

Quantum algorithms for testing bipartiteness and expansion of bounded-degree graphs (preliminary version)

Andrew M. Childs
University of Waterloo

Yi-Kai Liu
California Institute of Technology

October 22, 2009

Abstract

We study quantum algorithms for testing properties of sparse graphs. Let G be a bounded-degree graph on N vertices. Given an oracle that answers queries of the form “what is the i th neighbor of vertex v ?”, the problem is to distinguish whether G is bipartite or far from bipartite. A result of Goldreich and Ron shows that the classical query complexity of this task is $\Omega(\sqrt{N})$. We present a quantum algorithm that solves this problem using $\tilde{O}(N^{1/3})$ queries, giving a polynomial speed-up. Our algorithm combines techniques from classical property testing with the optimal quantum algorithm for element distinctness. In order to obtain a quantum algorithm whose running time is also $\tilde{O}(N^{1/3})$, we derandomize a corresponding classical randomized algorithm, using $O(\log N)$ -wise independent bits. We show similar results for the problem of testing graph expansion, which also has an $\Omega(\sqrt{N})$ classical lower bound.

1 Introduction

In the framework of *property testing*, for some given property P , one is asked to distinguish between objects that satisfy P and objects that are, in some sense, far from satisfying P . The goal is to design algorithms that test properties in sublinear or even constant time, and in particular, without reading the entire input. (Note that this is nontrivial even in cases where the property can be computed in polynomial time.) This class of problems is motivated by the practical question of how to extract meaningful information from massive data sets that are too large to fit in a single computer’s memory and can only be handled in small pieces.

Testing properties of graphs is an interesting special case. Here, the graph can be specified by an adjacency matrix (most suitable for dense graphs) or by a collection of adjacency lists (for bounded-degree graphs). Many graph properties, such as connectivity, planarity, etc., can be tested in constant time, independent of the size of the graph [13, 16]. However, there are exceptions: in the adjacency-list representation, testing bipartiteness and testing expansion of N -vertex graphs both require $\Omega(\sqrt{N})$ queries [16]. These bounds are nearly tight, since there are classical algorithms using $\tilde{O}(\sqrt{N})$ queries both for testing bipartiteness [14] and for testing expansion [11, 15, 19, 22] (where the tilde indicates that we suppress logarithmic factors).

To date, there has been only limited work on *quantum* property testing. Buhrman et al. gave an example of a language that can be tested in a constant number of quantum queries, yet that requires $\Omega(\log N)$ classical queries [7]. They also constructed a language—motivated

by Simon’s problem [23]—that gives an exponential separation between quantum and classical property testing, and showed that some languages cannot be efficiently tested quantumly. Atici and Servedio gave quantum algorithms for learning and testing juntas [5], and Inui and Le Gall gave an efficient quantum algorithm for testing solvability of a black-box group [18]. Most recently, Bravyi et al. showed quantum algorithms for testing uniformity and orthogonality of distributions, with lower query complexity than is possible in the classical case [8]. However, we are not aware of any previous work on quantum algorithms for testing graph properties.¹

Why might we expect quantum computers to offer an advantage for graph property testing? The classical algorithms for testing bipartiteness and expansion in sparse graphs are based on exploring the graph by multiple random walks starting from the same vertex and checking for correlations among the walks. Quantum analogues of random walks provide a powerful tool for searching graphs [3, 4, 9, 10, 21, 24], so we might expect an application of quantum walks to be fruitful. But we actually do not know if such an approach might be advantageous.

Instead, we use quantum walks indirectly. The classical algorithm for testing bipartiteness is based on checking whether a pair of short random walks form an odd-length cycle in the graph, thereby certifying non-bipartiteness [14]. The algorithm for testing expansion looks for collisions between the endpoints of short random walks, with a large number of collisions indicating that the walk is not rapidly mixing [15]. In both cases, the property can be tested by looking for collisions among a set of $\tilde{O}(\sqrt{N})$ items. Essentially, by applying a quantum walk algorithm for element distinctness [3, 21] to look for these collisions, we reduce the running time to $\tilde{O}(N^{1/3})$.

