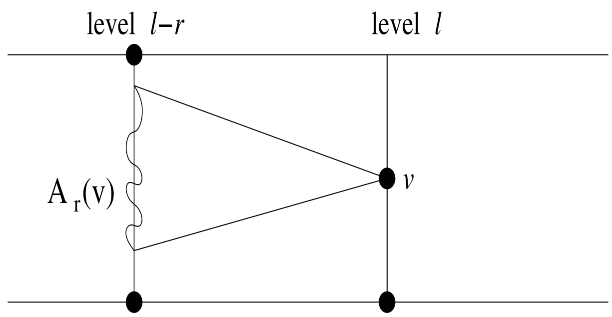

Figure 1: $A_r(v)$ is the set of nodes in level L_{l-r} that can reach v. Here v is **not** r-full as not all nodes in level L_{l-r} can reach v.

The following is a crucial ingredient for the rest of the sequel.

Definition 3.3 *Let $\delta < 1$. A BP of length ν, with start and accept nodes that are not necessarily defined, is said to be δ-**decomposable** if for some $\frac{\delta\nu}{20} \leq \ell \leq \nu - 1$ and $r \leq \lfloor \frac{\delta\ell}{10} \rfloor$, L_ℓ is r-full.*

For a given BP, B, the role of the non δ-decomposable subprogram of B is the following: We first show in Section 3.2, that if B' is not δ-decomposable for $\delta < \epsilon$, then ϵ-testing B' can indeed be reduced to testing 'narrower' BP's. Then, in Section 3.3, we show how a general BP can be decomposed into disjoint non-decomposable sub-programs, such that testing the BP can be reduced to testing not too many of the non-decomposable parts of it.

3.2 Testing non-decomposable BP's

The following Lemma, which is the main technical part of the proof of Theorem 1, relates testing w-width non-decomposable BP's to the test of general $(w-1)$-width BP's.

Lemma 3.4 *Let $\delta \leq \epsilon$ and let B be a non δ-decomposable BP of width w and length n. Then ϵ-testing $B[s:t]$, for any start and accept nodes s and t, requires at most $O(\frac{w^4}{\delta^3}(\log \frac{w^2}{\delta})^2) \cdot \tilde{q}(0.8\epsilon, w-1)$ queries.*

Proof: The idea of the proof is as follows: We fix $O(\frac{1}{\delta^2})$ levels that are equally spaced in B, leaving out enough space in the beginning of B. The assumption that B is not δ-decomposable will imply that for each two nodes u, v in the levels we choose, the test of $B[u:v]$ can be reduced to tests of $(w-1)$-width BP's. We then show how to combine the results of the tests on $B[u:v]$, for all such u, v, into an ϵ-test for B.

Formally: Let $m = \lceil \frac{\delta^2 n}{400} \rceil$. Let $\{l_0, ..., l_p\}$ be the set of numbers that are m apart, starting from $\lceil \frac{20m}{\delta} \rceil$ and ending at or before n. Namely, $l_i = \lceil \frac{20m}{\delta} \rceil + i \cdot m, \quad i = 0, 1, ..., p = \lfloor \frac{n - l_0}{m} \rfloor = O(\frac{1}{\delta^2})$. Let $\mathcal{S} = L_{l_0} \times ... \times L_{l_p}$. Our first aim is to show that for every pair $(u, v) \in L_{l_i} \times L_{l_{i+1}}$ the ϵ_1-test of $B[u : v]$ can be reduced to a small number of general test of $(w-1)$-width BP's.

We first need the following claims.

Claim 3.5 *For every $l \geq l_0$ level L_l is not $(2m)$-full.*

Proof: Immediate from the choice of parameters and the fact that B is not δ-decomposable. ■

For each l such that $l_i < l \leq l_{i+1}$, let $F(l)$ be the set of all $(l - l_i)$-full vertices in level L_l. In other words, $v \in F(l)$ if it is in the l-th level and it is reachable from every vertex of the l_i-th level. By our assumption on B, $F(l) \neq L_l$ as otherwise L_l would be $(l - l_{i-1})$-full in contradiction with Claim 3.5.

Hence the above implies the following claim:

Claim 3.6 *Let u, v be vertices in levels $L_{l_i}, L_{l_{i+1}}$ respectively and let l be such that $l_i \leq l \leq l_{i+1}$.*

- *Let u' be in level L_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_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_l. As $u' \notin F(l)$, u' is not $(l - l_i)$-full, then, by Fact 1, it is also not $(l - l')$-full for every $l' > l_i$. Namely, for every intermediate level $L_{l'}$, $l_i < l' < l$, there is a vertex that cannot reach u' and hence can be deleted from $B[u : u']$.

