Nested Quantum Walks with Quantum Data Structures[*]

Stacey Jeffery[†1,2], Robin Kothari[‡1,2], and Frédéric Magniez[§3]

[1]David R. Cheriton School of Computer Science, University of Waterloo, Canada
[2]Institute for Quantum Computing, University of Waterloo, Canada
[3]CNRS, LIAFA, Univ Paris Diderot, Sorbonne Paris-Cité, 75205 Paris, France

Abstract

We develop a new framework that extends the quantum walk framework of Magniez, Nayak, Roland, and Santha, by utilizing the idea of quantum data structures to construct an efficient method of nesting quantum walks. Surprisingly, only classical data structures were considered before for searching via quantum walks.

The recently proposed learning graph framework of Belovs has yielded improved upper bounds for several problems, including triangle finding and more general subgraph detection. We exhibit the power of our framework by giving a simple explicit constructions that reproduce both the $O(n^{35/27})$ and $O(n^{9/7})$ learning graph upper bounds (up to logarithmic factors) for triangle finding, and discuss how other known upper bounds in the original learning graph framework can be converted to algorithms in our framework. We hope that the ease of use of this framework will lead to the discovery of new upper bounds.

1 Introduction

1.1 Background

Shenvi, Kempe, and Whaley [SKW03] first pointed out the algorithmic potential of quantum walks by designing a walk based algorithm to emulate Grover search [Gro96]. The first algorithm using quantum walks that goes beyond the capability of Grover search is due to Ambainis [Amb04] for element distinctness. In his seminal paper he resolved the quantum query complexity of the problem, settling a difficult question that had been open for several years [BDH+05, AS04]. Based on Ambainis' quantum walk, Szegedy [Sze04] developed a general theory of quantum walk based algorithms. He constructed the quantum analogue $W(P)$ of any symmetric, ergodic Markov chain P, and designed a quantum search algorithm based on the use of $W(P)$. Later, this was generalized to reversible, ergodic Markov chains P, and the construction simplified by the use of quantum phase estimation [MNRS11, MNRS12].

After a series of new quantum walk algorithms with substantially better complexity in a variety of contexts [BŠ06, NM07, MSS07, ACR+10, CK11, Gal12], a new framework was introduced by Belovs, known as learning graphs [Bel12b]. Learning graphs are a model of computation related to

span programs [Rei09]. Introduced by Reichardt in the model of quantum query algorithms, span programs have been used to characterize quantum query complexity in terms of the negative adversary method [Rei09, Rei11, LMR+11]. Whereas explicitly constructing a span program is difficult in practice, coming up with a learning graph is more easily accomplished. Indeed, Belovs [Bel12b] gave a learning graph whose query complexity improved the best known quantum query algorithms for triangle finding. Later on, other learning graphs were constructed for constant-size subgraph finding [Zhu11, LMS11], and also for k-element distinctness [BL11, Bel12a]. Recently, a simpler and more efficient learning graph was proposed for triangle finding, together with a new application to associativity testing [LMS13].

In order to get an explicit quantum algorithm, one has to transform a learning graph into a span program, and then to an algorithm. This transformation is generic but incorporates many technical difficulties. The resulting algorithm is quite complicated. For instance, the time and space complexities may not be optimal. For specific problems, such as st-connectivity [BR12], one can find explicit implementations of span programs. Still, one would like to have a framework which leads easily to the explicit algorithm and its complexity (not only query but also time and space, for instance), as is the case in the setting of search algorithms based on quantum walks.

1.2 Quantum walk based search algorithms

At an abstract level, many search problems may be cast as the problem of finding a "marked" element from a set Ω with n elements. Let $M \subseteq \Omega$ be the set of marked elements. Assume that the search algorithms maintain a data structure $\mathcal{D}$ that associates some data $\mathcal{D}(u)$ with every state $u \in \Omega$. From $\mathcal{D}(u)$, we would like to determine if $u \in M$. In this framework, Grover search, and more generally amplitude amplification, consists of the following procedure, where π is some given probability distribution on Ω.

1. Set up a superposition $|\pi\rangle = \sum_u \sqrt{\pi_u}|u\rangle$ of elements $u \in \Omega$
   Compute $\mathcal{D}$, that is $|u\rangle \mapsto |u\rangle|\mathcal{D}(u)\rangle$, for all elements u in the current superposition

2. Iterate $\frac{1}{\sqrt{\varepsilon}}$ times, where $\varepsilon = \Pr_\pi(u \in M)$.

   (a) Flip marked elements, that is $|u\rangle|\mathcal{D}(u)\rangle \mapsto -|u\rangle|\mathcal{D}(u)\rangle$, if $u \in M$

   (b) Uncompute $\mathcal{D}$, that is $|u\rangle|\mathcal{D}(u)\rangle \mapsto |u\rangle$
   Apply a reflection through $|\pi\rangle$
   Compute $\mathcal{D}$, that is $|u\rangle \mapsto |u\rangle|\mathcal{D}(u)\rangle$, for all elements u in the current superposition

3. Output u if it is marked, otherwise output 'no marked element'

When operating with $\mathcal{D}$, we distinguish two types of cost.

Setup cost S: The cost of creating $|\pi\rangle$ and of constructing the data structure $\mathcal{D}(u)$ for the state u.

Checking cost C: The cost of checking if $u \in M$ using $\mathcal{D}(u)$.

These costs may be thought of as vectors listing all the measures of complexity of interest, such as query and time complexity. Then the above algorithm has cost of order $\frac{1}{\sqrt{\varepsilon}}(\mathsf{S} + \mathsf{C})$. This above approach is a priori only valid when the probability $\Pr_\pi(u \in M)$ is known in advance. This assumption may be removed using standard techniques, without increasing the asymptotic complexity of the algorithms [BBHT98]. Indeed, if only a lower bound $\varepsilon > 0$ on $\Pr_\pi(u \in M)$ is

known for non-empty M, then the above argument can be modified in order to determine if M is empty or find an element of M.

A more cost-effective approach is to implement the reflection through $|\pi\rangle$ using the quantum analogue $W(P)$ of a reversible and ergodic walk P on a given graph G. In this way, we can ensure that u has only small changes at each step, permitting us to update its data structure $\mathcal{D}(u)$ without uncomputing/computing it. We then introduce a third type of cost.

> **Update cost U:** The cost of simulating one application of $W(P)$ according to the Markov chain P, and of updating $\mathcal{D}(u)$ to $\mathcal{D}(v)$ where $(u, v) \in G$.

Then finding a marked element, if there is one, can be done with cost of order $\mathsf{S} + \frac{1}{\sqrt{\varepsilon}}(\frac{1}{\sqrt{\delta}}\mathsf{U} + \mathsf{C})$, where δ is the spectral gap of P [MNRS11].

1.3   Our contribution

In this paper, we introduce a new model of quantum walks which can reproduce the best known learning graph upper bounds for triangle finding and related problems (up to log factors). Our framework is also simple and leads to explicit algorithms that are much simpler to analyze than learning graphs.

To achieve our goal, we introduce the notion of quantum data structures for nested quantum walks. Surprisingly, only classical data structures were considered before for searching via quantum walk. We demonstrate the potential of this framework by giving explicit and simple quantum algorithms for triangle finding, achieving the same query complexity as their learning graph analogues up to polylogarithmic factors (Section 3.4) and more generally, reproducing all subgraph containment upper bounds of [LMS13] up to polylogarithmic factors, and potentially improving on these upper bounds for certain graphs (Section 4.2).