In the case of testing expansion, it is not sufficient to simply decide whether the number of collisions is nonzero; rather, we must determine whether the number of collisions is above or below some threshold. In general, we cannot hope to exactly count the collisions since this would require too many queries [6]. Nevertheless, we are able to test expansion by counting the collisions one at a time, since it turns out that we need to count at most a small number.

Another obstacle—in this case, to testing both bipartiteness and expansion—is that to obtain a running time of $\tilde{O}(N^{1/3})$, we cannot hope to explicitly store the $\Omega(\sqrt{N})$ random bits that determine the trajectories of the random walks. We overcome this issue by showing that it suffices to perform the walks using $O(\log N)$ -wise independent bits. This is the first instance we are aware of in which the success of a quantum algorithm is predicated on the derandomization of a classical algorithm.

In order to state our results more precisely, we define the model of quantum property testing. In the adjacency-list model for bounded-degree graphs, we fix a constant upper bound d on the degree of any vertex. Then a graph $G = (V, E)$ is represented by a function $f_G : V \times \{1, \dots, d\} \rightarrow V \cup \{*\}$, where $f_G(v, i)$ returns the i th neighbor of v in G , or $*$ if v has fewer than i neighbors. Whereas a classical computer would be allowed to query $f_G(v, i)$ for any desired $v \in V$ and $i \in \{1, \dots, d\}$, a quantum computer is provided with a unitary black box that reversibly computes f_G as $|v, i, z\rangle \mapsto |v, i, z \oplus f_G(v, i)\rangle$.

For bounded-degree graphs, we say that G is ε -far from satisfying graph property P if one must change at least εnd edges of G in order to satisfy P . We say that an algorithm ε -tests P if it accepts graphs that satisfy P with probability at least $2/3$, and rejects graphs that are ε -far from satisfying P with probability at least $2/3$. The query complexity of an algorithm for ε -testing P is the number of calls it makes to the black box for f_G . More generally, we may consider algorithms that determine whether a graph satisfies P or is ε -far from satisfying a related property P' , again

¹We remark that quantum speed-ups are known for *deciding* certain graph properties, without the promise that the graph either has the property or is far from having it [12, 20]. This turns out to be a fairly different setting, and the results there are not directly comparable to ours.

with bounded error.

Recall that a graph is bipartite if and only if it does not contain a cycle of odd length. We say that a graph G is an α -vertex-expander if for every $U \subseteq V$ with $|U| \leq |V|/2$, we have $|\partial(U)| \geq \alpha|U|$, where $\partial(U)$ denotes the vertex boundary of U , the set of vertices in $V \setminus U$ adjacent to at least one vertex of U .

Our two main results are as follows:

- There is a quantum algorithm for ε -testing bipartiteness in time $O(N^{1/3} \text{poly}(\log N, 1/\varepsilon))$.
- For any degree bound d , there is a constant c , such that there exists a quantum algorithm (with parameter $0 < \mu < \frac{1}{4}$) for deciding whether a graph is an α -vertex-expander or is ε -far from any $(c\mu\alpha^2)$ -vertex expander, in time $O(N^{(1/3)+3\mu}(d^2/\varepsilon\alpha^2) \text{poly}(\log N) \log(d/\alpha))$.

In particular, the above upper bounds on the running times also imply bounds on the query complexities.

Despite this progress, the actual quantum query complexities of testing bipartiteness and expansion remain unresolved. On the one hand, the $\Omega(\sqrt{N})$ lower bounds show that our quantum algorithms outperform any classical testers. On the other hand, we are not aware of any better-than-constant quantum lower bound for these problems. The standard quantum adversary method suffers from a property testing barrier [17], ruling out one potential approach to showing that our algorithms are optimal. It would be interesting to prove a nontrivial quantum lower bound for either of these testing problems. In general, much remains to be discovered about the quantum complexity of testing graph properties.

2 Testing bipartiteness

2.1 The classical algorithm

We first recall the classical algorithm for testing bipartiteness [14]. This algorithm is based on the observation that a bipartite graph contains no odd-length cycles, whereas if a graph is far from bipartite, then it contains many short odd-length cycles. The algorithm tries to find an odd-length cycle, by running several random walks from a common starting vertex s , and looking for “collisions” where two walks reach the same vertex v , one after an even number steps, the other after an odd number of steps.