For the second part assume first that v' is in level L_l for $l > l_i$ and $v' \in F(l)$. Let t be any node at level $L_{l'}$, $l < l' \leq l_{i+1}$, that is reachable from v'. Since $v' \in F(l)$ it follows that t is $(l' - l_i)$-full. Hence not all vertices in level $L_{l'}$ are reachable from v' as otherwise level $L_{l'}$ will be $(l' - l_i)$-full in contradiction to Claim 3.5. As this is true for every $l < l' \leq l_{i+1}$ it follows 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 $L_{l'}$, $l_i < l' \leq l_{i+1}$ cannot have all its nodes reachable from v'. Otherwise it would be $(l' - l_{i-1})$-full in contradiction to Claim 3.5. ■

Claim 3.6 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 $v_{i+1} \in F(l_{i+1})$ where the subprogram $B[v_i : v_{i+1}]$ might be of width w. The key observation here is that any path from v_i to v_{i+1} must start at $L_{l_i} - F(l_i)$ (as v_i is such) and end in $F(l_{i+1})$. Hence this path must intersect $F(l)$ for some intermediate level L_l, $l_i < l \leq l_{i+1}$. In addition, by Fact 1, once it intersects $F(l)$, it intersects $F(l')$ for every $l' > l$ (see Figure 2). This suggests the following:

Let $k = \frac{10}{\delta}$, we choose $k+1$ numbers, $p_0, ..., p_k$, that are $\frac{m}{k}$ apart in the range $\{l_i, ..., l_{i+1}\}$: $p_j = l_i + j \cdot \frac{m}{k}, \quad j = 0, ..., k$.

Claim 3.7 *For every $u \in L_{l_i} - F(l_i)$ and $v \in F(l_{i+1})$:*

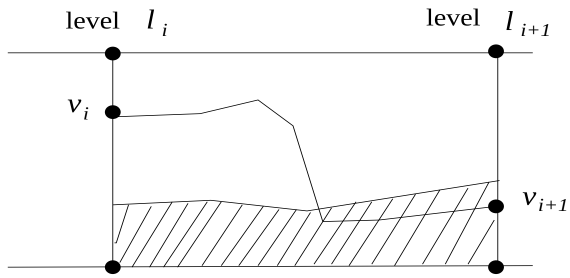

Figure 2: Vertices in shadowed area are in $F()$. If a path from v_i to v_{i+1} intersects $F(l)$ at some intermediate level L_l, then it intersects $F()$ for every following level.

- *If $y \in \{0,1\}^n$ is such that $dist(y, B[u:v]) = 0$, then for some $j \in \{1,...,k\}$ there are some $u' \in L_{p_{j-1}} - F(p_{j-1})$, $v' \in F(p_j)$ so that $dist(y, B[u:u']) = dist(y, B[v':v]) = 0$.*
- *If $y \in \{0,1\}^n$ is such that $dist(y, B[u:v]) \geq (1-\alpha)\epsilon m$ for some $\alpha < 1$. Then, for every $j \in \{1,...,k\}$ and for every $u' \in L_{p_{j-1}} - F(p_{j-1})$ and $v' \in F(p_j)$ such that u can reach u', u' can reach v' and v' can reach v,*

$$dist(y, B[u:u']) + dist(y, B[v':v]) \geq (1-\alpha)\epsilon m - \frac{m}{k} \geq (0.9-\alpha)\epsilon m. \quad \blacksquare$$

Proof: If $dist(y, B[u:v]) = 0$ then by the discussion above there is some level $l_i < l \leq l_{i+1}$ so that the path, $Path(y)$, that y defines from u to v intersects $F(l')$ for each $l \leq l' \leq l_{i+1}$ and does not intersect $F(l'')$ for each $l_i \leq l'' < l$. Let j be the smallest such that $p_j \geq l$. Let u' be the vertex that $Path(y)$ intersects in $L_{p_{j-1}}$ and v' be the vertex that $Path(y)$ intersects in L_{p_j}. Clearly for these j, u', v' the first part of the claim holds.

For the second part, assume that for $y \in \{0,1\}^n$ there is a $j \in \{1,...,k\}$, $u' \in L_{p_{j-1}} - F(p_{j-1})$ and $v' \in F(p_j)$ such that u can reach u', u' can reach v' and v' can reach v, and such that $dist(y, B[u:v]) < (0.9-\alpha)\epsilon m$. Then certainly $dist(y, B[u:v]) < (1-\alpha)\epsilon m$: First, by changing at most $(0.9-\alpha)\epsilon m$ bits of y in the range $\{l_i+1,...,p_{j-1}\}$ and $\{p_j+1,...,l_{i+1}\}$, we can get a y' such that its corresponding parts (to the places above) traverse B from u to u' and from v' to v. Then by changing possibly additional $m/k \leq 0.1\epsilon m$ bits, namely all bits in the range $\{p_{j-1}+1,...,p_j\}$, we get a y'' that traverses B from u to v through u' and v'. $\blacksquare$

We now can present the algorithm that $(1-\alpha)\epsilon$-test $B[u:v]$ for each $(u,v) \in L_{l_i} \times L_{i+1}$ given that we have a general 1-sided error test for $(w-1)$-width BP's. Note that the length of $B[u:v]$ for any such u and v is m.