Concretely, assume that we introduce another quantum walk based search as a checking subroutine. Let P_1 and P_2 be, respectively, the outer and inner walks, with respective costs $\mathsf{S}_1, \mathsf{U}_1, \mathsf{C}_1$ and $\mathsf{S}_2, \mathsf{U}_2, \mathsf{C}_2$, and parameters ε_1, δ_1 and ε_2, δ_2. Since the checking cost of the outer walk can be expressed as $\mathsf{C}_1 = O(\mathsf{S}_2 + \frac{1}{\sqrt{\varepsilon_2}}(\frac{1}{\sqrt{\delta_2}}\mathsf{U}_2 + \mathsf{C}_2))$ the global of cost of finding a marked element is of order (neglecting log factors)

$$\mathsf{S}_1 + \frac{1}{\sqrt{\varepsilon_1}}\left(\frac{1}{\sqrt{\delta_1}}\mathsf{U}_1 + \mathsf{S}_2 + \frac{1}{\sqrt{\varepsilon_2}}\left(\frac{1}{\sqrt{\delta_2}}\mathsf{U}_2 + \mathsf{C}_2\right)\right).$$

We will show that allowing a quantum data structure for the outer walk, one can push the setup cost S_2 of the inner walk to the outer one. In order to accomplish this, we include in the data structure $\mathcal{D}(u)$, for each state u of the outer walk, the initial state for the inner quantum walk. It is then necessary that the initial state for the inner walk be properly preserved by the outer update, as well as the inner walk, so that we needn't perform the inner setup multiple times. In equations, this leads to an improved cost of order (Theorem 3.2):

$$\mathsf{S}_1 + \mathsf{S}_2 + \frac{1}{\sqrt{\varepsilon_1}}\left(\frac{1}{\sqrt{\delta_1}}\mathsf{U}'_1 + \frac{1}{\sqrt{\varepsilon_2}}\left(\frac{1}{\sqrt{\delta_2}}\mathsf{U}_2 + \mathsf{C}_2\right)\right),$$

where some logarithmic factors are now omitted. The new update cost U'_1 is the cost for updating the two data structures when one step of the outer walk is performed. In our application, U'_1 will be of the same order as U_1. Note that the update of the inner walk may be performed much more often than the update of the outer walk. Some of the power of our nested framework comes from

keeping expensive operations in the outer update, and pushing only cheaper operations into the inner update.

Our very simple idea seems for now as powerful as most learning graph based algorithms [Bel12b, BL11, Zhu11, LMS11, LMS13], except the recent one for k-element distinctness [Bel12a], which deviates significantly from the original learning graph framework. Nonetheless, we conjecture that one might use our framework, possibly with further extensions, to design a similar algorithm.

The idea is a posteriori very natural. Let us make an analogy with the adversary method. The method was first designed with positive weights by Ambainis [Amb00]. Only several years later, the possibility to put negative weight was proven to be not only possible but useful. Later, using the dual of this method, span programs were proven to achieve optimal quantum query complexity. In a more modest direction, we hope that our contribution will lead to a better understanding of the power of quantum walks and learning graphs.

While quantum walks are a powerful computational primitive, our main motivation for extending the quantum walk framework to include recent learning graph algorithms was not to demonstrate that fact. We believe that the framework will be useful for designing new quantum algorithms, since it allows the use of intuition gained from experience with quantum and random walks. Some ideas, like function composition and subroutines, are very natural in the quantum walk framework, but not as straightforward in the learning graph framework. We hope that the different perspectives provided by the quantum walk framework and the learning graph framework will lead to the discovery of new quantum algorithms.

2 Preliminaries

2.1 Model and notation

Our framework applies to other notions of complexity such as time and space. Nonetheless we will only illustrate it within the model of query complexity. In the model of quantum query complexity, the input string x is accessed by querying a black box to learn the bits of x. More concretely, the black box performs the following unitary operation: $\mathcal{O}_x : |j, b\rangle \mapsto |j, b \oplus x_j\rangle$. The quantum query complexity of a quantum algorithm is the number of times the algorithm uses the operator $\mathcal{O}_x$. In this paper, x will always refer to the input string. For a more thorough introduction to the quantum query model, see [BBC$^+$01].

For problems where the input is a graph, such as the triangle finding problem, x will be the adjacency matrix of an undirected graph. The triangle finding problem asks if the input graph contains a *triangle*, i.e., three pairwise adjacent vertices. A vertex is called a *triangle vertex* if it is part of a triangle. Similarly, an edge is called a *triangle edge* if it is part of a triangle. The best known triangle finding algorithms solve, as a subroutine, the graph collision problem. In the graph collision problem parametrized by a graph G on n vertices, the input is a binary string of length n that indicates whether a vertex in G is marked or not. The objective is to determine whether there exist two adjacent vertices in G that are both marked. Note that the graph G is part of the problem specification, and not part of the input.

We will use the $\tilde{O}$ notation to suppress logarithmic factors. More precisely, $f(n) = \tilde{O}(g(n))$ means $f(n) = O(g(n) \log^k n)$ for some integer k.

Given any algorithm computing a function $f(x)$ with probability at least $2/3$, we can repeat this algorithm $O\left(\log\left(\frac{1}{\epsilon}\right)\right)$ times and take the majority vote of outcomes to boost the algorithm's success probability to $1 - \epsilon$. Since all the subroutines in this paper are used at most $\text{poly}(n)$ times, if we boost a bounded error subroutine's success probability to $1 - 1/\text{poly}(n)$, then the final algorithm will still be correct with bounded error. This adds one log factor to the complexity of

4

the subroutine; this is the source of the log factors in this paper. We will also use this technique to boost the success probability of quantum algorithms that do not output a bit, but instead flip the phase when $f(x)$ is 1. By standard techniques, such an algorithm can be converted to one that computes $f(x)$ in a separate register. Now we can repeat this algorithm logarithmically many times, flip the phase conditioned on the majority vote of outcomes and uncompute the answers to obtain an algorithm whose success probability has been boosted to $1 - 1/\text{poly}(n)$.

The Johnson graph $J(n, r)$ is a graph on $\binom{n}{r}$ vertices, where each vertex is an r-subset of $\{1, 2, \ldots, n\}$, and two vertices are adjacent if their symmetric difference contains exactly two elements. The spectral gap of $J(n, r)$ is $\Omega(1/r)$.

Let $G = (V, E)$ be a graph, and let $R, S \subseteq V$ and F a set of edges on V, not necessarily contained in E. We denote by G_R the subgraph of G induced by R, and by $G_{R,S}$ the bipartite subgraph of G induced by classes R and S. That is, $G_{R,S}$ has vertex set $R \cup S$ and edges set $R \times S \cap E$. Denote by $G(F)$ the subgraph of G obtained by restricting to edges in F. That is, $G(F)$ has vertex set V and edges set $E \cap F$.

2.2 Review of the quantum walk framework

We now review the quantum walk search framework of Magniez, Nayak, Roland and Santha [MNRS11]. We present a slight modification of the framework, which is tailored to our application. We discuss the minor differences between our presentation and the original framework at the end of this section.

Let P be a reversible Markov chain on (finite) state space Ω, with stationary distribution π. Let $\mathcal{D}_x : \Omega \to \mathcal{H}_D$, for some Hilbert space $\mathcal{H}_D$, be a mapping that associates data with a state of the Markov chain for input x (i.e., for $u \in \Omega$, $|\mathcal{D}_x(u)\rangle$ is the data associated with state u for input x). We will refer to $\{|\mathcal{D}_x(u)\rangle\}_{u \in \Omega}$ as the data structure associated with the Markov chain.

In all previous applications of the quantum walk search framework, the data structure has always been classical, i.e., $|\mathcal{D}_x(u)\rangle$ is a classical basis state for all $u \in \Omega$, however, this needn't be the case in general. In our framework, we make use of quantum data structures.

Setup Define $\mathcal{H}_L \cong \mathcal{H}_R \cong \mathcal{H}_\Omega := \text{span}\{\{|0\rangle\} \cup \{|u\rangle : u \in \Omega\}\}$. Define the initial state $|\pi^x\rangle$ as $|\pi^x\rangle := \sum_{u \in \Omega} \sqrt{\pi_u} |u\rangle_L |0\rangle_R |\mathcal{D}_x(u)\rangle_D$ and let $\mathsf{S} = \mathsf{S}(P, \mathcal{D}_x)$ be the cost of constructing $|\pi^x\rangle$. We call S the *setup cost*.

Update Define two unitary operators $\mathcal{U}_P$ and $\mathcal{U}_D$ with the following action for all $u \in \Omega$:

$$\mathcal{U}_P : |u\rangle_L |0\rangle_R \mapsto |u\rangle_L \sum_{v \in \Omega} \sqrt{P_{uv}} |v\rangle_R, \quad \text{and} \quad \mathcal{U}_D : |u\rangle_L |v\rangle_R |\mathcal{D}_x(u)\rangle \mapsto |u\rangle_L |v\rangle_R |\mathcal{D}_x(v)\rangle.$$

In [MNRS11], these two operations are defined as one, however, we find it beneficial to consider them separately, since they play different roles: $\mathcal{U}_P$ updates the walk according to the Markov chain P, while $\mathcal{U}_D$ updates the data structure. Since the Markov chain is independent of the input in all of our applications, the query complexity of implementing $\mathcal{U}_P$ is 0, although its time complexity is non-zero.

Let $\mathsf{U} = \mathsf{U}(P, \mathcal{D}_x)$ denote the cost of implementing $\mathcal{U}_D \mathcal{U}_P$. We refer to U as the *update cost*.