Given: an oracle f_G , specifying a graph G with N vertices and maximum degree d , and accuracy parameter ε .

Repeat the following procedure T times, for some $T = \Theta(1/\varepsilon)$:

Pick a random vertex s .

For $i = 1, \dots, K$, where $K = \text{poly}(\frac{\log N}{\varepsilon})\sqrt{N}$:

Starting from s , do a random walk of length $L = \text{poly}(\frac{\log N}{\varepsilon})$.

(The random walk proceeds as follows: at vertex v , for each adjacent vertex u , move to u with probability $\frac{1}{2d}$; stay at v with probability $1 - \frac{\deg(v)}{2d}$.)

Let $(w_{ij})_{j=0,1,2,\dots}$ be the sequence of vertices visited during the walk, omitting consecutive repetitions of the same vertex.

(I.e., when the walk stays at the same vertex for more than one time step, we only include that vertex once in the sequence. So $w_{ij} \neq w_{i(j+1)}$.)

End.

If $w_{ij} = w_{i'j'}$ for some i, j, i', j' , where j is even and j' is odd, then return “false.”

End.

If none of the above iterations returned “false,” then return “true.”

It was shown in [14] that this algorithm always returns “true” when G is bipartite, and returns “false” with probability $\geq 2/3$ when G is ε -far from bipartite. In addition, the algorithm uses $\text{poly}(\frac{\log N}{\varepsilon})\sqrt{N}$ oracle queries, and has running time $\text{poly}(\frac{\log N}{\varepsilon})\sqrt{N}$.

2.2 Derandomization

Our first step is to partially derandomize the above algorithm, by substituting in k -wise independent random variables. Intuitively, this is possible because the algorithm (and the analysis of its performance) only depend on the behavior of pairs of random walks, which are determined by subsets of $\sim \text{poly}(\log N)$ random bits. Derandomization allows us to reduce the number of random bits from $\sim \sqrt{N} \text{poly}(\log N)$ to $\sim \text{poly}(\log N)$. This will turn out to be important later on, in reducing the running time of our quantum algorithm.

We will use the following simple construction for k -wise independent random variables (Prop. 6.5 in [1]):

Proposition 1. *Suppose $n + 1$ is a power of 2 and k is odd, $k \leq n$. Then there exists a uniform probability space $\Omega = \{0, 1\}^m$ where $m = 1 + \frac{1}{2}(k-1) \log_2(n+1)$, and there exist k -wise independent random variables ξ_1, \dots, ξ_n over Ω , such that $\Pr[\xi_j = 1] = \Pr[\xi_j = 0] = \frac{1}{2}$.*

Furthermore, there exists an algorithm that, given $i \in \Omega$ and $1 \leq j \leq n$, computes $\xi_j(i)$ in time $O(k \log_2 n)$.

Note that even more efficient constructions are possible for random variables that are *almost* k -wise independent [2]. We forego this extra improvement for the time being.

The classical algorithm for testing bipartiteness uses $O(\sqrt{N} \text{poly}(\frac{\log N \log d}{\varepsilon}))$ bits of randomness. (Note that this involves a minor technical issue, because we need to choose uniformly among $2d$ outcomes (for the random walk), and when d is not a power of 2, we have to approximate the desired distribution. This can be handled using standard techniques [1].) We will derandomize each of the T repetitions of the algorithm separately.

We claim that it suffices to use k -wise independent random bits, for some $k = O(\text{poly}(\frac{\log N \log d}{\varepsilon}))$. To see this, note that the analysis of this algorithm in [14] only depends on the behavior of groups of up to 4 random walks. Specifically, their proof first establishes various properties of

an individual random walk, then uses a second moment argument to bound the random variable $X = \sum_{i < j} \eta_{ij}$ (where η_{ij} is a 0/1 random variable that indicates whether walk i collides with walk j). This depends on the expectation and variance of X , which can be written as sums of terms that involve 2 and 4 random walks, respectively. So these terms involve at most $O(\text{poly}(\frac{\log N \log d}{\epsilon}))$ random bits, and they will be unchanged if we use k -wise independent random bits, where k is chosen to be this large.