Algorithm $A_1((1-\alpha)\epsilon, w, B[u:v])$:
The first parameter is relative distance, the 2nd is width, $(u,v) \in L_{l_i} \times L_{l_{i+1}}$.

1. If $u \in F(l_i)$ or $v \notin F(l_{i+1})$ then by Claim 3.6, $B[u:v]$ is already of width $w' \leq w-1$. This test is done by calling the general $(1-\alpha)\epsilon$-testing procedure for $(w-1)$-width BP's.
2. Otherwise, if $u \notin F(l_i)$ and $v \in F(l_{i+1})$ let $k = \frac{10}{\delta}$ and $\rho = 1 + \frac{\log(kw^2)}{\log 3}$. We choose in the range $\{l_i, ..., l_{i+1}\}$ $k+1$ numbers, $p_0, ..., p_k$, that are $\frac{m}{k}$ apart: $p_j = l_i + j \cdot \frac{m}{k}$, $j = 0, ..., k$.

   For every triplet $(j, u'v')$ such that $j \in \{1, ..., k\}$, $u' \in L_{p_{j-1}} - F(p_{j-1})$, $v' \in F(p_j)$, and such that u can reach u', u' can reach v' and v' can reach v, a $(0.9-\alpha)\epsilon$-test is performed ρ independent times on $B[u:u']$ and $B[v':v]$. This is done by calling the general procedure for testing $(w-1)$-width BP's.

   If there is a triplet $(j, u'v')$ for which all ρ tests pass, then the outcome of A_1 is 'Yes'.

   Otherwise, if for every triplet (j, u', v') one or more of the ρ tests, on either $B[u:u']$ or $B[v':v]$ answer 'No', then the outcome of A_1 is 'No'.

Claim 3.8 *Let B be a w-width BP that is not δ-decomposable, and m, k as above. Let $x \in \{0,1\}^n$ be any input, then Algorithm A_1 makes $O(\frac{w^2}{\delta} \cdot \log \frac{w^2}{\delta})$ calls for a general $(1-\alpha)\epsilon$-test of $(w-1)$-width programs on x and:*

- *If $dist(x, B[u:v]) = 0$ then A_1 answers 'Yes' on x with probability 1.*
- *If $dist(x, B[u:v]) \geq (1-\alpha)\epsilon m$ then the outcome of A_1 on x is 'No' with probability at least $2/3$.*

Proof: For each triplet (j, u', v') that is relevant to the second case of Algorithm A_1, Claim 3.6 asserts that $B[u:u']$ and $B[v':v]$ are of width at most $(w-1)$. Hence all calls of A_1 are to tests of $(w-1)$-width BP's. There are at most $O(k \cdot w^2) = O(\frac{w^2}{\delta})$ such triplets, hence the claim on the number of calls to $(w-1)$-width tests is obvious.

We assume that the general $(w-1)$-test is a 1-sided error. Hence, for an input $x \in \{0,1\}^n$ with $dist(x, B[u:v]) = 0$ a 'No' result will be obtained if $B[u:v]$ is of width at most $w-1$ and the general $(w-1)$-width test answers 'No' (1st case of A_1) or if for every triplet (j, u', v') as above, one of the tests, to either $B[u:u']$ or $B[v':v]$, answers 'No'. Both cases occur with probability 0 by Claim 3.7.

Now, let $x \in \{0,1\}^n$ be an input for which $dist(x, B[u:v]) \geq (1-\alpha)\epsilon m$. If $B[u:v]$ is of width $(w-1)$, then A_1 answer 'Yes' only if the general $(w-1)$-width test errs. This occurs with probability at most $1/3$. If $B[u:v]$ is of width w, then by Claim 3.7, for every triplet (j, u', v') as above, $dist(y, B[u:u']) + dist(y, B[v':v]) \geq (0.9-\alpha)\epsilon m$. But then, for each such triplet, either $dist(y, B[u:u']) \geq (0.9-\alpha)\epsilon \cdot \frac{j-1}{k} \cdot m$ or $dist(y, B[v':v]) \geq (0.9-\alpha)\epsilon \cdot (1 - \frac{k-j}{k})m$. In any of these cases, a general $(0.9-\alpha)\epsilon$-test to the corresponding $(w-1)$-width BP would erroneously say 'Yes' with probability at most $1/3$. Since there are ρ such independent tests, all these tests would err with probability at most $(\frac{1}{3})^{\rho} \leq \frac{1}{3kw^2}$. This will cause A_1 to say erroneously 'Yes' due to this triplet. As there are at most kw^2 possible triplets, A_1 errs with probability at most 1/3. ■

We formally now end the proof of Lemma 3.4 by presenting the following proposition and the testing algorithm it implies.