Checking Fix some set $M_x \subseteq \Omega$ of marked elements. Let $\mathsf{C} = \mathsf{C}(M_x)$ be the cost of implementing the unitary that acts as

$$|u\rangle_\Omega |\mathcal{D}_x(u)\rangle_D \mapsto \begin{cases} -|u\rangle_\Omega |\mathcal{D}_x(u)\rangle_D & \text{if } u \in M_x \\ |u\rangle_\Omega |\mathcal{D}_x(u)\rangle_D & \text{else} \end{cases} .$$

We refer to C as the *checking cost*. With these definitions, we are ready to state the main theorem of [MNRS11].

Theorem 2.1. *Let $\delta > 0$ be the eigenvalue gap of P. Let ε be such that for any x, if M_x is non-empty, then $\pi(M_x) \geq \varepsilon$. Then there is a quantum algorithm with cost*

$$O\left(\frac{1}{\sqrt{\varepsilon}}\left(\frac{1}{\sqrt{\delta}}\mathsf{U} + \mathsf{C}\right)\right)$$

that, when given the initial state $|\pi^x\rangle$, determines with bounded error whether M_x is non-empty. More precisely, the algorithm implements (with bounded error) the map

$$|\pi^x\rangle \mapsto \begin{cases} -|\pi^x\rangle & \text{if } M_x \neq \emptyset \\ |\pi^x\rangle & \text{else} \end{cases} \quad .$$

We will use of the above theorem, which is proven, though not explicitly stated in these terms, in [MNRS11]. The explicit theorem of [MNRS11] follows directly from this theorem, and is stated in Corollary 2.2.

Corollary 2.2. *There is a quantum algorithm with cost*

$$O\left(\mathsf{S} + \frac{1}{\sqrt{\varepsilon}}\left(\frac{1}{\sqrt{\delta}}\mathsf{U} + \mathsf{C}\right)\right)$$

that determines with bounded error whether M_x is non-empty.

This presentation differs from the original presentation in some aspects. First, we allow the data structure to be quantum, while the original presentation implicitly assumed it to be classical. This, however, does not affect the proof. Second, the original presentation had a data structure associated with both registers of the walk, whereas we have only one data structure. As mentioned in [MNRS11], this is only for convenience. Third, we explicitly state the cost of the algorithm as a sum of two costs, the setup cost and the cost of running the walk. This is consistent with the original algorithm.

3 Two-level nested quantum walks

In this section we show how to augment the above framework with nested quantum walks using a quantum data structure. As motivation for this framework, it will be useful to compare the quantum walk algorithm of Magniez, Santha and Szegedy for triangle finding [MSS07], which has query complexity $\tilde{O}(n^{1.3})$, with the learning graph algorithm of Belovs [Bel12b], which has query complexity $O(n^{35/27}) = O(n^{1.297})$. In Section 3.2, we formally state the general construction for the case of two-level nested quantum walks. In Section 4.1, we explain how to generalize to k levels of nesting.

3.1 Warming up

Quantum walk algorithm for triangle finding In the quantum walk algorithm of [MSS07], a quantum walk is used to search for a pair of triangle vertices. The walk is on the Johnson graph of sets of r vertices, $J(n, r)$, which has spectral gap $\delta = \Omega\left(\frac{1}{r}\right)$. A state is marked if it contains two vertices of a triangle, so the proportion of marked states is lower bounded by $\varepsilon = \Omega\left(\frac{r^2}{n^2}\right)$.

If G denotes the input graph, the data associated with a state R of the Johnson graph is $|\mathcal{D}_G(R)\rangle = |G_R\rangle$, which encodes the subgraph of G induced by the vertices in R. The setup cost is $\mathsf{S} = O(r^2)$, since, for each state, we must initially query all $\binom{r}{2}$ possible edges in the subgraph induced by R. The update cost is $\mathsf{U} = O(r)$, since, at each step of the walk, we remove a vertex i and add a new vertex i' to the set of vertices, and must therefore test the adjacency of i' with all remaining vertices in the set.

Finally, to check whether we have two triangle vertices in a state R, we search for a vertex k that forms a triangle with two of the vertices in R. To check if a fixed vertex k forms a triangle with any two of the vertices in R, we solve the graph collision problem on G_R, which is encoded in $\mathcal{D}_G(R)$, with vertices in R defined as being marked if they are adjacent to k. Thus, we can check if a vertex k forms a triangle with two vertices in R using $O(r^{2/3})$ queries, and so we can check if any vertex forms a triangle with two vertices in R using $\mathsf{C} = O(\sqrt{n}r^{2/3})$ queries.

Plugging all of these values into the formula from Corollary 2.2 gives (neglecting logarithmic factors)

$$
\begin{aligned}
\mathsf{S} + \tfrac{1}{\sqrt{\varepsilon}}\left(\tfrac{1}{\sqrt{\delta}}\mathsf{U} + \mathsf{C}\right) &= r^2 + \tfrac{n}{r}\left(\sqrt{r}r + \sqrt{n}r^{2/3}\right) \\
&= r^2 + n\sqrt{r} + \tfrac{n^{3/2}}{r^{1/3}},
\end{aligned}
$$

which is optimized by setting $r = n^{3/5}$, yielding query complexity $\tilde{O}(n^{1.3})$ for triangle finding.

Learning graph algorithm for triangle finding In contrast, the recent learning graph algorithm of Belovs [Bel12b] for triangle finding considers sets of r vertices, but only queries a sparsification of the possible edges between them. If the sparsification parameter is $s \in (0, 1]$, then each potential edge is queried with probability s, giving an expected $s\binom{r}{2}$ queries for each state.

We can try to apply this edge sparsification idea to the above quantum walk. We will redefine the data structure to be a random sparsification of the potential edges in G_R, so that the setup cost decreases to $\mathsf{S} = O(sr^2)$ and similarly the update cost decreases to $\mathsf{U} = O(sr)$ (let us assume that the setup and update operations can be performed with their average cost, instead of their worst-case cost). However, now some states that contain two triangle vertices will not contain the edge between them, and will therefore not be detected by the checking procedure. Thus, the probability that a state is marked has also decreased to $\varepsilon = \Omega(s\tfrac{r^2}{n^2})$. If we plug in these new values, we get query complexity

$$
\begin{aligned}
\mathsf{S} + \tfrac{1}{\sqrt{\varepsilon}}\left(\tfrac{1}{\sqrt{\delta}}\mathsf{U} + \mathsf{C}\right) &= sr^2 + \tfrac{n}{r\sqrt{s}}\left(\sqrt{r}sr + \sqrt{n}r^{2/3}\right) \\
&= sr^2 + n\sqrt{rs} + \tfrac{n^{3/2}}{r^{1/3}\sqrt{s}},
\end{aligned}
$$

which is optimized by $s = 1$ and $r = n^{3/5}$, yielding no improvement over the standard quantum walk algorithm.

One key idea on which our framework is based, is to observe that a state R that contains two triangle vertices should be considered marked, even if its data structure is missing the edge between them. Of course, we need that edge to know that the state is marked, but we can check this by searching for a *good* data structure, i.e., one containing the triangle edge. To this end, we allow the data structure of a state R to be a quantum superposition over a number of classical data structures, some of which may contain some kind of certificate for R, if R is in fact marked. We walk on this data structure as a checking subroutine. Importantly, the setup of this inner walk, that is, the creation of an appropriate superposition of data structures, has already been taken care of as part of the setup of the outer walk. It is very important that we only perform this setup once, as opposed to at every checking step.

In , we use our framework to effectively apply the sparsification idea to triangle finding in a nested quantum walk. We first present the general framework for nesting quantum walks with a quantum data structure. We start with nesting one walk within another.

3.2 Framework for nested quantum walks

Let P be a reversible Markov chain on (finite) state space Ω, with stationary distribution π. For each input x, fix a marked set $M_x \subseteq \Omega$. Let $\mathcal{D}_x : \Omega \to \mathcal{H}_D$ be some associated data mapping. The Markov chain P defines what we refer to as the *outer walk*.

Fix $u \in \Omega$. We will define what we refer to as the *inner walk*, whose purpose is to check whether u is in M_x. Let P' be a reversible Markov chain on state space Ω' with stationary distribution π'. Let $M_x^u \subseteq \Omega'$ be the marked set for the inner walk, with respect to u. Let $\mathcal{D}_x^u : \Omega' \to \mathcal{H}_{D'}$ be the inner data mapping with respect to u. Note that, although we have the walk itself, defined by P', Ω' and π', independent of u, the marked set and data mapping are dependent on u.