Thus, we can substitute k -wise independent random variables, constructed using Prop. 1, with $n = O(\sqrt{N} \text{poly}(\frac{\log N \log d}{\epsilon}))$ and $k = O(\text{poly}(\frac{\log N \log d}{\epsilon}))$. This reduces the number of random bits required by the algorithm to $O(k \log_2 n) = O(\text{poly}(\frac{\log N \log d}{\epsilon}))$.

2.3 A quantum algorithm

We now give a quantum algorithm for testing bipartiteness. Let us first describe the basic idea. We will run several random walks starting from the same vertex s , and use Ambainis' algorithm for element distinctness [3] to find "collisions" between these walks. Actually, we will use a slight variant of Ambainis' algorithm [24] that solves a more general problem: given a function $f : X \rightarrow Y$, and a relation $R \subseteq Y \times Y$, find two distinct elements $x, x' \in X$ such that $(f(x), f(x')) \in R$. In our application, each element in X will be a sequence of coin-tosses; the function f will compute the end-point of the corresponding walk in the graph, together with the number of steps along the way; and the relation R will test whether two walks reach the same vertex, one after an even number of steps, the other after an odd number of steps.

The overall picture is that we are searching for collisions among $\tilde{O}(\sqrt{N})$ elements, so we require $\tilde{O}(N^{1/3})$ evaluations of the function f . (The \tilde{O} notation suppresses log factors.) Moreover, f can be computed using only $\text{poly}(\frac{\log N}{\epsilon})$ queries to the oracle for the graph. So the quantum algorithm will use $O(N^{1/3} \text{poly}(\frac{\log N}{\epsilon}))$ queries to the graph oracle.

Finally, it is now clear why derandomizing the classical algorithm is useful: it allows a concise representation of the set of elements. Rather than enumerating them explicitly, which would take $\tilde{O}(\sqrt{N})$ time, we can describe and manipulate them in time $\text{poly}(\frac{\log N}{\epsilon})$. So the quantum algorithm will run in time $O(N^{1/3} \text{poly}(\frac{\log N}{\epsilon}))$.

We now describe the details. We will use the following result from [3, 24]:

Theorem 2. *Let X and Y be finite sets. Say we are given oracle access to a function $f : X \rightarrow Y$, and let $R \subseteq Y \times Y$ be a binary relation, which we can compute in time $\text{poly}(\log |Y|)$. Let $0 \leq \alpha \leq 1$, and define*

$$p_{f,\alpha} = \Pr_H[\exists \text{ distinct } x, x' \in H \text{ s.t. } (f(x), f(x')) \in R], \tag{1}$$

where H is a random subset of X of size $|X|^\alpha$. Let $\epsilon > 0$. Then there is a quantum algorithm (with oracle f) that returns "true" with constant probability when $p_{f,\alpha} \geq \epsilon$, always returns "false" when $p_{f,\alpha} = 0$, and runs in time

$$O((|X|^\alpha + 1000\sqrt{|X|^\alpha/\epsilon}) \cdot \text{poly}(\log |Y|)). \tag{2}$$

An immediate corollary is:

Corollary 3. *There is a quantum algorithm (with oracle f) that returns "true" with constant probability when there exist distinct $x, x' \in X$ such that $(f(x), f(x')) \in R$, always returns "false" when such a collision does not exist, and runs in time $O(|X|^{2/3} \cdot \text{poly}(\log |Y|))$.*

Our quantum algorithm for testing bipartiteness is as follows:

Given: an oracle f_G , specifying a graph G with N vertices and maximum degree d , and accuracy parameter ε .

Repeat the following procedure T times, for some $T = \Theta(1/\varepsilon)$:

Pick a random vertex s .

Let $K = \text{poly}(\frac{\log N}{\varepsilon})\sqrt{N}$ and $L = \text{poly}(\frac{\log N}{\varepsilon})$.

Let $n = KL$ and $k = \Theta(L)$. Using Prop. 1, construct probability space Ω and k -wise independent random variables b_{ij} taking values in $\{0, 1, \dots, 2d - 1\}$ (for $i = 1, \dots, K$ and $j = 1, \dots, L$).

Choose $\omega \in \Omega$ uniformly at random.

Let $X = \{1, \dots, K\} \times \{1, \dots, L\}$.

Let $Y = \{1, \dots, N\} \times \{0, 1\}$.

Define $f : X \rightarrow Y$ as follows:

Given: (i, j) .