Proposition 2 *Let B be a non δ-decomposable BP of width w and length n. Let $m, \{l_0, ..., l_p\}$ and $\mathcal{S}$ be as defined above (right after the statement of Lemma 3.4). Let $y \in \{0,1\}^n$, then for any start and accept nodes $(s,t) \in L_0 \times L_n$:*

1. *If $dist(y, B[s:t]) = 0$, then there exists a tuple $(v_0, ..., v_p) \in \mathcal{S}$ such that s can reach v_0, v_p can reach t and $dist(y, B[v_i : v_{i+1}]) = 0$ for $i = 0, ..., p-1$.*
2. *Let $dist(y, B[s:t]) \geq \epsilon n$, then for each $(v_0, ..., v_p) \in \mathcal{S}$ such that s can reach v_0 and v_p can reach t, $\Sigma_{i=0}^{i=p-1} dist(x, B[v_i : v_{i+1}]) \geq \epsilon n - l_0 - (n - l_p) \geq 0.9\epsilon n$.*

Proof: If $dist(y, B[s:t]) = 0$, then the path that y takes in B defines the tuple $(v_0, ..., v_p) \in \mathcal{S}$ which contains the nodes in which this path intersects L_{l_i}, $i = 0, ..., p$ along the way from s to t. This tuple asserts the first item of the proposition.

If $dist(y, B[s:t]) \geq \epsilon n$, then for any $(v_0, ..., v_p) \in \mathcal{S}$ such that s can reach v_0 and v_p can reach t, $dist(y, B[v_0 : v_p]) \geq \epsilon n - l_0 - (n - l_p) \geq 0.9\epsilon n$. But $dist(y, B[v_0 : v_p]) = \Sigma_{i=0}^{i=p-1} dist(x, B[v_i : v_{i+1}])$. ■

Proposition 2 defines a way of how to combine answers to tests on BP's of the form $B[v_i : v_{i+1}]$ into an ϵ-test of B. Intuitively, on an input $x \in \{0,1\}^n$, we just need to check for all tuples $(v_0, ..., v_p) \in \mathcal{S}$, and check whether there exists one for which $dist(x, B[v_i : v_{i+1}]) = 0$ for $i = 0, ..., p-1$.

Formally, let $x \in \{0,1\}^n$ be the input. The following is an ϵ-test of B for any start and accept nodes:

Algorithm $A_2(\epsilon, w)$: (B is a non δ-decomposable BP of width w).
Let m and $\mathcal{S}$ be as above and let $\nu = 1 + \frac{\log(pw^2)}{\log 3} = O(\log \frac{w}{\delta})$.

1. For each $(u,v) \in L_{l_i} \times L_{l_{i+1}}, i = 0, ..., p-1$, call $A_1(0.9 \cdot \epsilon, w, B[u:v])$ (namely with $\alpha = 0.1$) independently, for ν times. If for (u,v) all of these tests answer 'Yes' then define $T(u,v) = 1$. Otherwise, if there is a test out of the ν tests that answers 'No' for (u,v) then set $T(u,v) = 0$.
2. Define the following directed graph $G = (V, E)$: $V = L_0 \cup L_n \cup (\cup_{i=0}^{p} L_{l_i})$ and
$$E = \{(s,u) \in L_0 \times L_{l_0} |\ s \text{ can reach } u \text{ in } B\} \cup$$
$$\{(v,t) \in L_{l_p} \times L_n |\ v \text{ can reach } t \text{ in } B\} \cup$$
$$\{(u,v) \in L_{l_i} \times L_{l_{i+1}}, i = 0, ..., p-1 |\ \text{such that } T(u,v) = 1\}.$$
3. Answer 'Yes' for $(s,t) \in L_0 \times L_n$ if and only if s can reach t in G.

Claim 3.9 *For any $(s,t) \in L_0 \times L_n$ and for every input x:*

1. *If* $dist(x, B[s:t]) = 0$ *then Algorithm* A_2 *answers 'Yes' on* (s,t) *with probability 1.*

2. *If* $dist(x, B[s:t]) > \epsilon n$ *then Algorithm* A_2 *answers 'No' on* (s,t) *with probability at least* $2/3$.