We suppose the following relationship between the inner and outer marked sets:

$$M_x = \{u \in \Omega : M_x^u \neq \emptyset\}.$$

We can then view the set M_x^u as a set of witnesses for $u \in M_x$.

In order to make use of the quantum data structure, we suppose the following relationship holds for the inner and outer data structures. First, we require $\mathcal{H}_D = \mathcal{H}_{\Omega'} \otimes \mathcal{H}_{D'}$, and in particular, we need

$$|\mathcal{D}_x(u)\rangle = \sum_{s \in \Omega'} \sqrt{\pi_s'} |s\rangle |\mathcal{D}_x^u(s)\rangle.$$

That is, $|\mathcal{D}_x(u)\rangle$ is the initial state of the inner walk with respect to u. In general, the data structure for the outer walk could store more information than just the initial state of the inner walk, but this is all we need for our applications.

Setup Define spaces $\mathcal{H}_L \cong \mathcal{H}_R \cong \mathcal{H}_\Omega$ and $\mathcal{H}_{L'} \cong \mathcal{H}_{R'} \cong \mathcal{H}_{\Omega'}$. The setup cost, $\mathsf{S} = \mathsf{S}(P, \mathcal{D}_x)$, is the cost of constructing the initial state:

$$
\begin{aligned}
|\pi^x\rangle &= \sum_{u \in \Omega} \sqrt{\pi_u} |u\rangle_L |0\rangle_R |\mathcal{D}_x(u)\rangle_D |0\rangle_{R'} \\
&= \sum_{u \in \Omega} \sqrt{\pi_u} |u\rangle_L |0\rangle_R \sum_{s \in \Omega'} \sqrt{\pi_s'} |s\rangle_{L'} |\mathcal{D}_x^u(s)\rangle_{D'} |0\rangle_{R'}.
\end{aligned}
$$

We emphasize that the initial state for the outer walk contains, in superposition, initial states for the inner walks with respect to each $u \in \Omega$.

Update The update cost, $\mathsf{U} = \mathsf{U}(P, \mathcal{D}_x)$, is the cost of implementing $\mathcal{U}_D \mathcal{U}_P$. Recall that $\mathcal{U}_P$ is defined by the action $\mathcal{U}_P : |u\rangle_L |0\rangle_R \mapsto |u\rangle_L \sum_{v \in \Omega} \sqrt{P_{u,v}} |v\rangle_R$. Similarly, $\mathcal{U}_D$ is defined by the action

$$\mathcal{U}_D : |u\rangle_L |v\rangle_R |\mathcal{D}_x(u)\rangle_D \mapsto |u\rangle_L |v\rangle_R |\mathcal{D}_x(v)\rangle_D.$$

This mapping is equivalently stated as:

$$\mathcal{U}_D : |u\rangle_L |v\rangle_R \sum_{s \in \Omega'} \sqrt{\pi_s'} |s\rangle_{L'} |\mathcal{D}_x^u(s)\rangle_{D'} \quad \mapsto \quad |u\rangle_L |v\rangle_R \sum_{s \in \Omega'} \sqrt{\pi_s'} |s\rangle_{L'} |\mathcal{D}_x^v(s)\rangle_{D'},$$

which can be implemented by an operation with the action:

$$|u\rangle_L|v\rangle_R|\mathcal{D}_x^u(s)\rangle_{D'} \mapsto |u\rangle_L|v\rangle_R|\mathcal{D}_x^v(s)\rangle_{D'}.$$

The most important point to emphasize about this outer update is that we must update from the inner initial state with respect to some outer state u (contained in $|\mathcal{D}_x(u)\rangle$), to the inner initial state with respect to some other outer state v (contained in $|\mathcal{D}_x(v)\rangle$).

Checking To check whether a state u is in M_x, we want to implement the reflection

$$\mathcal{C} : |u\rangle|\mathcal{D}_x(u)\rangle \mapsto \begin{cases} -|u\rangle|\mathcal{D}_x(u)\rangle & \text{if } u \in M_x \\ |u\rangle|\mathcal{D}_x(u)\rangle & \text{else} \end{cases},$$

which, in this case, is:

$$\mathcal{C} : |u\rangle \sum_{s\in\Omega'} \sqrt{\pi_s'}|s\rangle|\mathcal{D}_x^u(s)\rangle \;\mapsto\; \begin{cases} |u\rangle\left(-\sum_{s\in\Omega'} \sqrt{\pi_s'}|s\rangle|\mathcal{D}_x^u(s)\rangle\right) & \text{if } M_x^u \neq \emptyset \\ |u\rangle\left(\sum_{s\in\Omega'} \sqrt{\pi_s'}|s\rangle|\mathcal{D}_x^u(s)\rangle\right) & \text{else} \end{cases}.$$

This is simply a reflection about the initial state of the inner walk with respect to u. Importantly, this preserves the inner initial state for future use, so we needn't set it up again. We accomplish this reflection by way of Theorem 2.1 applied to the inner walk. Let $\mathsf{U}' = \mathsf{U}'(P', \mathcal{D}_x^u)$ be an upper bound (over all inputs x and states u) on the update cost of the inner walk. That is, U' is the cost of implementing

$$\mathcal{U}_P' : |s\rangle_{L'}|0\rangle_{R'} \mapsto |s\rangle_{L'} \sum_{t\in\Omega'} \sqrt{P_{st}'}|t\rangle_{R'} \quad \text{and} \quad \mathcal{U}_D' : |s\rangle_{L'}|t\rangle_{R'}|\mathcal{D}_x^u(s)\rangle_{D'} \mapsto |s\rangle_{L'}|t\rangle_{R'}|\mathcal{D}_x^u(t)\rangle_{D'}.$$

Let $\mathsf{C}' = \mathsf{C}'(M_x^u)$ be the checking cost of the inner walk, that is, the cost of implementing the reflection

$$|s\rangle|\mathcal{D}_x^u(s)\rangle \mapsto \begin{cases} -|s\rangle|\mathcal{D}_x^u(s)\rangle & \text{if } s \in M_x^u \\ |s\rangle|\mathcal{D}_x^u(s)\rangle & \text{else} \end{cases}.$$

Let δ' denote the spectral gap of P' and ε' denote a lower bound on $\pi'(M_x^u)$ for non-empty $\widehat{M}_x^u$.

Lemma 3.1. *Let $\widehat{\mathsf{C}}$ be the cost of implementing $\mathcal{C}$ with bounded error. Then $\widehat{\mathsf{C}} = O\left(\frac{1}{\sqrt{\varepsilon'}}\left(\frac{1}{\sqrt{\delta'}}\mathsf{U}' + \mathsf{C}'\right)\right)$.*

Proof. This follows directly from Theorem 2.1, since $|\mathcal{D}_x(u)\rangle$ is the initial state of the inner walk (with respect to u). $\qquad\square$

Theorem 3.2. *We can decide whether M_x is non-empty, with bounded error, in quantum query complexity*

$$\tilde{O}\left(\mathsf{S} + \frac{1}{\sqrt{\varepsilon}}\left(\frac{1}{\sqrt{\delta}}\mathsf{U} + \frac{1}{\sqrt{\varepsilon'}}\left(\frac{1}{\sqrt{\delta'}}\mathsf{U}' + \mathsf{C}'\right)\right)\right).$$

Proof. We can reduce the error in $\mathcal{C}$ from bounded error to error that is inverse polynomial at the cost of an additional log factor, as mentioned in Section 2. Since we only run for a polynomial number of steps, the result follows. $\qquad\square$

This construction can be applied inductively to achieve nesting to any depth k, though the number of terms in the complexity will be proportional to k. Though the generalization to k levels is fairly straightforward, we give the explicit construction in Section 4.1.

3.3 Averaging costs

In this section we prove a lemma about averaging costs of operations that will be useful in some of our nested walk applications. In particular, we sometimes have update costs which are much lower on average than in the worst case, and we would like to incur the average, rather than worst case update cost.

Let $\{U_i\}$ be a set of unitaries acting on the same space. If we know how to exactly implement each U_i with c_i queries, then we can exactly implement the controlled unitary $U = \sum_i |i\rangle\langle i| \otimes U_i$ with $\max_i c_i$ queries. However, if the state we are implementing this unitary on has a nearly uniform distribution over the control register, then intuitively it seems like we should be able to approximately implement the unitary with a cost that goes like the average instead of the maximum. The following lemma makes this precise.