Run a random walk in G , starting at s , with random coin-flips $(b_{i1}(\omega), \dots, b_{ij}(\omega))$.

Let v be the end-point of the walk.

Let q be the number of steps taken in the graph, not counting steps where the random walk chooses to stay at its current location.

Return $(v, q \bmod 2)$.

Define $R \subseteq Y \times Y$ as: $((v, c), (v', c')) \in R$ iff $(v = v' \text{ and } c \neq c')$.

Run the algorithm from Corollary 3. If it finds a collision, return “false.”

End.

If none of the above iterations returned “false,” then return “true.”

Note that there is a slight subtlety in the subroutine for computing f . We actually need to implement the unitary operation $U_f : |x\rangle|z\rangle \mapsto |x\rangle|z \oplus f(x)\rangle$, which is required by the quantum algorithm for element distinctness. We have access to the unitary operation $U_{f_G} : |v, i\rangle|w\rangle \mapsto |v, i\rangle|w \oplus f_G(v, i)\rangle$, provided by the oracle f_G . Note that computing f only requires classical operations and queries to f_G . So we can perform U_f by using reversible classical computation and applying U_{f_G} .

Theorem 4. *The above algorithm always returns “true” when G is bipartite, returns “false” with constant probability when G is ε -far from bipartite, and runs in time $O(N^{1/3} \text{poly}(\frac{\log N}{\varepsilon}))$.*

Proof: Our quantum algorithm follows the same strategy as the classical algorithm of [14]: it looks for a “collision” where two random walks reach the same vertex v , one after an even number of steps, the other after an odd number of steps.

When G is bipartite, it has no odd-length cycles, so the algorithm never finds a collision. Thus the algorithm returns “true.”

When G is ε -far from bipartite, the analysis of [14] implies that, with constant probability, one of the sets of random walks sampled by the algorithm contains a collision. So the algorithm returns “false.”

For the bound on the running time, note that evaluating the k -wise independent random variables $b_{ij}(\omega)$ takes time $O(k \log n) = O(\text{poly}(\frac{\log N}{\varepsilon}))$. Also, each evaluation of the function f takes time $\text{poly}(\frac{\log N}{\varepsilon})$. Since X has size $O(\sqrt{N} \text{poly}(\frac{\log N}{\varepsilon}))$, finding a collision takes time $O(N^{1/3} \text{poly}(\frac{\log N}{\varepsilon}))$. \square

3 Testing expansion

3.1 The classical algorithm

The classical algorithm for testing expansion [15] is as follows. The basic idea is to test how rapidly a random walk from some starting vertex s converges to the uniform distribution. This can be done by running several random walks starting from s and counting the number of collisions among their endpoints—the number of collisions is smallest when the distribution is uniform.

Given: an oracle f_G , specifying a graph G with N vertices and maximum degree d , accuracy parameter ε , expansion parameter α , and running time parameter μ .
Repeat the following procedure T times, for some $T = \Theta(1/\varepsilon)$:

- Pick a random vertex s .
- For $i = 1, \dots, K$, where $K = N^{(1/2)+\mu}$:
 - Starting from s , do a random walk of length $L = (16d^2/\alpha^2) \log N$.
 - (The random walk proceeds as follows: at vertex v , for each adjacent vertex u , move to u with probability $\frac{1}{2d}$; stay at v with probability $1 - \frac{\deg(v)}{2d}$.)
 - Let w_i be the endpoint of the walk.
- End.
- Let X be the number of pairwise collisions among the vertices w_1, \dots, w_K .
- Let $M = \frac{1}{2}N^{2\mu} + \frac{1}{128}N^{(1.75)\mu}$.
- If $X > M$, then return “false.”

End.
If none of the above iterations returned “false,” then return “true.”