Proof: Assume that $dist(x, B[s:t]) = 0$ for an input $x \in \{0,1\}^n$ and $(s,t) \in L_0 \times L_n$. Then, by Proposition 2, there exists a tuple $(v_0, ..., v_p) \in \mathcal{S}$ such that s can reach v_0, v_p can reach t and $dist(y, B[v_i : v_{i+1}]) = 0$ for $i = 0, ..., p-1$. By Claim 3.8 Algorithm A_1 answers 'Yes' on each of the calls $A_1(0.9 \cdot \epsilon, w, B[v_i : v_{i+1}])$ with probability 1. Hence, the path $(s, v_0, ..., v_p, t)$ is a valid path in G with probability 1, causing A_2 to answer 'Yes' with the same probability.

For the second part assume that $dist(x, B[s:t]) > \epsilon n$. Then, by Proposition 2, for each $(v_0, ..., v_p) \in \mathcal{S}$ such that s can reach v_0 and v_p can reach t, $\Sigma_{i=0}^{i=p-1} dist(x, B[v_i : v_{i+1}]) \geq 0.9\epsilon n$. But then for each such $(v_0, ..., v_p) \in \mathcal{S}$, for some $i \leq p-1$, $dist(x, B[v_i : v_{i+1}]) \geq 0.9\epsilon\frac{n}{p} > 0.9\epsilon m$. Let E' be the set that contains for each $(v_0, ..., v_p) \in \mathcal{S}$ a corresponding (v_i, v_{i+1}) for which $dist(x, B[v_i : v_{i+1}]) \geq 0.9\epsilon m$. Note that E' defines an (s,t)-cut in G. Namely, s cannot reach t in $G - E'$.

For each member $(v_i, v_{i+1}) \in E'$, Claim 3.8 asserts that Algorithm A_1 answers 'No' on the call $A_1(0.9 \cdot \epsilon, w, B[v_i : v_{i+1}])$ with probability 2/3. Hence it answers erroneously 'Yes' on all ν calls for a pair $(u,v) \in E'$ with probability at most $(\frac{1}{3})^\nu \leq \frac{1}{3pw^2}$. Namely, $T(u,v)$ is set erroneously to '1' in step 1 of the algorithm with probability at most $\frac{1}{3pw^2}$. However, there are at most pw^2 possible pairs in E'. This implies that with probability at most 1/3 there exists a pair $(u,v) \in E'$ for which $T(u,v) = 1$. In particular, it follows that s cannot reach t in G with probability at least 2/3. ■

Claim 3.10 *For any* $(s,t) \in L_0 \times L_n$ *and for every input* x*, Algorithm* A_2 *make* $O(\frac{w^2}{\delta^2} \log \frac{w}{\delta})$ *calls to* A_1 *with distance parameter* 0.9ϵ.

Proof: There are $O(pw^2) = O(\frac{w^2}{\delta^2})$ possible pairs $(u,v) \in L_{l_i} \times L_{l_{i+1}}, i = 0, ..., p-1$. For each pair (u,v) there are $\nu = O(\log \frac{w}{\delta})$ calls for A_1. ■

The following corollary ends the proof of Lemma 3.4.

Corollary 3.11 *Algorithm* A_2 *provides a 1-sided error* ϵ*-test for* $B[s:t]$*, for every start and accept nodes* $(s,t) \in L_0(B) \times L_n(B)$*, making at most* $O(\frac{w^4}{\delta^3}(\log \frac{w^2}{\delta})^2) \cdot \tilde{q}(0.8\epsilon, w-1)$ *queries.*

Proof: Claim 3.9 asserts the correction of A_2 as a 1-sided error ϵ-test for $B[s:t]$, for each $(s,t) \in L_0(B) \times L_n(B)$. Observe that the calls for A_1 do not depend on the choice of s and t. Hence, with the same amount of queries as described above for a given choice of s, t, A_2 provides an ϵ-test for every choice of s and t; for each s and t the outcome has probability at least 2/3 to be correct.

According to Claim 3.8, each call for A_1 results in possibly $O(\frac{w^2}{\delta} \cdot \log \frac{w^2}{\delta})$ calls to a general 0.8ϵ-test of $(w-1)$-width BP's. Claim 3.10 asserts that there are $O(\frac{w^2}{\delta^2} \log \frac{w}{\delta})$ calls to A_1, hence the claim follows. ■

3.3 The general case

In order to test general w-width BP's it remains to be shown how to reduce testing of decomposable BP's to that of non-decomposable ones. We need the following:

Proposition 3 *For a BP, B and $t > r$, let t_1, t_2 be r-full vertices in L_t and let $u \in L_l$, with $l \le t - r$. Then for every $y \in \{0,1\}^n$*

$$|dist(y, B[u : t_2]) - dist(y, B[u : t_1])| \le r.$$

Proof: The closest y' to y that traverses B from u to t_1 must intersect $A_r(t_2)$. Hence, by changing only the r last bits of y', we get a y'' that traverses B from u to t_2. ∎