Lemma 3.3. *Let q_i be the query complexity of exactly implementing unitary U_i. Let U be the controlled unitary $U = \sum_i |i\rangle\langle i| \otimes U_i$. For any prescribed upper bound q, we can convert the state $|s\rangle = \sum_i \alpha_i |i\rangle |s_i\rangle$ to a state $|t\rangle$ using at most q queries, such that $|t\rangle$ has inner product at least $1 - \epsilon_q$ with $U|s\rangle = \sum_i \alpha_i |i\rangle U_i |s_i\rangle$, where $\epsilon_q \leq \sum_{i:q_i > q} |\alpha_i|^2$. In particular, with $O(\sum_i |\alpha_i|^2 q_i)$ queries, we can achieve any constant ϵ.*

Proof. The algorithm for implementing this conversion is simple: We only implement those matrices U_i whose cost is less than the prescribed upper bound. Otherwise we implement, for example, the identity matrix. More precisely, we implement the unitary $\tilde{U} = \sum_{i:q_i \leq q} |i\rangle\langle i| \otimes U_i + \sum_{i:q_i > q} |i\rangle\langle i| \otimes I$. Clearly this unitary costs at most q queries to implement.

The inner product between $U|s\rangle$ and $\tilde{U}|s\rangle$ is $\sum_{i:q_i \leq q} |\alpha_i|^2 \langle s_i|s_i\rangle + \sum_{i:q_i > q} |\alpha_i|^2 \langle s_i|U|s_i\rangle$, which is at least $\sum_{i:q_i \leq q} |\alpha_i|^2$. Thus $\epsilon_q \leq \sum_{i:q_i > q} |\alpha_i|^2$.

To show the final portion of the theorem, consider the random variable that takes value q_i with probability $|\alpha_i|^2$. By Markov's inequality, the probability that this random variable takes a value greater than k times the mean is less than $1/k$. So, for any $k \geq 1$, if we set $q = k(\sum_i |\alpha_i|^2 q_i)$, we can achieve $\epsilon \leq 1/k$. $\qquad\square$

Note that Markov's inequality uses very minimal information about the structure of the random variable. If the random variable has more structure, such as being tightly concentrated around the mean, then we can get much better estimates. The lemma is simple and constructive, and thus it can also be made efficient in terms of time and space if the approximate unitary $\tilde{U}$ can be implemented efficiently.

3.4 Two applications to triangle finding

We can now present our first two nested quantum walk algorithms, both for the triangle finding problem. We first focus on the the learning graph of Belovs [Bel12b] since it is more challenging to implement due to the sparsification idea. Later we will sketch the quantum walk algorithm corresponding to the more recent and more efficient learning graph in [LMS13].

Theorem 3.4. *There is a nested quantum walk algorithm for triangle finding with quantum query complexity $\tilde{O}(n^{\frac{35}{27}})$.*

Proof. Let the input graph be called G. The algorithm consists of two quantum walks, which we will call the inner and outer walk.

Outer walk   The outer walk is a quantum walk on $J(n, r)$, just as in the previous triangle finding algorithm, where $1 \leq r \leq n$ is a parameter to be chosen later. Each state of the Johnson graph corresponds to an r-subset of the vertex set of G. A state of the Johnson graph, R, is said to be marked if it contains two triangle vertices. This fully describes the Markov chain of the outer walk; we now have to describe the data structure associated with each state and the checking subroutine, which is the inner walk. As described in Section 3.2, the data structure associated with each state of the outer walk will be the initial state of the inner walk.

Inner walk   The inner walk is on $J\left(\binom{r}{2}, s\binom{r}{2}\right)$, where $1/r < s \leq 1$ is a parameter to be chosen later. A state, T, of the inner walk is an $s\binom{r}{2}$-sized subset of the potential edges of an r-vertex graph. The data structure associated with a state T with outer state R, $\mathcal{D}^R(T)$, is the subgraph $G_R(T)$. A state T is said to be marked if $G_R(T)$ contains a triangle edge. The checking subroutine for this walk is the same as the checking step in the quantum walk described in Section 3: we search for a vertex outside of R that forms a triangle with one of the edges in $G_R(T)$.

Outer walk data structure   We now return to describing the data structure of the outer walk. The quantum data structure corresponding to any state R of the outer walk will be the initial state of the inner quantum walk, i.e., the uniform superposition over all states of the inner walk. For the inner walk, the initial state is $|\pi_2^R\rangle = \sum_{T \in \Omega_2} \frac{1}{\sqrt{|\Omega_2|}} |T\rangle |G_R(T)\rangle$, where Ω_2 is the state space of the inner walk, and $|\Omega_2| = \binom{\binom{r}{2}}{s\binom{r}{2}}$.

Inner walk cost analysis   We can now compute the costs of the various operations in the inner walk. In the inner walk, the checking step is the same as the one described in Section 3.1 and thus costs $\mathsf{C}_2 = O(\sqrt{n}r^{2/3})$. The update operation just deletes and adds a new edge, giving $\mathsf{U}_2 = O(1)$. The spectral gap of the Johnson walk is $\delta_2 = \Omega(1/sr^2)$ and the fraction of marked states, assuming the outer state is marked, is $\varepsilon_2 \geq s$. Thus the cost of performing the inner walk, given the inital state for the walk, which is the checking cost for the outer walk, is

$$
\begin{aligned}
\mathsf{C}_1 &= \tilde{O}\left(\frac{1}{\sqrt{\varepsilon_2}}\left(\frac{\mathsf{U}_2}{\sqrt{\delta_2}} + \mathsf{C}_2\right)\right) \\
&= \tilde{O}\left(\frac{1}{\sqrt{s}}\left(\sqrt{sr^2} + \sqrt{n}r^{2/3}\right)\right) \\
&= \tilde{O}\left(r + \sqrt{\frac{n}{s}}r^{2/3}\right).
\end{aligned}
$$

Since the checking step with cost C_2 is only bounded error, we need to amplify the success probability, which leads to the log factor subsumed in the $\tilde{O}$ notation.

Outer walk cost analysis   Now we can move to the outer walk. The superposition corresponding to a state in the outer walk is

$$
|R\rangle |\pi_2^R\rangle = |R\rangle \sum_{T \in \Omega_2} \frac{1}{\sqrt{|\Omega_2|}} |T\rangle |G_R(T)\rangle.
$$

Setting up this superposition clearly costs $\mathsf{S}_1 = sr^2$ queries. A state is considered marked if it contains both vertices of a triangle edge. Thus $\varepsilon_1 = \Omega\left(\left(\frac{r}{n}\right)^2\right)$. The spectral gap of this walk is $\delta_1 = \Omega(1/r)$.

Finally, we have to compute the update cost. The update cost is the cost of performing the map

$$|R\rangle \otimes |R'\rangle \otimes \sum_{T\in\Omega_2} \frac{1}{\sqrt{|\Omega_2|}}|T\rangle|G_R(T)\rangle$$

to

$$|R\rangle \otimes |R'\rangle \otimes \sum_{T\in\Omega_2} \frac{1}{\sqrt{|\Omega_2|}}|T\rangle|G_{R'}(T)\rangle.$$

Since R and R' are adjacent states in a Johnson graph, R has one vertex not in R', and R' has one vertex not in R. Let us call these vertices a and a' respectively. The cost of updating $|G_R(T)\rangle$ to $|G_{R'}(T)\rangle$ is therefore $\deg(a) + \deg(a')$. In the worst case the degree of a vertex is r. However, on average any vertex has degree sr. Now we can invoke Lemma 3.3. For a given query upper bound q, ϵ_q is the probability that a randomly selected vertex will have degree greater than q. If we set $q = 7(sr) + 100\log n$, then we need to compute the probability that the degree of a vertex is more than 7 times its mean (the extra term is to deal with the case when sr is very small). The degree of a vertex is a random variable distributed according to the hypergeometric distribution, for which strong tail bounds are known. By Theorem 2.10 and eq. (2.11) of [JLR11], $\epsilon_q \leq \exp(-q)$ for any q at least 7 times the mean. In our case, this gives $\epsilon_q \leq \exp(-7sr)\frac{1}{n^{100}}$, which is sufficient for our purposes.