The following performance guarantee is known for this algorithm [11, 19, 22]:

Theorem 5. *Assume $d \geq 3$, $0 < \alpha < 1$, and $0 < \mu < \frac{1}{4}$. The above algorithm runs in time $O(N^{(1/2)+\mu} \log N \cdot 16d^2/\varepsilon\alpha^2)$. Furthermore, there exists a constant $c > 0$ (which depends on d) such that, for any $0 < \varepsilon < 1$:*

1. *If G is an α -vertex expander, then the algorithm returns “true” with probability at least $2/3$.*
2. *If G is ε -far from any $(c\mu\alpha^2)$ -vertex expander of degree at most d , then the algorithm returns “false” with probability at least $2/3$.*

3.2 Derandomization

Next, we partially derandomize the above algorithm using k -wise independent random variables, with similar motivation and techniques as for testing bipartiteness. Note that the algorithm originally requires $O(N^{(1/2)+\mu}(16d^2/\alpha^2) \log N \log d)$ random bits for each of the T repetitions. As for bipartiteness, we derandomize each of the repetitions separately, with each iteration independent of the others. We observe that the proof of Theorem 5 works by analyzing properties of a single random walk, then using a second moment argument on the random variable $X = \sum_{i < j} \eta_{ij}$ (where η_{ij} indicates whether walk i collides with walk j). This only depends on correlations among up to 4 random walks. Since the random walks are short, of length $O((d^2/\alpha^2) \log N)$, these correlations involve subsets of at most $O((d^2/\alpha^2) \log N \log d)$ random bits. Thus, we can substitute k -wise independent random bits, where k is chosen to be this large; the algorithm then requires only $O((d^2/\alpha^2) \log N \log d)$ bits of randomness for each of the T repetitions. This will improve the running time of our quantum algorithm, below.

3.3 A quantum algorithm

We now describe a quantum algorithm for testing expansion. The basic idea is similar to that for bipartiteness, with one additional detail: we run several random walks from the same starting vertex and use a quantum algorithm to *count* the number of collisions among the endpoints of the walks. More precisely, we need to determine whether the number of collisions is greater or less than M , for small values of M . We do this by explicitly finding up to M collisions, one at a time; in particular, we run the element distinctness algorithm [3] many times, modifying the relation R at each step to exclude collisions which have been found previously.

We will use the following result, which gives a quantum algorithm for finding collisions [3,21]:

Theorem 6. *Let X and Y be finite sets. Say we are given oracle access to a function $f : X \rightarrow Y$, and let $R \subseteq Y \times Y$ be a binary relation, which we can compute in time $\text{poly}(\log |Y|)$. We define a “collision” to be a distinct pair $x, x' \in X$ such that $(f(x), f(x')) \in R$. Then there is a quantum algorithm (with oracle f) that finds a collision with constant probability when collisions exist, always returns “false” when there are no collisions, and runs in time $O(|X|^{2/3} \cdot \text{poly}(\log |Y|))$.*

This leads to a simple algorithm for counting collisions, which is efficient when the number of collisions is small:

Given: a set X , an oracle $f : X \rightarrow Y$, a relation $R \subseteq Y \times Y$, and a number M .
 Initialize $S = \emptyset$.
 For $i = 1, \dots, M$:
 Repeat the following procedure up to t times, for some $t = \Theta(\log M)$:
 Run the algorithm of Theorem 6 to find some distinct $x, x' \in X$ such that $(f(x), f(x')) \in R$ and $(x, x') \notin S$.
 If the algorithm finds a collision (x, x') , then set $S = S \cup \{(x, x'), (x', x)\}$ and break out of this loop.
 End.
 If the algorithm did not find a collision on any of the t tries, then return “false.”
 End.
 If the algorithm did not return “false” on any of the M iterations, then return “true.”

Corollary 7. *The above algorithm returns “true” with constant probability if there are M or more collisions, always returns “false” if there are strictly fewer than M collisions, and runs in time $O(M \log M \cdot |X|^{2/3} \cdot \text{poly}(\log |Y|))$.*

Proof. Suppose there are M or more collisions. Then, in each iteration ($i = 1, \dots, M$), there are collisions to be found. Consider what happens in iteration i . Say that the algorithm from Theorem 6 returns “false” with probability at most p (some constant). We will try running the algorithm $t = \log_{1/p}(3M)$ times. The probability that it returns “false” on every attempt is at most $p^t = \frac{1}{3M}$. So the probability that we return “false” during iteration i is at most $\frac{1}{3M}$, and by the union bound, the probability that we return “false” is at most $1/3$.

The other claims are easy to see. □

We now have the following quantum algorithm for testing expansion:

Given: an oracle f_G , specifying a graph G with N vertices and maximum degree d , accuracy parameter ε , expansion parameter α , and running time parameter μ .