Definition 3.12 *Let $y \in \{0,1\}^n$ and $0 \le a < b \le n$, we define:*

$$dist(y, B_{a:b}) = min\{dist(y, B[u : v])| \quad u \in L_a, v \in L_b\}.$$

Claim 3.13 *Let $B[s : t]$ be a BP of length ν with start vertex s (the only vertex at level L_0) and accept vertex t (the only vertex at level L_ν). Assume that there is a sequence of numbers $l_0 = 1, ..., l_h = \nu$ and a sequence of numbers $r_1, ..., r_h$, such that level L_{l_i} is r_i-full for each $i = 1, ..., h$. Then, for every $y \in \{0,1\}^\nu$, $\Sigma_i dist(y, B_{l_i:l_{i+1}}) \ge dist(y, B[s : t]) - \Sigma_i r_i$.*

Proof: Let y be such that $\Sigma_1^h dist(y, B_{l_{i-1}:l_i}) = d$. We will show that $dist(y, B[s : t]) \le d + \Sigma_i r_i$ which implies the claim.

Indeed, let $w_i = y[l_{i-1} + 1, : l_i]$, $i = 1, ..., h$ be the substring of y that corresponds to the variables of $B_{l_{i-1}:l_i}$. Let y_i', $i = 1, ..., h$ be such that $dist(w_i, y_i') = d_i$, $dist(y_i', B_{l_{i-1}:l_i}) = 0$ and so that $\Sigma_1^h d_i = d$. Then for each y_i', $i = 1, ..., h$ let $(u_i, v_i) \in L_{l_{i-1}} \times L_{l_i}$ be such that $dist(y_i', B_{l_{i-1}:l_i}) = dist(y_i', B[u_i : v_i]) = 0$. Namely, u_i, v_i are the start and end nodes through which y_i' travels through $B_{l_{i-1}:l_i}$. Then, by Proposition 3, for every $i = 1, ..., h$ there is a string z_i such that $dist(y_i', z_i) \le r_i$ and $dist(z_i, B[u_i : u_{i+1}]) = 0$. But then the string $z = z_1 \cdot ... \cdot z_h$, which is the concatenation of $z_i, i = 1, ..., h$, 'travels' in B through all the $u_i, i = 0, ..., h$. In particular $dist(z, B[s : t]) = 0$ (as s and t are the only nodes in levels L_0, L_ν respectively).

On the other hand $dist(y, B[s : t]) \le dist(y, z) \le \Sigma_i dist(w_i, z_i) \le \Sigma_i(dist(w_i, y_i') + dist(y_i', z')) \le \Sigma_i(d_i + r_i) = d + \Sigma_i r_i$. ∎

We are ready now to prove Theorem 1.

Proof: Let B be a BP of width w and length n with start and accept nodes $s \in L_0$ and $t \in L_n$ respectively. Let $a_0 = 0$ and let $a_1 = l_1$ be the smallest integer such that level L_{a_1} is r_1-full for $r_1 \le \frac{\epsilon l_1}{20}$. Let l_2 be the smallest integer for which level L_{a_2}, $a_2 = a_1 + l_2$ is r_2-full for $r_2 \le \frac{\epsilon l_2}{20}$, etc. This defines a sequence of numbers $\mathcal{L} = (a_0, a_1, ...)$ of which the last may or may not be n. If the last number in $\mathcal{L}$ is not n then we add n as the last member resulting in a sequence $\mathcal{L}'$, otherwise we set $\mathcal{L}' = \mathcal{L}$. Assume that $\mathcal{L}' = (a_0 = 0, a_1, ..., a_h = n)$. This defines a sequence of h BP's (with start and accept nodes that are not necessarily defined), $B_1, ..., B_h$, $B_i = B_{a_{i-1}:a_i}$ of length $l_i = a_i - a_{i-1}$.

Note, by our choice, for every $i = 1, ..., h$, either $l_i = O(1)$ or B_i is not ϵ_1-decomposable for $\epsilon_1 = 0.5\epsilon$. Moreover, for every $i = 1, ..., h-1$, the last level of B_i is r_i-full, and for B_h (with L_n as last level) L_n is either r_i-full if $n \in \mathcal{L}$ or is 1-full if n was added to result in $\mathcal{L}'$ (as t is always 1-full in $B[s:t]$).

An ϵ-test of B is done as follows: For $4/\epsilon$ times, independently an $i \in \{1, ..., h\}$ is chosen at random with probability proportional to the length l_i. Let I be the multi-set that contains the $4/\epsilon$ chosen i's, possibly with multiplicity. Let T_i be a Boolean flag associated with each $i \in I$. For each $i \in I$ an ϵ_1-test is performed on $B_{a_{i-1}:a_i}$ for every choice of start and accept nodes $(u, v) \in L_{a_{i-1}} \times L_{a_i}$. If for some pair $(u, v) \in L_{a_{i-1}} \times L_{a_i}$, the answer to (u, v) in this test is 'Yes' then we mark T_i as '1'. Otherwise, if for all such pairs (u, v) the answer is 'No', we mark it as '0'.