Thus the final query complexity is

$$\tilde{O}\left(\mathsf{S}_1 + \frac{1}{\sqrt{\varepsilon_1}}\left(\frac{\mathsf{U}_1}{\sqrt{\delta_1}} + \mathsf{C}_1\right)\right)$$
$$= \tilde{O}\left(sr^2 + \frac{n}{r}\left(sr\sqrt{r} + r + \sqrt{\frac{n}{s}}r^{2/3}\right)\right)$$
$$= \tilde{O}\left(sr^2 + ns\sqrt{r} + n + \frac{n^{3/2}}{\sqrt{s}r^{1/3}}\right),$$

which is optimized by choosing $r = n^{2/3}$ and $s = n^{-1/27}$, giving a final query complexity of $\tilde{O}(n^{35/27})$. □

We note that the optimal values of r and s are the same as those obtained in the learning graph upper bound. In fact, if we add some unnecessary layers of nesting by breaking up the outer walk into several nested walks, we can obtained the learning graph complexity term for term. This suggests a natural correspondence between these two frameworks.

To emphasize a key ingredient in our framework, we note that the inner update is much cheaper than the outer update, since the inner update involves changing a single edge, whereas the outer update involves changing all edges adjacent to a single vertex. This is ideal, since the inner update operation is performed much more frequently than the outer update.

A very recent unpublished result of Lee, Magniez, and Santha [LMS13] constructs a learning graph for triangle finding with query complexity $O(n^{9/7}) = O(n^{1.286})$. We can easily convert their learning graph to a very simple nested walk. Due to its simplicity, it is perhaps less illustrative of the power of the framework than the original learning graph for triangle finding, however, it illustrates the cost savings that can be obtained by performing expensive updates less frequently (by keeping them in the outer walk).

Theorem 3.5. *There is a nested quantum walk algorithm for triangle finding with query complexity* $\tilde{O}(n^{9/7})$.

Proof. We will make use of the fact that collision finding on an unbalanced bipartite graph on classes of size r_1 and r_2 with $r_1 \leq r_2 \leq r_1^2$ has quantum query complexity $O((r_1 r_2)^{1/3})$ (see Appendix A).

Outer walk The outer walk is on $J(n, r_1)$, for $1 \leq r_1 \leq n$. A state R_1 is marked if it contains a triangle vertex. Thus, we have $\delta_1 = \Omega(\frac{1}{r_1})$ and $\varepsilon_1 = \Omega(\frac{r_1}{n})$.

Inner walk The inner walk is on $J(n, r_2)$, for $1 \leq r_2 \leq n$. Suppose the outer state, R_1, is marked, so it contains the vertex from some triangle. A state R_2 is marked if it contains a vertex from the same triangle.

The data structure associated with a state R_2 with outer state R_1, $\mathcal{D}^{R_1}(R_2)$, is the subgraph of G containing exactly those edges with one endpoint in R_1, and one endpoint in R_2, G_{R_1, R_2}. Thus, $\delta_1 = \Omega(\frac{1}{r_2})$ and $\varepsilon_2 = \Omega(\frac{r_2}{n})$. To update, we must query r_1 potential edges from a new vertex to R_1, so $\mathsf{U}_2 = r_1$. To check if a state is marked, we search for a third triangle vertex, using a graph collision subroutine, with total cost $\mathsf{C}_2 = \sqrt{n}(r_1 r_2)^{1/3}$.

Outer walk data structure The quantum data structure corresponding to a state R_1 of the outer walk is $\sum_{R_2} \frac{1}{\sqrt{|\Omega_2|}} |R_2\rangle |G_{R_1, R_2}\rangle$. The cost of learning G_{R_1, R_2} is $\mathsf{S}_1 = r_1 r_2$, and the cost of updating G_{R_1, R_2} to G_{R_1', R_2} is $\mathsf{U}_1 = r_2$, since we must query r_2 potential edges between the new vertex and all of R_2.

The final query complexity is (neglecting log factors)

$$
\begin{aligned}
&\mathsf{S}_1 + \frac{1}{\sqrt{\varepsilon_1}} \left(\frac{1}{\sqrt{\delta_1}} \mathsf{U}_1 + \frac{1}{\sqrt{\varepsilon_2}} \left(\frac{1}{\sqrt{\delta_2}} \mathsf{U}_2 + \mathsf{C}_2 \right) \right) \\
=\ & r_1 r_2 + \sqrt{\frac{n}{r_1}} \left(\sqrt{r_1} r_2 + \sqrt{\frac{n}{r_2}} \left(\sqrt{r_2} r_1 + \sqrt{n}(r_1 r_2)^{1/3} \right) \right) \\
=\ & r_1 r_2 + \sqrt{n} r_2 + n\sqrt{r_1} + \frac{n^{3/2}}{r_1^{1/6} r_2^{1/6}}.
\end{aligned}
$$

Setting $r_1 = n^{4/7}$ and $r_2 = n^{5/7}$ gives the claimed upper bound. Note that the optimal values have r_2 larger than r_1, making the inner update cheaper than the outer update. The inner update is run more often, not only because $\frac{1}{\sqrt{\delta_2}} > \frac{1}{\sqrt{\delta_1}}$, but more fundamentally, because the term $\frac{1}{\sqrt{\delta_2}} \mathsf{U}_2$ has a factor of $\frac{1}{\sqrt{\varepsilon_1 \varepsilon_2}}$, whereas the term $\frac{1}{\sqrt{\delta_1}} \mathsf{U}_1$ only has a factor of $\frac{1}{\sqrt{\varepsilon_1}}$. $\qquad\square$

For this walk, in some sense, the inner marked set is independent of the outer state. To see this, consider the case where there is a single triangle, and note that this is the edge case from which we calculate a lower bound on the probability that a state is marked. In the case of a single triangle, we can imagine the inner and outer walks being on independent sets of vertices, R_1 and R_2, each of which is marked if it contains a triangle vertex. (Of course, they should not contain the same triangle vertex.) In some sense then, this algorithm does not use the full power of the model.

4 Multi-level nested quantum walks

4.1 Generalization of the framework

The simple two-level nesting from Section 3 can be generalized in a straightforward way to k levels of nesting.

Consider k walks $\{(P^{(i)}, \Omega^{(i)}, \pi^{(i)})\}_{i=1}^{k}$. Suppose the ith walk has marked sets

$$\{M_x^u\}_{u \in \Omega^{(1)} \times \cdots \times \Omega^{(i-1)}}$$

(with M_x implicitly being the unique marked set for the outermost walk) and data mappings

$$\{\mathcal{D}_x^u : \Omega^{(i)} \to \mathcal{H}_{D^{(i)}}\}_{u \in \Omega^{(1)} \times \cdots \times \Omega^{(i-1)}}.$$

Suppose that for all $i = 1, \ldots, k$ and all $u \in \Omega^{(1)} \times \cdots \times \Omega^{(i-1)}$, we have

$$M_x^u = \{v \in \Omega^{(i)} : M_x^{(u,v)} \neq \emptyset\},$$

and for $i = 1, \ldots, k-1$, $u \in \Omega^{(1)} \times \cdots \times \Omega^{(i-1)}$ and $v \in \Omega^{(i)}$, we have $\mathcal{H}_{D^{(i)}} = \mathcal{H}_{\Omega^{(i+1)}} \otimes \mathcal{H}_{D^{(i+1)}}$, and

$$|\mathcal{D}_x^u(v)\rangle = \sum_{w \in \Omega^{(i+1)}} \sqrt{\pi_w^{(i+1)}} |w\rangle |\mathcal{D}_x^{(u,v)}(w)\rangle.$$

Let δ_i denote the spectral gap of $P^{(i)}$, and ε_i be a lower bound for non-empty $\pi^{(i)}(M_x^u)$ for all $u \in \Omega^{(1)} \times \cdots \times \Omega^{(i-1)}$. Let S denote the cost of constructing

$$\sum_{u \in \Omega^{(1)}} \sqrt{\pi_u^{(1)}} |u\rangle |\mathcal{D}_x(u)\rangle.$$

Let U_i be the update cost of the ith walk. Let C be the cost of implementing, for $w \in \Omega^{(k)}$, and $u \in \Omega^{(1)} \times \cdots \times \Omega^{(k-1)}$,

$$|w\rangle |\mathcal{D}_x^u(w)\rangle \mapsto \begin{cases} -|w\rangle |\mathcal{D}_x^u(w)\rangle & \text{if } w \in M_x^u \\ |w\rangle |\mathcal{D}_x^u(w)\rangle & \text{else} \end{cases}.$$

Corollary 4.1. *We can decide whether M_x is non-empty with bounded error in*

$$\tilde{O}\left(\mathsf{S} + \sum_{i=1}^{k} \left(\prod_{j=1}^{i} \frac{1}{\sqrt{\varepsilon_j}} \right) \frac{1}{\sqrt{\delta_i}} \mathsf{U}_i + \left(\prod_{j=1}^{k} \frac{1}{\sqrt{\varepsilon_j}} \right) \mathsf{C} \right) \tag{1}$$

queries.