Repeat the following procedure T times, for some $T = \Theta(1/\varepsilon)$:

Pick a random vertex s .

Let $K = N^{(1/2)+\mu}$ and $L = (16d^2/\alpha^2) \log N$.

Let $n = KL$ and $k = \Theta(L)$. Using Prop. 1, construct probability space Ω and k -wise independent random variables b_{ij} taking values in $\{0, 1, \dots, 2d - 1\}$ (for $i = 1, \dots, K$ and $j = 1, \dots, L$).

Choose $\omega \in \Omega$ uniformly at random.

Let $X = \{1, \dots, K\}$.

Let $Y = \{1, \dots, N\}$.

Define $f : X \rightarrow Y$ as follows:

Given i , return the end-point of the random walk in G

that starts at s and uses random coin-flips $(b_{i1}(\omega), \dots, b_{iL}(\omega))$.

Define $R \subseteq Y \times Y$ as: $(v, v') \in R$ iff $v = v'$.

Let $M = \frac{1}{2}N^{2\mu} + \frac{1}{128}N^{(1.75)\mu}$.

Repeat the following procedure t times, for some $t = \Theta(1)$:

Run the algorithm of Corollary 7, to test whether there are $M + 1$ or more collisions.

If the algorithm returns “true,” then return “false.”

End.

End.

If none of the above iterations returned “false,” then return “true.”

Theorem 8. *The above algorithm runs in time $O(N^{(1/3)+3\mu} \text{poly}(\log N) \cdot (d^2/\varepsilon\alpha^2) \log(d/\alpha))$. Furthermore, there exists a constant $c > 0$ (which depends on d) such that, for any $0 < \varepsilon < 1$:*

1. *If G is an α -vertex-expander, then the algorithm returns “true” with probability at least $2/3$.*
2. *If G is ε -far from any $(c\mu\alpha^2)$ -vertex-expander of degree at most d , then the algorithm returns “false” with probability at least 0.6 .*

Proof. Suppose G is an α -vertex-expander. Then, with probability at least $2/3$, in each of the T repetitions, the number of collisions is at most M . When this happens, the collision-counting algorithm always returns “false,” so we return “true.”

Suppose G is ε -far from any $(c\mu\alpha^2)$ -vertex-expander of degree at most d . Then, with probability at most $2/3$, in at least one of the T repetitions, the number of collisions is at most $M + 1$. When this happens, the collision-counting algorithm returns “true” with probability at least p (some constant). We try running the collision-counting algorithm t times, where $t = \log_{1/(1-p)} 10$. The probability that this returns “false” every time is at most $(1 - p)^t = 1/10$. So, with probability at least $9/10$, the collision-counting algorithm returns “true” at least once, and thus we return “false.”

The bound on the running time is straightforward. In particular, implementing the k -wise independent random variables b_{ij} takes time and space $O(k \log n) = O((d^2/\alpha^2) \text{poly}(\log N) \log(d/\alpha))$. Also, note that the only time we query the graph oracle f_G is when we are evaluating the function f . Evaluating f requires $L = O((d^2/\alpha^2) \log N)$ queries to f_G , and the collision-counting algorithm requires $O(N^{(1/3)+3\mu} \text{poly}(\log N))$ evaluations of f . \square

Acknowledgments

We thank the Kavli Institute for Theoretical Physics for its hospitality. This research was supported in part by the National Science Foundation under Grant No. PHY05-51164. AMC also received support from MITACS, NSERC, QuantumWorks, and the US ARO/DTO. YKL is supported by an NSF postdoctoral fellowship and ARO/NSA.