Finally, if there exists a chosen $i \in I$ for which $T_i = 0$, then the answer to the ϵ-test for B is 'No'. Otherwise, if for all chosen i's $T_i = 1$, then the answer to the ϵ-test on B is 'Yes'.

Let us first analyze the query complexity of the above test: As was remarked before, each B_i is either of $O(1)$ length or non ϵ_1-decomposable. Hence, for each chosen i, an ϵ_1-test for each start and accept nodes $(u, v) \in L_{a_{i-1}} \times L_{a_i}$ can either be done in $O(1)$ queries (in the former case) or it can be done by calling Algorithm A_2 for non-decomposable BP's. Note that in the latter case, Corollary 3.11 asserts that one call to A_2 provides a test for each start and accept node.

Since there are at most $4/\epsilon$ calls for A_2 (with $\delta = \epsilon_1$), the total complexity is

$$\tilde{q}(\epsilon, w) \leq \frac{4}{\epsilon} \cdot O(\frac{w^4}{\epsilon_1^3}(\log \frac{w^2}{\epsilon_1})^2) \cdot \tilde{q}(0.8\epsilon_1, w-1) = O(\frac{w^4}{\epsilon^4}(\log \frac{w^2}{\epsilon})^2) \cdot \tilde{q}(0.4\epsilon, w-1),$$

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

Let us check the error probability of this algorithm. If for an input $x \in \{0,1\}^n$, $dist(x, B[s:t]) = 0$, then for every $i \in I$ that is chosen in the algorithm above, $dist(x, B[u:v]) = 0$ for some $(u, v) \in L_{a_{i-1}} \times L_{a_i}$. Hence the answer will be 'Yes' with probability 1.

For an input x such that $dist(x, B[s:t]) \geq \epsilon n$, Claim 3.13 implies that $\Sigma_i dist(x, B_{a_i:a_{i+1}}) \geq \epsilon n - \Sigma_i r_i$. However, as $r_i \leq \frac{\epsilon \cdot l_i}{20}, i = 1, ..., h-1$ and $r_h \leq max\{1, \frac{\epsilon \cdot l_h}{20}\}$ we conclude that $\Sigma r_i \leq \frac{\epsilon n}{20} + 1$ and hence $\Sigma dist(x, B_{a_i:a_{i+1}}) \geq \frac{94}{100}\epsilon n$ for large enough n. Thus, by sampling one $i \in \{1, ..., h\}$ as above, we get that $dist(x, B_{a_i:a_{i+1}}) \geq \frac{1}{2}\epsilon l_i = \epsilon_1 l_i$ with probability at least 0.44ϵ. To see this let $D = \{i |\ dist(x, B_{a_{i-1}:a_i}) \geq \frac{1}{2}\epsilon l_i\}$ and let $d_i = dist(x, B_{a_{i-1}:a_i})$, then

$$\frac{94}{100}\epsilon n \ \leq\ \Sigma_{i \in D} d_i + \Sigma_{i \notin D} d_i \ \leq\ \Sigma_{i \in D} l_i + \frac{1}{2}\epsilon \Sigma_{i \notin D} l_i \ \leq\ Prob(i \in D) \cdot n + \frac{1}{2}\epsilon n$$

which implies that $Prob(i \in D) \geq 0.44\epsilon$.

Assuming that $i \in D$, then for every $u \in L_{a_{i-1}}, v \in L_{a_i}$, $dist(x, B[u:v]) \geq \epsilon_1 l_i$. Thus, the success probability for a chosen i is at least $0.44\epsilon \cdot \frac{2}{3}$. Namely, that $i \in D$ and that the ϵ_1-test on B_i answers 'No' as it should, for at least one pair of start and accept nodes of B_i. Making $4/\epsilon$ independent such tests will, again, reduce the error probability to below $\frac{1}{3}$. ∎

3.4 Time Complexity

We end up this section with a note on the total running time of the algorithm. Every fixed BP, B, defines a property $\mathcal{P}_B \subseteq \{0,1\}^n$. We have presented in the section above an 'algorithm scheme'. Namely, it produces an ϵ-test for any given ϵ and w-width oblivious read-once BP, B. For the algorithm-scheme, the input is ϵ and B, while for the property tester the input is $x \in \{0,1\}^n$. These two notions should not be confused. Thus, in analyzing the running time of the ϵ-test of $\mathcal{P}_B$, for a given BP, B, we may assume that we have at hand the decomposition of B into non-decomposable parts for all possible recursion levels. We also assume that we have $F(l)$ for every level l and for every possible subprogram that is considered in any of the recursion levels. We don't discuss how this data is represented or computed, which is out of the scope of this paper. We note however that, by computing all-pairs-connectivity, the data above can easily be obtained. Hence the above can be done in polynomial time (in the length of B and $1/\epsilon$).