This is not the most general nested walk. Rather than have a single i^{th} walk $(P^{(i)}, \Omega^{(i)}, \pi^{(i)})$, we could have a family of ith walks, $\{(P^u, \Omega^u, \pi^u)\}_{u \in \Omega^{(1)} \times \cdots \times \Omega^{(i-1)}}$, where $\Omega^{(i)} = \bigcup_{u \in \Omega^{(1)} \times \cdots \times \Omega^{(i-1)}} \Omega^u$. However, such a general framework sacrifices the simplicity that is a feature of nested quantum walks.

4.2 Application to subgraph containment

It is simple to extend our quantum walk algorithms for triangle finding to obtain the more general learning graph upper bounds for subgraph containment [LMS11, Zhu11, LMS13]. We now give an explicit construction of nested quantum walks corresponding to the learning graphs of [LMS13], matching their complexity up to polylogarithmic factors, and even potentially improving their result for some cases. For simplicity consider the case of undirected graphs with no self-loops. Let H be a graph on vertices $\{1, 2, \ldots, k\}$. Then a graph G contains a copy of H if G induces a graph that is isomorphic to H on some subset of k vertices.

The learning graph construction In [LMS13], a meta-language is introduced for designing a learning graph that detects a copy of H in G. Instructions are `Setup`, `LoadVertex(i)` for $i \in [k]$ and `LoadEdge(i, j)` for $\{i, j\} \in E(H)$. A 'program' is an ordering of $\{\texttt{Setup}\} \cup \{\texttt{LoadVertex}(i)\}_{i \in [k]} \cup \{\texttt{LoadEdge}(i, j)\}_{\{i,j\} \in E(H)}$ such that `Setup` appears first and for each edge $\{i, j\}$ of H, instruction `LoadEdge(i, j)` occurs in the program after instructions `LoadVertex(i)` and `LoadVertex(j)`.

Parameters are vertex-set sizes $r_i \geq 1$ for each vertex i of H, and average degrees $d_{ij} \geq 1$ for each edge $\{i, j\}$ of H. These parameters are not related to H but parameterize the resulting learning graph. Assuming some natural conditions on these parameters, one can derive an algorithm whose complexity only depends on them. Optimizing them can be done via a linear program (see [LMS13] for a link to the program). We assume that $r_i \leq n$, $d_{ij} \leq \max(r_i, r_j)$, and that:

$$\text{For all } i \text{ there exists } j \text{ such that } \{i, j\} \in E(H) \text{ and } d_{ij}(2r_j + 1)/(2r_i + 1) \geq 1. \tag{2}$$

Each instruction has a cost that propagates to later instructions, and its total effective cost. The first one is called *global cost* and the second one *local cost*. In [LMS13], a table is given, nearly identical to Table 1, providing global and local costs for each of the three types of instructions. Given a program of $\tau = k + |E(H)| + 1$ instructions, if l_t denotes the local cost of instruction t, and g_t the global cost, then the resulting upper bound for deciding if H is in G is

$$\sum_{t=1}^{\tau} \left(\prod_{t' < t} g_{t'} \right) l_t. \tag{3}$$

The corresponding nested walk Given a program, we can define a sequence of nested walks as follows. The `Setup` instruction corresponds to our quantum walk setup. Each non-setup instruction corresponds to a quantum walk, with the walk corresponding to the tth instruction being nested at depth t; that is, the tth walk is the checking procedure for the $(t-1)^{\text{th}}$ walk. Each non-setup instruction corresponds to an edge or vertex in H, and so each of our walks will as well.

All walks will be on Johnson graphs. The state of the walk corresponding to a vertex i is described by a subset of vertices R_i of size r_i, and the state of the walk corresponding to an edge $\{i, j\}$ is described by a subset $T_{i,j}$ of $d_{ij} \min(r_i, r_j)$ potential edges between R_i and R_j. For each edge $\{i, j\}$ of H, our database maintains the edges $G(T_{ij})$ of G in $T_{i,j}$.

A state R_i is marked if there exists some copy of H in G such that: for each R_j in a walk at depth *less than* the depth of R_i, R_j contains the jth vertex in the subgraph, and for each T_{uv} at depth less than the depth of R_i, T_{uv} contains edge $\{u, v\}$ in the subgraph, and finally, R_i contains the ith vertex in the subgraph. A state T_{ij} has a similar condition for being marked, except T_{ij} must contain edge $\{i, j\}$ in the subgraph.

Setup To setup, we must create the uniform superposition $|\pi\rangle$ of subgraphs $G(T_{ij})$, one for each edge $\{i, j\} \in E(H)$. The total cost is

$$\mathsf{S} = \sum_{\{i,j\} \in H} \min(r_i, r_j) d_{ij}.$$

Vertex walks To update a walk on vertex set R_i we remove a vertex and add a new one. To update the data, we must update the graphs $G(T_{ij})$, for each j adjacent to i in H. By Lemma 3.3, this complexity is related to the average degree in R_i, which we can calculate as $d_{ij} \min(1, r_j/r_i)$. The total cost for this type of update is then

$$\mathsf{U} = \sum_{j: \{i,j\} \in H} d_{ij} \min(1, r_j/r_i).$$

Instruction	Global Cost	Local Cost
Setup	1	$\mathsf{S} = \sum\limits_{\{i,j\}\in H} \min\{r_i, r_j\}d_{ij}$
LoadVertex(i)	$\frac{1}{\sqrt{\varepsilon}} = \sqrt{n/r_i}$	$\frac{1}{\sqrt{\varepsilon\delta}}\mathsf{U} = \sqrt{n} \times \sum\limits_{j:\{i,j\}\in H} d_{ij}\min(1, r_j/r_i)$
LoadEdge(i,j)	$\frac{1}{\sqrt{\varepsilon}} = \sqrt{\max\{r_i, r_j\}/d_{ij}}$	$\frac{1}{\sqrt{\varepsilon\delta}}\mathsf{U} = \sqrt{r_i r_j}$

Table 1: Global and local costs for quantum walks corresponding to each type of instruction.

Observe that this quantity is always $\Omega(1)$, by the imposed condition in (2).

We can easily calculate that the proportion of marked states in this type of walk is $\varepsilon = r_i/n$, and the spectral gap is $\delta = 1/r_i$.

Edge walks Lastly, to update a walk on edge set T_{ij} we simply replace one edge in T_{ij} and update $G(T_{ij})$. Thus the cost of this type of update is simply $\mathsf{U} = 1$. For this type of walk we have $\varepsilon = d_{ij}/\max(r_i, r_j)$ (since ε is the probability that a particular edge of $R_i \times R_j$ is in T_{ij}) and $\delta = 1/\min(r_i, r_j)$.

Checking A state of the deepest walk is marked if the data encodes an entire copy of H in G. Since we can detect this from the data with no further queries to G, the final checking cost is $\mathsf{C} = 0$.

Global and local costs We can rephrase our costs for each walk into global and local costs. The global cost is simply the part of the cost that propagates forward to deeper walk costs. For the setup (which we can consider as a walk on a complete graph in which every state is marked), this is simply 1, whereas for other walks, the global cost is $\frac{1}{\sqrt{\varepsilon}}$ (see (1)). Similarly, we can consider local costs. For the setup this is defined as S, whereas for each other walk, it is defined as $\frac{1}{\sqrt{\varepsilon\delta}}\mathsf{U}$. We list all global and local costs in Table 1.

Comparing (1) and (3), we see that the upper bounds we achieve for subgraph detection are the same as those of [LMS13] (up to polylogarithmic factors), as a function of the global and local costs. Furthermore, our expressions for global and local costs are almost identical to those of [LMS13]. The only exception is that, depending on some sparsity condition, namely whether $d_{ij}\min(r_i, r_j) < \max(r_i, r_j)$, the local cost of LoadEdge(i,j) has two different expressions in [LMS13]: either $\sqrt{r_i r_j}$ (when the condition holds) or $\max(r_i, r_j)$ (otherwise). Therefore, ignoring polylogarithmic factors, our total complexity is the same as that of [LMS13] or potentially better in some cases, because of our potentially better edge-local cost, though we know of no particular graph for which we have asymptotically better complexity. We remark that we believe that this improvement in edge-local cost could also be made in the learning graph construction of [LMS13], however, it isn't immediately clear how to do so in that setting, whereas in the quantum walk setting it is natural and immediate.