References

- [1] N. Alon, L. Babai, and A. Itai, *A fast and simple randomized parallel algorithm for the maximal independent set problem*, Journal of Algorithms **7** (1986), no. 4, 567–583.
- [2] N. Alon, O. Goldreich, J. Håstad, and R. Peralta, *Simple constructions of almost k -wise independent random variables*, Random Structures and Algorithms **3** (1992), no. 3, 289–304. Preliminary version in FOCS 1990.
- [3] A. Ambainis, *Quantum walk algorithm for element distinctness*, SIAM Journal on Computing **37** (2007), no. 1, 210–239, available at quant-ph/0311001. Preliminary version in FOCS 2004.
- [4] A. Ambainis, J. Kempe, and A. Rivosh, *Coins make quantum walks faster*, Proc. 16th ACM-SIAM Symposium on Discrete Algorithms, 2005, pp. 1099–1108, available at quant-ph/0402107.
- [5] A. Atici and R. Servedio, *Quantum algorithms for learning and testing juntas*, Quantum Information Processing **6** (2007), no. 5, 323–348, available at arXiv:0707.3479.
- [6] H. Buhrman, C. Durr, M. Heiligman, P. Høyer, F. Magniez, M. Santha, and R. de Wolf, *Quantum algorithms for element distinctness*, SIAM Journal on Computing **34** (2005), no. 6, 1324–1330, available at quant-ph/0007016. preliminary version in CCC 2001.
- [7] H. Buhrman, L. Fortnow, I. Newman, and H. Roehrig, *Quantum property testing*, SIAM Journal on Computing **37** (2008), no. 5, 1387–1400, available at quant-ph/0201117. Preliminary version in SODA 2003.
- [8] S. Bravyi, A.W. Harrow, and A. Hassidim, *Quantum algorithms for testing properties of distributions*, 2009. arXiv:0907.3920.
- [9] A. M. Childs and J. Goldstone, *Spatial search by quantum walk*, Phys. Rev. A **70** (2004), no. 2, 022314, available at quant-ph/0306054.
- [10] ———, *Spatial search and the Dirac equation*, Phys. Rev. A **70** (2004), no. 4, 042312, available at quant-ph/0405120.
- [11] A. Czumaj and C. Sohler, *Testing expansion in bounded-degree graphs*, Proc. 48th IEEE Symposium on Foundations of Computer Science, 2007, pp. 570–578.
- [12] C. Durr, M. Heiligman, P. Hoyer, and M. Mhalla, *Quantum query complexity of some graph problems*, 2004. arXiv:quant-ph/0401091.
- [13] O. Goldreich, S. Goldwasser, and D. Ron, *Property testing and its connection to learning and approximation*, Journal of the ACM **45** (1998), no. 4, 653–750. Preliminary version in STOC 1995.
- [14] O. Goldreich and D. Ron, *A sublinear bipartiteness tester for bounded degree graphs*, Combinatorica **19** (1999), no. 3, 335–373. Preliminary version in STOC 1998.
- [15] ———, *On testing expansion in bounded-degree graphs*, 2000. ECCC report TR00-020.
- [16] ———, *Property testing in bounded degree graphs*, Algorithmica **32** (2002), no. 2, 302–343. Preliminary version in STOC 1997.
- [17] P. Høyer, T. Lee, and R. Špalek, *Negative weights make adversaries stronger*, Proc. 39th ACM Symposium on Theory of Computing, 2007, pp. 526–535, available at quant-ph/0611054.
- [18] Y. Inui and F. Le Gall, *Quantum property testing of group solvability*, Proc. 8th Latin American Symposium on Theoretical Informatics, 2008, pp. 772–783, available at arXiv:0712.3829.
- [19] S. Kale and C. Seshadhri, *Testing expansion in bounded-degree graphs*, 2007. ECCC report TR07-076.
- [20] F. Magniez, M. Santha, and M. Szegedy, *Quantum Algorithms for the Triangle Problem*, SIAM Journal on Computing **37** (2007), no. 2, 413–424. Preliminary version in SODA 2005.
- [21] F. Magniez, A. Nayak, J. Roland, and M. Santha, *Search via quantum walk*, Proc. 39th ACM Symposium on Theory of Computing, 2007, pp. 575–584, available at quant-ph/0608026.

- [22] A. Nachmias and A. Shapira, *Testing the expansion of a graph*, to appear in Information and Computation. ECCC report TR07-118.
- [23] D. R. Simon, *On the power of quantum computation*, SIAM Journal on Computing **26** (1997), no. 5, 1474–1483. Preliminary version in FOCS 1994.
- [24] M. Szegedy, *Quantum speed-up of Markov chain based algorithms*, Proc. 45th IEEE Symposium on Foundations of Computer Science, 2004, pp. 32–41.