For an input $x \in \{0,1\}^n$, the operations in a given recursion level involve sampling a decomposable subprogram, calling A_1 and A_2, and processing the return answers of Algorithms A_1 and A_2. Sampling one decomposable program takes $O(\log n)$ steps since there might be $O(n)$ non-decomposable B_i's in the top level. Once all calls to A_1 are done, computing the outcome of Algorithm A_2, for a w-width BP in the top recursion level, is done by forming the graph G and then checking whether s can reach t in G. Given the answers of the calls to A_1, preparing the graph G takes $O(pw^2) = O(\frac{w^2}{\epsilon^2})$ steps. Then, solving the connectivity problem on G takes $O(\frac{w^2}{\epsilon^2})$ steps. Putting this together yields the following recursion for the time $t(\epsilon, w, n)$ where ϵ is the distance parameter, w is the width and n is the length of the BP:

$$t(\epsilon, w, n) = \frac{4}{\epsilon} \cdot [O(\log n) + O(\frac{w^2}{\epsilon^2}) + O(\frac{w^2}{\epsilon^2} \log \frac{w}{\epsilon}) \cdot O(\frac{w^2}{\epsilon} \cdot \log \frac{w^2}{\epsilon}) \cdot t(0.4\epsilon, w-1, n)].$$

The $\frac{4}{\epsilon}$ term comes from the number of i's chosen in the top level general test. The $\log n$ comes from sampling one i. The $O(\frac{w^2}{\epsilon^2})$ term comes from deciding the connectivity in A_2, and the rest comes from A_1 multiplied by the number of calls to it from A_2.

Solving the above yields $t(\epsilon, w, n) = (\frac{2^w \cdot \log n}{\epsilon})^{O(w)}$.

4 Examples of some interesting functions and open problems

We present here some examples of functions that have narrow width, read-once BP's and are 'efficiently' testable (sometimes, 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 is conceptually different than that 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 ϵ is worse.

Other very simple families are k-term-DNF and k-clauses-CNF, each having 2^k-width oblivious read-once BP. A function $g : \{0,1\}^n \longrightarrow \{0,1\}$ is k-term-DNF if it has a DNF representation (a disjunction of terms where each is a conjunction of literals) with at most k terms. Analogously, a k-clause CNF is defined. Two remarks are due here: For both k-term-DNF and k-clause-CNF ϵ-tests are known (folklore): k-term DNF is $(\epsilon, O(\frac{k \log k}{\epsilon}))$-testable by testing for each term separately. k-term CNF can be tested by 0 queries for any $\epsilon > \frac{k}{n}$ as such function is either constant or every input has distance at most k to a satisfiable one.

It is also interesting to note that 1-term-DNF includes examples of functions that are, say, in uniform $SPACE(\log \log n)$ but not regular and hence not in $SPACE(o(\log \log n))$. One such interesting example is the following example of Papadimitriou [14]: Let b be 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 $w = 1$. 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.

In view of Theorem 1 one may ask what is the true dependence of ϵ-testing w-width read-once BP's on w and ϵ. 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, say, less than n^δ for any $\delta > 0$). Currently we do not have any candidate for a counter example to this.

Another issue is how far the current result may be generalized. One restriction that may be considered is being 'read-once' – can this be replaced by, say, polynomial total size? To this, the answer is false: Barrington [4], has proved that every NC^1 function has a polynomial length oblivious 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 testable. A negative answer is given in [9]: They show that there is a Boolean function $g : \{0,1\}^n \longrightarrow \{0,1\}$ that is computed by a read-twice constant width oblivious BP and that is not ϵ-testable for some fixed $\epsilon > 0$ (a read-k-times BP is a BP where each variable appears in at most k levels).

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. Beame, 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] A. Cobham, The recognition problem for the set of perfect squares, 7th IEEE Symp. on Switching and Automata Theory, 78-87, 1966.

[9] E. Fischer and I. Newman, Functions that have read-twice, constant width, branching programs are not necessarily testable, to appear in COMPLEXITY 2002.

[10] 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.

[11] O. Goldreich, S. Goldwasser and D. Ron, Property testing and its connections to learning and approximation. J. of the ACM, 45 (1998), 653–750.

[12] J. Hartmanis, P.L. Lewis II, R.E. Stearns, Hierarchies of memory-limited computations, Proc. 6th IEEE Symp. on Switching Circuits and Logic Design (1965), 179-190.

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

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

[15] M. Parnas, D. Ron and R. Rubinfeld, Testing Parenthesis Languages, in Proc. of APPROX/RANDOM 2001, Springer-Verlag LNCS 2129 (2001), 261-272.

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