5 Notes on other learning graphs

In [Bel12a], a learning graph algorithm for graph collision is presented. We note that this is, in fact, based on an unpublished quantum walk algorithm due to Ambainis [Bel12a], which has the same complexity, up to log factors.

As to the possibility of reproducing the recent learning graph upper bound for k-distinctness [Bel12a], while this upper bound is achieved via a method that deviates significantly from the original learning graph framework, we do believe that we should be able to construct a quantum walk algorithm with matching complexity.

Acknowledgments

References

[ACR$^+$10] A. Ambainis, A. M. Childs, B. W. Reichardt, R. Špalek, and S. Zhang. Any AND-OR formula of size N can be evaluated in time $N^{1/2+o(1)}$ on a quantum computer. *SIAM Journal on Computing*, 39(6):2513–2530, 2010.

[Amb00] A. Ambainis. Quantum lower bounds by quantum arguments. In *Proceedings of 32nd Annual ACM Symposium on Theory of Computing*, pages 636–643, 2000.

[Amb04] A. Ambainis. Quantum walk algorithm for element distinctness. In *Proceedings of 45th IEEE Symposium on Foundations of Computer Science*, pages 22–31, 2004.

[AS04] S. Aaronson and Y. Shi. Quantum lower bounds for the collision and element distinctness problems. *Journal of the ACM*, 51:595–605, 2004.

[BBC$^+$01] R. Beals, H. Buhrman, R. Cleve, M. Mosca, and R. de Wolf. Quantum lower bounds by polynomials. *Journal of the ACM*, 48, 2001.

[BBHT98] M. Boyer, G. Brassard, P. Høyer, and A. Tapp. Tight bounds on quantum searching. *Fortschritte der Physik*, 46(4-5):493–505, 1998.

[BDH$^+$05] H. Buhrman, C. Dürr, M. Heiligman, P. Høyer, M. Santha, F. Magniez, and R. de Wolf. Quantum algorithms for Element Distinctness. *SIAM Journal of Computing*, 34(6):1324–1330, 2005.

[Bel12a] A. Belovs. Learning-graph-based quantum algorithm for k-distinctness. In *Prooceedings of 53rd Annual IEEE Symposium on Foundations of Computer Science*, 2012.

[Bel12b] A. Belovs. Span programs for functions with constant-sized 1-certificates. In *Proceedings of 44th Symposium on Theory of Computing Conference*, pages 77–84, 2012.

[BL11] A. Belovs and T. Lee. Quantum algorithm for k-distinctness with prior knowledge on the input. Technical report, arXiv, 2011. `arXiv:1108.3022`.

[BR12] A. Belovs and B. Reichardt. Span programs and quantum algorithms for st-connectivity and claw detection. In *Proceedings of 20th European Symposium on Algorithms*, 2012.

[BŠ06] H. Buhrman and R. Špalek. Quantum verification of matrix products. In *Proceedings of 17th ACM-SIAM Symposium On Discrete Algorithms*, pages 880–889, 2006.

[CK11]   A. Childs and R. Kothari. Quantum query complexity of minor-closed graph properties. In *Proceedings of 28th Symposium on Theoretical Aspects of Computer Science*, pages 661–672, 2011.

[Gal12]   F. Le Gall. Improved output-sensitive quantum algorithms for Boolean matrix multiplication. In *Proceedings of 23rd ACM-SIAM Symposium On Discrete Algorithms*, pages 1464–1476, 2012.

[Gro96]   L. K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of 28th ACM Symposium on Theory of Computing*, pages 212–219, 1996.

[JLR11]   S. Janson, T. Luczak, and A. Rucinski. *Random Graphs*. Wiley Series in Discrete Mathematics and Optimization. John Wiley & Sons, 2011.

[LMR$^+$11] T. Lee, R. Mittal, B. Reichardt, R. Spalek, and M. Szegedy. Quantum query complexity of state conversion. In *Proceedings of 52nd Annual IEEE Symposium on Foundations of Computer Science*, pages 344–353, 2011.

[LMS11]   T. Lee, F. Magniez, and M. Santha. A learning graph based quantum query algorithm for finding constant-size subgraphs. Technical report, arXiv, 2011. `arXiv:1109.5135`.

[LMS13]   T. Lee, F. Magniez, and M. Santha. Improved quantum query algorithms for triangle finding and associativity testing. In *Proceedings of 24th ACM-SIAM Symposium On Discrete Algorithms*, 2013.

[MNRS11]   F. Magniez, A. Nayak, J. Roland, and M. Santha. Search via quantum walk. *SIAM Journal on Computing*, 40(1):142–164, 2011.

[MNRS12]   F. Magniez, A. Nayak, P. Richter, and M. Santha. On the hitting times of quantum versus random walks. *Algorithmica*, 63(1):91–116, 2012.

[MSS07]   F. Magniez, M. Santha, and M. Szegedy. Quantum algorithms for the triangle problem. *SIAM Journal on Computing*, 37(2):413–424, 2007.

[NM07]   A. Nayak and F. Magniez. Quantum complexity of testing group commutativity. *Algorithmica*, 48, 2007.

[Rei09]   B. Reichardt. Span programs and quantum query complexity: The general adversary bound is nearly tight for every Boolean function. In *Proccedings of 50th Annual Symposium on Foundations of Computer Science*, 2009.

[Rei11]   B. Reichardt. Reflections for quantum query algorithms. In *Proceedings of 22nd ACM-SIAM Symposium on Discrete Algorithms*, pages 560–569, 2011.

[SKW03]   N. Shenvi, J. Kempe, and K. B. Whaley. Quantum random-walk search algorithm. *Physical Review A*, 67, 2003. Article no. 052307.

[Sze04]   M. Szegedy. Quantum speed-up of Markov chain based algorithms. In *Proceedings of 45th Annual IEEE Symposium on Foundations of Computer Science*, pages 32–41, 2004.

[Zhu11]   Y. Zhu. Quantum query of subgraph containment with constant-sized certificates, 2011. `arXiv:1109.4165v1`.

A Graph collision subroutine

In this section, we show how to compute graph collision on a bipartite graph with classes of size r_1 and r_2 in quantum query complexity $O\left((r_1 r_2)^{1/3}\right)$. The algorithm is a straightforward generalization of the standard graph collision algorithm of [MSS07], based on the element distinctness algorithm of [Amb04]. Note that when $r_1 = r_2$, the well-known $O\left(r_1^{2/3}\right)$ upper bound is achieved.

Lemma A.1. *The quantum query complexity of graph collision on a bipartite graph with classes of size r_1 and r_2, when $r_1 \leq r_2 \leq r_1^2$, is in $O\left((r_1 r_2)^{1/3}\right)$.*

Proof. Let G be a bipartite graph on classes R_1 and R_2 with $|R_1| = r_1$ and $|R_2| = r_2$. We proceed as in the standard graph collision algorithm: let $m \leq r_1$ be a parameter to be optimized later. We walk on the product graph $J(r_1, m) \times J(r_2, m)$, where a state S_1, S_2 corresponds to a pair of subsets; S_1 is an m-subset of R_1 and S_2 is an m-subset of R_2. Two states S_1, S_2 and S_1', S_2' are adjacent in $J(r_1, m) \times J(r_2, m)$ if S_1 is adjacent to S_1' in $J(r_1, m)$ and S_2 is adjacent to S_2' in $J(r_2, m)$.

The data associated with a state S_1, S_2 is the set of query values for vertices $i \in S_1 \cup S_2$. Thus, the setup cost is $\mathsf{S} = O(m)$, and the update cost is $\mathsf{U} = O(1)$. A state is marked if there is a pair $i_1 \in S_1$ and $i_2 \in S_2$ such that i_1 is adjacent to i_2 and both are marked. Since the data contains query values for both i and i', checking costs $\mathsf{C} = 0$ queries. The proportion of marked states is at least $\varepsilon = \Omega\left(\frac{m^2}{r_1 r_2}\right)$. The spectral gap of $J(r_1, m) \times J(r_2, m)$ is $\delta = \Omega\left(\frac{1}{m}\right)$. Thus, the query complexity is

$$O\left(m + \frac{\sqrt{r_1 r_2}}{m} \sqrt{m}\right),$$

which is optimized by setting $m = (r_1 r_2)^{1/3}$ (which is smaller than r_1 when $r_2 \leq r_1^2$), yielding quantum query complexity $O\left((r_1 r_2)^{1/3}\right)$. □